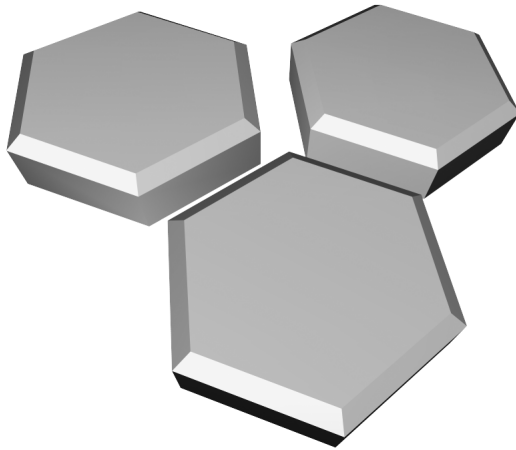


# Das VB.NET Codebook





J. Fuchs. A. Barchfeld

# Das VB.NET 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  
06 05 04

ISBN 3-8273-2007-1

© 2004 by Addison-Wesley Verlag,  
ein Imprint der Pearson Education Deutschland GmbH  
Martin-Kollar-Straße 10–12, D-81829 München/Germany  
Alle Rechte vorbehalten

Korrektorat: G & U Technische Dokumentation GmbH, Flensburg  
Lektorat: Frank Eller, [feller@pearson.de](mailto:feller@pearson.de)  
Herstellung: Monika Weiher, [mweiher@pearson.de](mailto:mweiher@pearson.de)  
Satz: reemers publishing services gmbh, Krefeld – gesetzt aus der Minion  
Umschlaggestaltung: Marco Lindenbeck, [mlindenbeck@webwo.de](mailto:mlindenbeck@webwo.de)  
Druck und Verarbeitung: Kösel, Kempten ([www.Koeselbuch.de](http://www.Koeselbuch.de))

Printed in Germany



# Inhaltsverzeichnis

|                                                                                        |           |
|----------------------------------------------------------------------------------------|-----------|
| <b>Teil I Einführung</b>                                                               | <b>13</b> |
| <b>Vorwort</b>                                                                         | <b>15</b> |
| 1 Die Autoren                                                                          | 16        |
| 2 Homepage für dieses Buch                                                             | 16        |
| <b>Einleitung</b>                                                                      | <b>17</b> |
| 3 Von gestern bis heute                                                                | 17        |
| 4 Was sich mit Visual Basic .NET realisieren lässt und was nicht                       | 19        |
| 5 Inhalt des Buches                                                                    | 19        |
| <b>Teil II Rezepte</b>                                                                 | <b>31</b> |
| <b>Basics</b>                                                                          | <b>33</b> |
| 1 Zahlen-, Zeichen- und String-Literale                                                | 33        |
| 2 Ganzzahlen dual, oktal oder hexadezimal darstellen                                   | 35        |
| 3 String mit dualer, oktaler oder hexadezimaler Darstellung in Zahlenwert wandeln      | 35        |
| 4 Zahlenwerte formatieren                                                              | 35        |
| 5 Positive und negative Zahlen unterschiedlich formatieren                             | 36        |
| 6 Zusammengesetzte Formatierungen                                                      | 37        |
| 7 Format-Provider für eigene Klassen definieren                                        | 38        |
| 8 Ausgaben in länderspezifischen Formaten                                              | 39        |
| 9 Informationen zu länderspezifischen Einstellungen abrufen                            | 41        |
| 10 Größter und kleinster Wert eines numerischen Datentyps                              | 44        |
| 11 Berechnen der signifikanten Vorkommastellen                                         | 45        |
| 12 Lange einzeilige Texte umbrechen                                                    | 47        |
| 13 Zahlenwerte kaufmännisch runden                                                     | 48        |
| 14 Überprüfen, ob ein Bit in einem Integer-Wert gesetzt ist                            | 49        |
| 15 Bit in einem Integer-Wert setzen                                                    | 50        |
| 16 Bit in einem Integer-Wert löschen                                                   | 51        |
| 17 Bit in einem Integer-Wert einen bestimmten Zustand zuweisen                         | 51        |
| 18 Bit in einem Integer-Wert umschalten (togglen)                                      | 52        |
| 19 Boolean-Array aus Bit-Informationen eines Integer-Wertes erzeugen                   | 52        |
| 20 Integer-Wert aus Bit-Informationen eines Boolean-Arrays zusammensetzen              | 53        |
| 21 Gesetzte Bits eines Integer-Wertes abfragen                                         | 54        |
| 22 Nicht gesetzte Bits eines Integer-Wertes abfragen                                   | 55        |
| 23 Konvertierungen zwischen 8-Bit, 16-Bit, 32-Bit und 64-Bit Datentypen                | 56        |
| 24 Basistyp für Enumeration festlegen                                                  | 59        |
| 25 Enum-Werte ein- und ausgeben                                                        | 60        |
| 26 Bezeichner und Werte eines Enum-Typs abfragen                                       | 62        |
| 27 Prüfen, ob ein Zahlenwert als Konstante in einer Enumeration definiert ist          | 63        |
| 28 Prüfen, ob ein bestimmter Enumerationswert in einer Kombination von Werten vorkommt | 63        |
| 29 Objekte eigener Klassen vergleichbar und sortierbar machen                          | 64        |
| 30 Binäre Suche in Arrays und Auflistungen                                             | 67        |

|                             |                                                                                         |            |
|-----------------------------|-----------------------------------------------------------------------------------------|------------|
| 31                          | Strings in Byte-Arrays konvertieren und vice versa                                      | 69         |
| 32                          | Ersatz für unveränderliche (konstante) Zeichenketten-Arrays                             | 71         |
| <b>Datum und Zeit</b>       |                                                                                         | <b>73</b>  |
| 33                          | Umgang mit Datum und Uhrzeit                                                            | 73         |
| 34                          | Schaltjahre                                                                             | 75         |
| 35                          | Wochentag berechnen                                                                     | 76         |
| 36                          | Beginn einer Kalenderwoche berechnen                                                    | 76         |
| 37                          | Anzahl der Kalenderwochen eines Jahres bestimmen                                        | 77         |
| 38                          | Berechnung der Kalenderwoche zu einem vorgegebenen Datum                                | 78         |
| 39                          | Berechnung des Osterdatums                                                              | 80         |
| 40                          | Berechnung der deutschen Feiertage                                                      | 81         |
| 41                          | Darstellung der Feiertage im Kalender-Steuerelement                                     | 93         |
| 42                          | Gregorianisches Datum in Julianische Tageszählung                                       | 95         |
| 43                          | Julianische Tageszählung in Gregorianisches Datum                                       | 97         |
| 44                          | Datum und Uhrzeit im ISO 8601-Format ausgeben und einlesen                              | 99         |
| <b>Anwendungen</b>          |                                                                                         | <b>103</b> |
| 45                          | Anwendungskonfigurationsdatei erstellen und bearbeiten                                  | 103        |
| 46                          | Globale Werte aus der .config-Datei lesen                                               | 104        |
| 47                          | Zusätzliche Konfigurationsdatei-Sektionen mit Attributwerten lesen                      | 105        |
| 48                          | Zusätzliche Konfigurationsdatei-Sektionen mit Key/Value-Einträgen lesen                 | 106        |
| 49                          | Knoten beliebiger Struktur aus einer Konfigurationsdatei lesen                          | 107        |
| 50                          | Konfigurationsdatei zur Laufzeit ändern                                                 | 110        |
| 51                          | Eine eigene Konfigurationsdatei für Ihre Anwendung                                      | 110        |
| 52                          | Konfigurationsmanager und -daten trennen                                                | 116        |
| 53                          | Verhindern, dass mehrere Instanzen einer Anwendung gleichzeitig gestartet werden können | 121        |
| 54                          | Zentrales Exception-Handling                                                            | 124        |
| <b>GDI+ Zeichnen</b>        |                                                                                         | <b>127</b> |
| 55                          | Outline-Schrift erzeugen                                                                | 127        |
| 56                          | Text im Kreis ausgeben und rotieren lassen                                              | 128        |
| 57                          | Schriftzug mit Hintergrundbild füllen                                                   | 133        |
| 58                          | Transparente Schriftzüge über ein Bild zeichnen                                         | 136        |
| 59                          | Blockschrift für markante Schriftzüge                                                   | 138        |
| 60                          | Text mit versetztem Schatten zeichnen                                                   | 140        |
| 61                          | Schriftzug perspektivisch verzerren                                                     | 142        |
| 62                          | Font-Metrics zur exakten Positionierung von Schriftzügen ermitteln                      | 144        |
| 63                          | Schatten durch Matrix-Operationen erzeugen                                              | 147        |
| 64                          | Rechtecke mit abgerundeten Ecken zeichnen                                               | 150        |
| 65                          | 3D-Schriften erzeugen                                                                   | 152        |
| 66                          | 3D- und Beleuchtungseffekte auf strukturierten Hintergrundbildern                       | 154        |
| <b>GDI+ Bildbearbeitung</b> |                                                                                         | <b>159</b> |
| 67                          | Bilder zeichnen                                                                         | 159        |
| 68                          | Bildausschnitt zoomen                                                                   | 162        |
| 69                          | Basisklasse für eine Dia-Show                                                           | 166        |
| 70                          | Horizontal und vertikal überblenden                                                     | 171        |
| 71                          | Diagonal überblenden                                                                    | 177        |
| 72                          | Elliptische Überblendung                                                                | 179        |

|    |                                                                           |     |
|----|---------------------------------------------------------------------------|-----|
| 73 | Überblendung durch zufällige Mosaik-Muster                                | 181 |
| 74 | Überblendung durch Transparenz                                            | 183 |
| 75 | Bilder verzerrungsfrei maximieren                                         | 186 |
| 76 | Blockierung der Bilddatei verhindern                                      | 191 |
| 77 | Ordnerauswahl mit Miniaturenansicht der enthaltenen Bilder                | 191 |
| 78 | Drehen und Spiegeln von Bildern                                           | 205 |
| 79 | Encoder für verschiedene Dateiformate zum Speichern von Bildern ermitteln | 207 |
| 80 | Bilder im JPEG-Format abspeichern                                         | 208 |
| 81 | Bilder im GIF-Format speichern                                            | 209 |
| 82 | Thumbnails für Web-Seiten erstellen                                       | 210 |
| 83 | Invertieren eines Bildes                                                  | 212 |
| 84 | Farbbild in Graustufenbild wandeln                                        | 213 |
| 85 | Weitere Bildmanipulationen mit Hilfe der ColorMatrix                      | 215 |
| 86 | Bitmapdaten in ein Array kopieren                                         | 216 |
| 87 | Array in Bitmap kopieren                                                  | 217 |
| 88 | Allgemeiner Schärfefilter                                                 | 219 |
| 89 | Schärfe nach Gauss                                                        | 222 |
| 90 | Schärfe mittels Sobel-Filter                                              | 224 |
| 91 | Schärfe mittels Laplace-Filter                                            | 225 |
| 92 | Der Boxcar Unschärfefilter                                                | 226 |
| 93 | Adaptive Schärfe                                                          | 227 |

## **Windows Forms** **229**

|     |                                                                   |     |
|-----|-------------------------------------------------------------------|-----|
| 94  | Fenster ohne Titelleiste anzeigen                                 | 229 |
| 95  | Fenster ohne Titelleiste verschieben                              | 229 |
| 96  | Halbtransparente Fenster                                          | 231 |
| 97  | Unregelmäßige Fenster und andere Transparenzeffekte               | 232 |
| 98  | Startbildschirm                                                   | 234 |
| 99  | Dialoge kapseln                                                   | 238 |
| 100 | Gekapselter Dialog mit Übernehmen-Schaltfläche                    | 243 |
| 101 | Dialog-Basisklasse                                                | 245 |
| 102 | Validierung der Benutzereingaben                                  | 249 |
| 103 | Screenshots erstellen                                             | 254 |
| 104 | TextViewer-Klasse                                                 | 259 |
| 105 | RTFTextViewer-Klasse                                              | 261 |
| 106 | PictureViewer-Klasse                                              | 263 |
| 107 | HTML-Viewer                                                       | 266 |
| 108 | Drag&Drop-Operationen aus anderen Anwendungen ermöglichen         | 269 |
| 109 | Analyseprogramm für Drag&Drop-Operationen aus anderen Anwendungen | 271 |
| 110 | Anzeigen von Daten aus der Zwischenablage                         | 277 |
| 111 | Exportieren von Daten über die Zwischenablage                     | 278 |
| 112 | Exportieren von Daten über Drag&Drop                              | 280 |
| 113 | Windows XP Darstellungsformate nutzen                             | 282 |
| 114 | Eingabetaste in TextBox abfangen                                  | 284 |
| 115 | Pfade so kürzen, dass sie in den verfügbaren Bereich passen       | 286 |

## **Windows Controls** **293**

|     |                                                   |     |
|-----|---------------------------------------------------|-----|
| 116 | Ersatz für VB6-Control-Arrays                     | 293 |
| 117 | ListBox-Items selber zeichnen                     | 296 |
| 118 | Mehrspaltige DropDown-Liste (ComboBox)            | 299 |
| 119 | Basisklassen für selbst definierte Steuerelemente | 302 |

|     |                                                                                          |            |
|-----|------------------------------------------------------------------------------------------|------------|
| 120 | ComboBox mit Auto-Vervollständigen-Funktion                                              | 303        |
| 121 | Benutzersteuerelement als Container für andere Steuerelemente                            | 304        |
| 122 | Scroll-Balken eines Benutzersteuerelementes im Design-Mode verfügbar machen              | 305        |
| 123 | Benutzersteuerelemente und die Text-Eigenschaft                                          | 305        |
| 124 | PanelGroupBox – ein Steuerelement für alle Fälle                                         | 306        |
| 125 | Einem Benutzersteuerelement ein Icon zuweisen                                            | 312        |
| 126 | Transparenz eines Steuerelementes im Designer einstellen                                 | 313        |
| 127 | Abfangen von Windows-Nachrichten                                                         | 313        |
| 128 | Steuerelement für Verzeichnisauswahl                                                     | 319        |
| 129 | Ein Windows-Explorer-Steuerelement im Eigenbau                                           | 330        |
| 130 | ListView des Explorer-Steuerelementes sortieren                                          | 334        |
| 131 | FolderBrowser-Steuerelement mit zusätzlichen CheckBoxen zum Aufbau von Verzeichnislisten | 341        |
| 132 | Benutzerdefinierte Steuerelemente mit nicht rechteckigem Umriss                          | 351        |
| 133 | Mausposition in kreisförmigen Steuerelementen in Winkel umrechnen                        | 353        |
| 134 | Maus-Ereignisse zur Entwurfszeit abfangen                                                | 356        |
| 135 | Ein Steuerelement zur grafischen Anzeige von Zeitbereichen programmieren                 | 358        |
| 136 | Neue Zeitscheiben zur Laufzeit mit der Maus hinzufügen                                   | 368        |
| 137 | Nachrichten verschicken mit SendMessage                                                  | 372        |
| 138 | Zeilen ermitteln, wie sie in einer mehrzeiligen TextBox dargestellt werden               | 373        |
| 139 | Anzahl der Zeilen einer mehrzeiligen TextBox ermitteln                                   | 375        |
| 140 | Zeilenindex aus Zeichenindex ermitteln (mehrzeilige TextBox)                             | 376        |
| 141 | Index des ersten Zeichens einer Zeile ermitteln (mehrzeilige TextBox)                    | 377        |
| 142 | Index der ersten sichtbaren Zeile einer mehrzeiligen TextBox bestimmen                   | 377        |
| 143 | Zeichenindex aus Grafikkoordinaten berechnen (mehrzeilige TextBox)                       | 378        |
| 144 | Koordinate eines Zeichens ermitteln (mehrzeilige TextBox)                                | 379        |
| 145 | Mehrzeilige TextBox per Code auf- und abwärts scrollen                                   | 380        |
| 146 | Tabulatorpositionen in einer mehrzeiligen TextBox setzen                                 | 381        |
|     | <b>Eigenschaftsfenster (PropertyGrid)</b>                                                | <b>383</b> |
| 147 | Grundlegende Attribute                                                                   | 383        |
| 148 | Eigenschaften mehrerer Objekte gleichzeitig anzeigen                                     | 388        |
| 149 | Abfangen ungültiger Werte                                                                | 389        |
| 150 | Standardwerte für Eigenschaften                                                          | 390        |
| 151 | Festlegen einer Standard-Eigenschaft                                                     | 391        |
| 152 | Eigenschaften gegen Veränderungen im PropertyGrid-Control schützen                       | 391        |
| 153 | Enumerationswerte kombinieren                                                            | 392        |
| 154 | Geschachtelte expandierbare Eigenschaften                                                | 393        |
| 155 | DropDown-Liste mit Standardwerten für Texteeigenschaften                                 | 397        |
| 156 | Visualisierung von Eigenschaftswerten mit Miniaturbildern                                | 399        |
| 157 | Einen eigenen DropDown-Editor anzeigen                                                   | 401        |
| 158 | Eigenschaften über einen modalen Dialog bearbeiten                                       | 405        |
| 159 | Datei öffnen-Dialog für Eigenschaften bereitstellen                                      | 410        |
| 160 | Auflistungen anzeigen und bearbeiten                                                     | 411        |
| 161 | Aktionen über Hyperlink-Tasten anbieten                                                  | 419        |
| 162 | Eigenschaften dynamisch erstellen und hinzufügen                                         | 423        |
| 163 | Eigenschaften in unterschiedlichen Sprachen anzeigen (Lokalisierung)                     | 431        |
| 164 | Neue Tab-Flächen hinzufügen                                                              | 434        |

|                                                                                    |            |
|------------------------------------------------------------------------------------|------------|
| <b>Dateisystem</b>                                                                 | <b>439</b> |
| 165 System-Verzeichnisse mit .NET                                                  | 439        |
| 166 Anwendungs-/Bibliotheksname des laufenden Prozesses                            | 442        |
| 167 Existenz eines Verzeichnisses                                                  | 443        |
| 168 Verzeichnis erstellen                                                          | 445        |
| 169 Verzeichnis löschen                                                            | 446        |
| 170 Verzeichnis umbenennen/verschieben                                             | 446        |
| 171 Verzeichnis kopieren                                                           | 448        |
| 172 Verzeichnisgröße mit Unterverzeichnissen                                       | 450        |
| 173 Existenz einer bestimmten Datei                                                | 453        |
| 174 8.3 Dateinamen                                                                 | 454        |
| 175 Datei umbenennen/verschieben                                                   | 456        |
| 176 Datei kopieren                                                                 | 457        |
| 177 Dateiversion feststellen                                                       | 458        |
| 178 Dateigröße                                                                     | 460        |
| 179 Dateivergleich                                                                 | 462        |
| 180 Temporäre Dateinamen                                                           | 464        |
| 181 Datei in mehreren Verzeichnissen suchen am Beispiel der Verzeichnisse von PATH | 466        |
| 182 Dateiinformationen mit File System Object                                      | 468        |
| 183 Laufwerksinformationen mit FSO                                                 | 470        |
| 184 Delimited-Dateien nach XML transformieren                                      | 472        |
| 185 Überwachung des Dateisystems                                                   | 475        |
| 186 Datei-Attribute                                                                | 478        |
| 187 Bestandteile eines Pfads ermitteln                                             | 482        |
| 188 Absolute und gekürzte (kanonische) Pfade ermitteln                             | 483        |
| 189 Relativen Pfad ermitteln                                                       | 484        |
| 190 Icons und Typ einer Datei ermitteln                                            | 486        |
| 191 Dateien kopieren, verschieben, umbenennen und löschen mit SHFileOperation      | 490        |
| <b>Netzwerk</b>                                                                    | <b>501</b> |
| 192 IPv4-Adressen nach IPv6 umrechnen                                              | 501        |
| 193 IPv6-Adressen nach IPv4 umrechnen                                              | 503        |
| 194 IP-Adresse eines Rechners                                                      | 504        |
| 195 Netzwerkadapter auflisten                                                      | 506        |
| 196 Freigegebene Laufwerke anzeigen                                                | 508        |
| 197 Web-Service                                                                    | 510        |
| 198 Internet Explorer starten                                                      | 513        |
| <b>System</b>                                                                      | <b>515</b> |
| 199 WMI-Namensräume                                                                | 515        |
| 200 WMI-Klassen                                                                    | 516        |
| 201 Ist WMI installiert?                                                           | 517        |
| 202 BIOS-Informationen                                                             | 518        |
| 203 Computer-Modell                                                                | 523        |
| 204 Letzter Boot-Status                                                            | 524        |
| 205 Sommer-/Winterzeit                                                             | 525        |
| 206 Computerdomäne                                                                 | 525        |
| 207 Domänenrolle                                                                   | 526        |
| 208 Benutzername                                                                   | 527        |
| 209 Monitorauflösung                                                               | 529        |
| 210 Der Monitor-Typ                                                                | 529        |

|     |                                                |     |
|-----|------------------------------------------------|-----|
| 211 | Auflösung in Zoll                              | 530 |
| 212 | Logische Laufwerke mit WMI                     | 531 |
| 213 | Physikalische Platten                          | 536 |
| 214 | Installierte Programme                         | 539 |
| 215 | Programm über Namen starten                    | 542 |
| 216 | Programm über Datei starten                    | 543 |
| 217 | Parameterübergabe per Befehlszeile             | 544 |
| 218 | Systemprozesse mit WMI                         | 546 |
| 219 | Systemprozesse mit .System.Diagnostics         | 549 |
| 220 | Liste aller Dienste                            | 553 |
| 221 | Dienst starten                                 | 554 |
| 222 | Dienst anhalten                                | 557 |
| 223 | Dienst fortsetzen                              | 559 |
| 224 | Dienst stoppen                                 | 560 |
| 225 | Prozess abbrechen (»killen«)                   | 561 |
| 226 | Leistungsüberwachung/Performance Counter       | 563 |
| 227 | Registry-Einträge abfragen                     | 566 |
| 228 | Registry-ey anlegen                            | 568 |
| 229 | Registry-Key löschen                           | 569 |
| 230 | Informationen zum installierten Betriebssystem | 570 |
| 231 | Prozessorgeschwindigkeit                       | 575 |
| 232 | Prozessorauslastung                            | 577 |
| 233 | Bitbreite des Prozessors                       | 577 |
| 234 | Prozessor-Informationen                        | 578 |
| 235 | SMTP – eMail                                   | 578 |
| 236 | Fax senden                                     | 580 |
| 237 | Logon-Sessions mit XP                          | 581 |

## **Datenbanken 583**

|     |                                          |     |
|-----|------------------------------------------|-----|
| 238 | Erreichbare SQL-Server                   | 584 |
| 239 | Default-Anmeldung am SQL-Server          | 585 |
| 240 | NT-Security-Anmeldung am SQL-Server      | 587 |
| 241 | SQL-Server-Anmeldung                     | 587 |
| 242 | Datenbanken eines Servers                | 588 |
| 243 | Datenbank festlegen                      | 589 |
| 244 | Tabellen einer Datenbank                 | 589 |
| 245 | Felder einer Tabelle                     | 590 |
| 246 | Einfaches Backup einer Datenbank         | 590 |
| 247 | Einfaches Zurücksichern einer Datenbank  | 591 |
| 248 | Erstellen eines Backup-Devices           | 592 |
| 249 | Datensicherung auf ein Backup-Device     | 594 |
| 250 | Liste der Backup-Devices                 | 596 |
| 251 | Rücksicherung von einem Backup-Device    | 597 |
| 252 | Erstellen einer Datenbank                | 599 |
| 253 | Erstellen eines T-SQL-Datenbank-Skriptes | 601 |
| 254 | Erstellen eines Jobauftrages             | 604 |
| 255 | Auflistung der vorhandenen Jobaufträge   | 608 |
| 256 | Tabellenindizes                          | 610 |
| 257 | Bilder in Tabellen abspeichern           | 612 |
| 258 | Datagrid füllen                          | 614 |
| 259 | MDAC-Version ermitteln                   | 616 |
| 260 | Excel als Datenbank abfragen             | 617 |

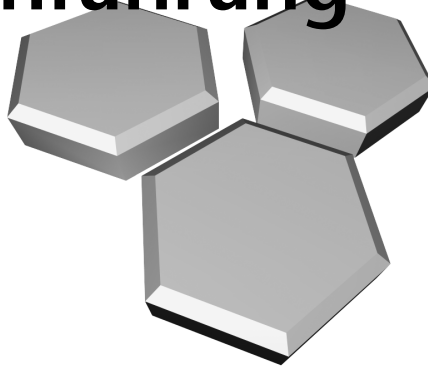
|                                                                           |                |
|---------------------------------------------------------------------------|----------------|
| <b>XML</b>                                                                | <b>621</b>     |
| 261 Schreiben von XML-Dateien mit dem XmlTextWriter                       | 621            |
| 262 Lesen von XML-Dateien mit dem XmlTextReader                           | 625            |
| 263 Bilder und andere Binärdaten in XML-Dateien speichern                 | 629            |
| 264 Bilder und andere Binärdaten aus XML-Dateien lesen                    | 632            |
| 265 XML-Dateien lesen mit XmlDocument                                     | 633            |
| 266 Hinzufügen, Entfernen und Ändern von Knoten mit XmlDocument           | 636            |
| 267 XmlDocument mit XPath-Ausdrücken durchsuchen                          | 638            |
| 268 XPath-Abfragen und XML-Namespaces                                     | 643            |
| 269 Schnellere Suche mit XPathDocument und XPathNavigator                 | 646            |
| 270 XmlView-Steuerelement zur strukturierten Darstellung von XML-Dateien  | 650            |
| 271 Nachrichten aus RSS-Kanälen aufbereiten                               | 657            |
| 272 Das Wichtigste der Tagesschau im UserControl                          | 664            |
| 273 XML-Dateien validieren                                                | 668            |
| 274 XSL-Transformationen                                                  | 674            |
| 275 XSL-Transformation mit Parametern                                     | 677            |
| 276 Einer XSL-Transformation zusätzliche Funktionen bereitstellen         | 678            |
| 277 Parallelbetrieb von XmlDataDocument und DataSet                       | 681            |
| 278 Klassenhierarchie aus XML-Schema erzeugen                             | 684            |
| 279 Serialisierung mit Hilfe der Klasse XmlSerializer                     | 685            |
| 280 Unbekannte XML-Inhalte bei der Deserialisierung mit dem XmlSerializer | 691            |
| 281 Serialisierung mit Hilfe der Klasse SoapFormatter                     | 692            |
| <br><b>Wissenschaftliche Berechnungen und Kurvendiagramme</b>             | <br><b>697</b> |
| 282 Gleitkommazahlen vergleichen                                          | 697            |
| 283 Typsichere Maßeinheiten                                               | 699            |
| 284 Definition von Längenmaßen                                            | 706            |
| 285 Definition von Flächenmaßen                                           | 708            |
| 286 Definition von Geschwindigkeiten                                      | 710            |
| 287 Definition von Zeiten                                                 | 712            |
| 288 Definition von Temperaturen                                           | 713            |
| 289 Universeller Umrechner für Maßeinheiten                               | 716            |
| 290 Schnittstelle für Datenquellen mit typsicheren physikalischen Werten  | 721            |
| 291 Skalierung für Diagramme berechnen                                    | 723            |
| 292 Einfaches T-Y-Diagramm mit statischen Werten                          | 727            |
| 293 Kontinuierliches T-Y-Diagramm mit dynamischen Werten                  | 741            |
| 294 Die Zahl p                                                            | 748            |
| <br><b>Verschiedenes</b>                                                  | <br><b>751</b> |
| 295 Sound abspielen über API-Funktionen                                   | 751            |
| 296 Sound abspielen über DirectX                                          | 753            |
| 297 Trace- und Debug-Ausgaben über Config-Datei steuern                   | 755            |
| 298 Debug- und Trace-Ausgaben an eine TextBox weiterleiten                | 757            |
| 299 Debug- und Trace-Ausgaben in einer Datei speichern                    | 759            |
| 300 Debug- und Trace-Ausgaben an das Eventlog weiterleiten                | 761            |
| 301 Eigene EventLogs für die Ereignisanzeige anlegen und beschreiben      | 762            |
| 302 EventLog überwachen und lesen                                         | 765            |
| 303 Leistungsindikatoren anlegen und mit Daten versorgen                  | 766            |
| 304 Zeiten mit hoher Auflösung messen                                     | 771            |
| 305 API-Fehlermeldungen aufbereiten                                       | 773            |

|                                                                 |            |
|-----------------------------------------------------------------|------------|
| <b>Teil III Anhang</b>                                          | <b>775</b> |
| <b>Visual Basic .NET</b>                                        | <b>777</b> |
| 1 Klassen – Referenzen – Objekte                                | 777        |
| 2 Strukturen (Wertetypen)                                       | 779        |
| 3 (Instanz-)Methoden                                            | 780        |
| 4 Statische Methoden und statische Variablen                    | 781        |
| 5 Module                                                        | 782        |
| 6 Eigenschaften (Properties)                                    | 783        |
| 7 Vererbung                                                     | 784        |
| 8 Arrays                                                        | 789        |
| 9 Listen                                                        | 798        |
| 10 CLS-Kompatibilität                                           | 803        |
| <b>Visual Studio</b>                                            | <b>805</b> |
| 11 Texte in der Toolbox zwischenspeichern                       | 805        |
| 12 Standard-Einstellungen für Option Strict                     | 805        |
| 13 Projektweite Imports-Einstellungen                           | 806        |
| 14 Steuerelemente und Fensterklasse im Entwurfsmodus debuggen   | 807        |
| 15 Verknüpfung einer Datei einem Projekt hinzufügen             | 809        |
| 16 Dokumentgliederung anzeigen und nutzen                       | 809        |
| 17 Tabellenansicht einer XML-Datei                              | 809        |
| 18 XML-Schema für vorhandene XML-Datei erstellen und bearbeiten | 811        |
| 19 Navigation über die Klassenansicht                           | 811        |
| <b>Internetquellen</b>                                          | <b>813</b> |
| 20 Websites zu .NET                                             | 813        |
| 21 Newsgroups                                                   | 814        |
| 22 Recherche mit Google                                         | 816        |
| <b>Grundlagen weiterer Technologien</b>                         | <b>819</b> |
| 23 Kurzer Überblick über WMI                                    | 819        |
| 24 XML DOM-Grundlagen                                           | 822        |
| <b>API-Funktionen</b>                                           | <b>825</b> |
| <b>Stichwortverzeichnis</b>                                     | <b>831</b> |



# TEIL I

## Einführung





# Vorwort

Mit .NET hat Microsoft eine umfassende und richtungsweisende Plattform für die Programmierung von Anwendungssystemen geschaffen. Die Realisierung komplexer Aufgaben wird durch .NET erheblich vereinfacht und vereinheitlicht. Eine umfangreiche Klassenbibliothek in Form eines Frameworks bietet vielfältige Möglichkeiten, die von verschiedenen Programmiersprachen gleichermaßen genutzt werden können. Um auch mit Visual Basic von diesen Möglichkeiten Gebrauch machen zu können, musste die Sprache grundlegend geändert werden.

Daher ist der Umstieg von Visual Basic 6 nach Visual Basic .NET gewaltig. Geblieben ist ein Teil der Syntax, hinzugekommen der Zwang zur Objektorientierten Programmierung und die Verfügbarkeit eines komplexen, fast unüberschaubaren Frameworks, das die meisten Aufgabenbereiche abdeckt. Um die Vorteile von .NET nutzen zu können, ist eine intensive Beschäftigung mit dem .NET Framework unumgänglich.

Das Erlernen der Objektorientierten Programmierung können wir Ihnen nicht abnehmen. Hierzu gibt es inzwischen umfangreiche Literatur. Aber den täglichen Umgang mit dem Framework wollen wir Ihnen in praxisorientierten Rezepten näher bringen. Beginnend mit einfachen Beispielen zu grundlegenden Formatierungen und Umrechnungen bis hin zu komplexen Techniken wie WMI, XML und professionellen Steuerelementen deckt das Buch ein breites Spektrum an Fachgebieten ab.

Trotz der großen Komplexität des .NET Frameworks bietet es zurzeit noch nicht alle Details, die Sie bisher mit Windows-Anwendungen realisieren konnten. Auch hier zeigen wir anhand einiger Beispiele auf, wie fehlende Funktionalitäten im Framework durch andere Techniken ergänzt werden können.

Aus den Praxiserfahrungen im Umgang mit .NET, die wir seit dem Erscheinen der Beta-Version 2001 gesammelt haben, ist dieses Buch entstanden. Die einzelnen Rezepte spiegeln Lösungen für oft gestellte Fragestellungen wider. Neben der einfachen Anwendung der Rezepte können diese auch als Einstiegspunkt für eine tiefer gehende Auseinandersetzung mit neuen Technologien dienen.

Unsere Rezepte bestehen nicht nur aus kommentierten Quelltexten, sondern auch aus detaillierten Erläuterungen zur Aufgabenstellung und deren Lösung. So lassen sie sich auch auf ähnliche Aufgabenstellungen übertragen. Soweit es der Umfang des Buches gestattet, gehen wir im Anhang auch auf weniger bekannte Technologien ein. Die Quelltexte lassen sich schnell mit Hilfe des auf der CD enthaltenen Code-Repositories in eigene Projekte übernehmen. Ergänzend finden Sie alle Projekte zu den einzelnen Rezepten auf der Buch-CD.

Dieses Buch ersetzt in vielen Fällen umfangreiche Recherchen in der Online-Hilfe und/oder im Internet. Informationen, die sonst nur sehr verstreut zu finden sind, werden in verschiedenen Themengruppen zusammengefasst. Dadurch bleibt Ihnen in vielen Fällen eine zeitaufwändige Suche erspart.

Wir danken unseren Frauen für ihr Verständnis und für ihre Geduld mit uns während der Erstellung dieses Buches. Auch danken wir unserem Lektor, Frank Eller, für seine Unterstützung.

## 1 Die Autoren

Dr. Joachim Fuchs ist selbständiger Softwareentwickler und Dozent, seit 2001 mit dem Schwerpunkt »Softwareentwicklung mit .NET«. Sein umfangreiches Wissen gibt er unter anderem in Seminaren, in den Microsoft Newsgroups und in Form von Fachartikeln weiter. Sie erreichen ihn über seine Homepage <http://www.fuechse-online.de/beruflich/index.html>.

Andreas Barchfeld ist IT-Leiter in einem Hamburger Krankenhaus und verfügt über mehrere Jahre Erfahrung als System- und Organisationsprogrammierer im Bereich Windows und Unix. Er beschäftigt sich seit dem Erscheinen der .NET-Beta-Version mit diesem Programmierumfeld. Seine Schwerpunkte in diesem Bereich liegen bei VB, C++ und Datenbanken.

## 2 Homepage für dieses Buch

noch nicht geklärt.

Entweder

<http://www.fuechse-online.de/vbcodebook/index.html>

oder

[www.vbcodebook.de](http://www.vbcodebook.de)

oder

[www.vbcodebook.com](http://www.vbcodebook.com)

# Einleitung

## 3 Von gestern bis heute

Ein kleiner historischer Rückblick soll die Entstehungsgeschichte und die Ziele von Visual Basic .NET erläutern. Keine Angst, wir gehen hier nicht ins Detail, sondern skizzieren nur die Umstände und Intentionen der Entwickler von Basic, Visual Basic und Visual Basic.NET.

### 10 LET A=4

So oder ähnlich kennen viele noch die ersten Basic-Programme. Auf Kleincomputern von Sinclair, Apple, Commodore usw. gab es bereits Ende der 70er die ersten Basic-Interpreter. Basic war eine simple Programmiersprache, die jeder verstehen konnte und die alles Notwendige für den üblichen Bedarf mitbrachte. Während Fortran, Cobol etc. vorwiegend auf Großrechnern zu finden waren, gehörten Basic-Interpreter bald zur Standardausrüstung der Homecomputer.

Mit der unaufhaltsamen Weiterentwicklung höherer Programmiersprachen wie C, C++, Pascal und vielen anderen geriet Basic immer mehr in Vergessenheit, konnte man doch mit den neueren Sprachen wesentlich eleganter und sauberer programmieren als mit dem üblichen Basic-Spaghetti-Code.

### On Error Goto

Als Microsoft begann, mit Windows das Betriebssystem mit einer grafischen Oberfläche auszustatten, war man der festen Überzeugung, dass die Zukunft der Programmierung mit der Sprache C fest verbunden sei. Windows selbst ist zum größten Teil in Standard-C programmiert worden. Auch die ersten Anwendungsprogramme wurden in C geschrieben.

Bald stellte sich heraus, dass C für Windows-Programme, die hauptsächlich ereignisgesteuert sind, zu umständlich war. So kam der Umstieg auf C++, um wenigstens die oft benötigten Vorgehensweisen in Klassenbibliotheken bereitstellen zu können. Aber auch der Umgang mit den (anfangs sehr vielen) Klassenbibliotheken war alles andere als einfach und hielt viele Programmierer davon ab, Windows-Programme zu entwickeln.

Benötigt wurde eine einfache Sprache mit einer integrierten Entwicklungsumgebung, die alles bereithält, um »mal eben« ein Fenster mit ein paar Steuerelementen anzulegen. So wurde Visual Basic ins Leben gerufen. Basic als zugrunde liegende einfache Programmiersprache, verbunden mit einer (zumindest später) komfortablen Entwicklungsumgebung. Schon die Version 3 konnte viele Anwendungsfälle abdecken, die bis dahin eine Domäne der C-Programmierung waren.

Was in VB fehlte, konnte man über den direkten Aufruf von API-Funktionen ergänzen. VB selbst wurde ständig weiterentwickelt. Ab Version 4 kamen erste objektorientierte Ansätze hinzu, die leider auch in der Version 6 noch lange nicht vervollständigt worden sind. Der erste Ansatz, mit Komponenten in Form von VBX-Dateien Visual Basic erweiterbar zu machen, wurde bald wieder eingestellt und durch COM (Component Object Model) ersetzt (und später in ActiveX umbenannt ;-)).

Mit COM war der erste Schritt zu einem Baukastensystem in Form von beliebig zusammenstellbaren Komponenten getan. Mit VB erstellte Komponenten lassen sich auch heute noch in anderen Programmen verwenden.

Bei den Office-Produkten wurde Visual Basic for Applications (VBA) zum Standard für die Automatisierung. Durch die Verbreitung von VBA hat Visual Basic noch mehr an Bedeutung gewonnen.

## Try Catch Finally

Mit .NET hat Microsoft nun endlich eine umfangreiche sprachübergreifende Plattform geschaffen, die universell für fast alle Bereiche der Softwareentwicklung einsetzbar ist. Aus den Erfahrungen, die bislang bei der Programmierung mit C++, VB und Java gesammelt wurden, entstanden die Bausteine des .NET-Konzeptes.

Alle Programmiersprachen arbeiten mit demselben Typsystem. Definitionen wie CLS (Common Language Specification), CLR (Common Language Runtime), CTS (Common Type System) und die allen Sprachen gemeinsame Kompilierung in einen von JIT (Just In Time)-Compilern auf dem jeweiligen Zielsystem übersetzten Zwischencode (MSIL – Microsoft Intermediate Language) sorgt für flexible Austauschbarkeit von Komponenten. Klassen, die in einer VB.NET-Klassenbibliothek abgelegt sind, können in einem C#-Programm eingebunden, benutzt oder gar durch Vererbung erweitert werden und vice versa. Aber auch jede andere Programmiersprache, die intern auf den Regeln des .NET aufbaut, kann eine so erstellte Bibliothek benutzen (Eiffel.NET, Perl.NET etc.)

Die Sprachen selbst haben an Bedeutung verloren. Eine Programmiersprache dient lediglich zur Umsetzung von Algorithmen in lauffähige Programme. Das gesamte Umfeld, also die Erzeugung von Fenstern, die Zugriffe auf Datenbanken, die Zeichenoperationen und vieles mehr sind nicht mehr Bestandteil der Sprache, sondern werden vom Framework, einer gewaltigen Klassenbibliothek, bereitgestellt.

Unglücklicherweise hat man bei den alten VB-Versionen auch versucht, fehlende Funktionalität von Basic in die Sprache Visual Basic einzubauen. So finden sich auch in VB.NET leider immer noch viele Altlasten in Form von Funktionen, die eigentlich nichts in der Programmiersprache zu suchen haben, sondern in den Aufgabenbereich des Betriebssystems fallen. Der eigentliche Wunsch, den Umsteigern von VB6 nach VB.NET zu helfen, indem man einen Großteil der alten Basic-Funktionen auch unter VB.NET verfügbar macht, schlägt leider allzu oft ins Gegenteil um. Ohne Hilfe ist kaum nachzuvollziehen, ob und wie eine Methode von der Programmiersprache auf das Framework abgebildet wird.

Während der Schritt von C nach C++ hauptsächlich darin bestand, auf die Sprache C einen objektorientierten Ansatz aufzupropfen, wurde mit VB.NET quasi eine neue Sprache entwickelt, die mit VB6 und den Vorgängern (ab sofort VB Classic genannt) nur noch einen kleinen Teil der Syntax gemein hat. Das Verständnis der objektorientierten Programmierung (OOP) ist zwingende Voraussetzung für die Programmierung unter .NET, auch für VB.NET. Selbst wenn man einfache Aufgaben mit der prozeduralen Vorgehensweise, wie sie leider bei der VB Classic-Programmierung vorherrschte, auch auf ähnliche Weise mit VB.NET erledigen kann, stößt man sehr schnell an Grenzen. Die Möglichkeiten des Frameworks lassen sich nur erschließen, wenn man die wichtigsten OOP-Konzepte (Klassen – Objekte – Referenzen, Vererbung, Schnittstellen usw.) beherrscht. Auch wenn viele Rezepte in Form von Funktionen realisiert sind, verstehen sich diese Funktionen als Bestandteil einer übergeordneten Klasse.

## 4 Was sich mit Visual Basic .NET realisieren lässt und was nicht

Da .NET alle Sprachen mit den gleichen Möglichkeiten ausstattet, hat sich das Einsatzgebiet von Visual Basic .NET gegenüber VB Classic erheblich erweitert. Neben Windows-Applikationen und Klassenbibliotheken können Sie nun auch mit Visual Basic .NET Web-Anwendungen, WebServices und Konsolenanwendungen programmieren. Sogar Windows-Dienste sind realisierbar.

Für Handheld-PCs, die mit dem entsprechenden Framework ausgestattet sind, lässt sich ebenfalls Software in Visual Basic .NET entwickeln. Visual Studio bietet für diese Geräte eine spezielle Testumgebung an.

Kein Einsatzgebiet für .NET-Anwendungen sind Gerätetreiber. Diese bleiben nach wie vor eine Domäne der C-Programmierung und können nicht mit VB.NET realisiert werden.

Aber auch bei den direkt nutzbaren Features des Betriebssystems hat sich einiges getan. Multithreading konnte man beispielsweise mit VB6 zwar einsetzen, aber nicht mit der Entwicklungsumgebung austesten. Das hat sich mit VB.NET geändert.

Grundsätzlich lässt sich mit VB.NET alles realisieren, was Sie auch mit C# realisieren können. Aufgrund der syntaktischen Unterschiede der Sprachen lassen sich manche Dinge in der einen oder anderen Sprache eleganter ausdrücken. Unüberwindbare Einschränkungen, wie es sie früher für VB6 in Bezug auf C++ gab, gibt es bei den .NET-Sprachen nicht mehr.

## 5 Inhalt des Buches

Will man ein Buch zu .NET schreiben, stellt sich die Frage, auf welcher Ebene man aufsetzt und welches Ziel verfolgt werden soll. Wie wir schon im Vorwort erwähnt haben, richtet sich das Buch nicht an VB Classic-Programmierer, die mit .NET noch keine Erfahrung gesammelt haben. Für den Umstieg von VB6 nach VB.NET und den Einstieg in die Objektorientierte Programmierung gibt es bereits umfangreiche Literatur. Die wichtigsten Begriffe rund um die Objektorientierte Programmierung mit Visual Basic .NET haben wir im Referenzteil erläutert. Dennoch kann die kurze Zusammenfassung nicht jedes Detail beleuchten und ersetzt nicht die Literatur zum Erlernen der OOP.

Ziel des Buches ist es vielmehr, anhand von vielen praxisorientierten Rezepten Vorgehensweisen aufzuzeigen, wie man effektiv mit den Framework-Klassen arbeitet, wie man mit Techniken wie GDI+, XML, ADO.NET, Windows Forms, Windows Controls, WMI usw. umgeht und nicht zuletzt, wie man notfalls auch auf das Windows-API zugreifen kann, wenn die benötigte Funktionalität (noch) nicht im Framework vorhanden ist.

Diese Rezepte, die Sie direkt im Anschluss an die Einleitung finden, stellen den Hauptteil des Buches dar. In mehrere Kategorien aufgeteilt, haben wir für Sie ca. 300 Rezepte aus den unterschiedlichsten Bereichen entwickelt. Ausgewählt haben wir die Themen und Aufgabenstellungen aus eigenen praktischen Erfahrungen, aus vielen Fragestellungen, die im Rahmen von Seminaren an uns herangetragen worden sind, aus Diskussionen in den Newsgroups und aus den Anregungen der zahlreichen Beiträge im Internet.

Ein Rezept erklärt zunächst eine Aufgabenstellung und zeigt dann eine Lösung, bestehend aus kommentierten Listings, Abbildungen und, vor allem, einer Erläuterung des Lösungsweges. Zu den meisten Rezepten finden Sie auf der Buch-CD die Visual Studio-Projekte, mit denen wir den Code getestet haben. Die abgedruckten Listings können Sie über das ebenfalls auf der CD befindliche Repository kopieren. Sie müssen also nichts abtippen.

## Zielgruppe

Das Buch richtet sich an Anwendungsentwickler, die bereits Erfahrungen mit .NET gesammelt haben. Gute Programmierkenntnisse und Kenntnisse der Objektorientierten Programmierung werden bei der Erläuterung der Rezepte vorausgesetzt. Natürlich sind alle Quelltexte mit Visual Basic .NET geschrieben worden, sonst wäre ja das Thema verfehlt. Aber auch Programmierer anderer Sprachen, die die Syntax von VB.NET einigermaßen beherrschen, können die Rezepte umsetzen.

Der Schwerpunkt der Themen liegt eindeutig auf der Windows-Programmierung, da die Web-Programmierung bereits durch das ASP.NET Codebook abgedeckt wird. Aber auch Web-Programmierer werden viele nützliche Rezepte in diesem Buch finden. Selbst die Rezepte zu GDI+ können für Web-Anwendungen interessant sein, wenn z.B. auf dem Server Grafiken online erstellt werden müssen.

## Voraussetzungen zur Nutzung der Rezepte

Alle Projekte, die Sie auf der Buch-CD finden, wurden mit Visual Studio .NET 2003 und der Framework-Version 1.1 unter Windows 2000 erstellt. Wir gehen davon aus, dass Sie selbst mit Visual Studio .NET arbeiten, denn das ist der Standard für die Programmierung mit Visual Basic .NET.

Alternativ können Sie auch die Freeware SharpDevelop einsetzen. Sie kann den Umfang von Visual Studio .NET zurzeit aber bei weitem nicht erreichen. Wichtige Features wie Debugging und umfassende visuelle Designer fehlen. Für die ganz hart Gesotenen bleiben dann noch die kostenlose Nutzung von Notepad-Editor und Kommandozeilenaufrufe des Compilers ☺.

Die Softwareentwicklung mit Visual Basic .NET sollten Sie nur auf den Betriebssystemen Windows 2000 und Windows XP bzw. zukünftigen Nachfolgern vornehmen. Visual Studio .NET 2003 arbeitet ohnehin nur noch mit diesen Betriebssystemen. SharpDevelop funktioniert angeblich auch unter Windows NT4 und Windows 98/ME, wir raten aber davon ab.

## Typische Fragen zum Visual Basic .NET Codebook und zur Programmierung mit Visual Basic .NET

### Wie bekomme ich die abgedruckten Quelltexte aus den Rezepten in meine Anwendung?

Auf der Buch-CD befindet sich ein Repository-Programm. Nach Kategorien organisiert suchen Sie das betreffende Rezept und dort das gewünschte Listing aus und kopieren den Quelltext über die Zwischenablage in Ihre Anwendung.

### Kann ich die Rezepte ohne Veränderung in meine Anwendung übernehmen?

Wenn es sich realisieren ließ, haben wir die Rezepte so aufgebaut, dass sie ohne Änderung übernommen werden können. Allerdings war das in vielen Fällen nicht möglich, da bei der Objektorientierten Programmierung z.B. Methoden nicht einfach aus dem Klassenverbund herausgerissen werden können. Bei Windows-Forms oder Windows-Controls ergibt sich das Problem, dass wir aus Platzgründen nicht den Code zur Generierung der Steuerelemente etc. abdrucken können. Auch syntaktische Gegebenheiten, wie z.B. Imports-Anweisungen am Anfang einer Datei, verhindern oft, dass Methoden und Klassen mit einem Mausklick kopiert werden können. Meist müssen an mehreren Stellen Änderungen vorgenommen werden.

Eine Reihe von Rezepten haben wir anhand von Beispielen erläutert. Hier müssen Sie natürlich den Beispielcode für Ihre Zwecke anpassen.



**Gibt es zu den Rezepten Beispielprojekte?**

Ja. Für fast alle Rezepte finden Sie auf der CD Beispielprojekte, in denen der beschriebene Code eingesetzt und demonstriert wird. Dabei gibt es einige Rezepte, die an einem zusammenhängenden Beispiel erläutert werden und dann auch in einem Projekt zusammengefasst werden. Nur für ganz wenige Rezepte, die ohne nennenswerten Code auskommen (z.B. zur Diskussion von Basis-Klassen für Steuerelemente), gibt es keine eigenen Projekte.

**Wie finde ich das zu einem Rezept zugehörige Projekt?**

Auf der CD finden Sie eine Projekt-Referenz, die es Ihnen erlaubt, über eine Rezeptauswahl zum Projektverzeichnis zu navigieren.

**Kann ich die Beispielprojekte sofort verwenden?**

Das sollte gehen. Bedenken Sie aber bitte, dass das Projekt zuvor auf die Festplatte kopiert werden muss, da sonst der Compiler keine Dateien und Verzeichnisse auf der CD anlegen kann.

**Ich programmiere mit C#. Kann ich das Buch auch nutzen?**

Ja. Denn das Kernthema des Buches ist die Programmierung mit dem .NET Framework und nicht, wie man mit Visual Basic eine Schleife programmiert. Wenn Sie C# beherrschen und sich ein bisschen mit der Syntax von VB.NET auseinander setzen, werden Sie die Rezepte nutzen können. Hierzu können Sie wahlweise den VB-Code selbst in C# umsetzen oder einfach die Projekte nutzen und als Klassenbibliothek in einem C#-Projekt einbinden.

**Ich programmiere mit ASP.NET. Was bringt mir das Buch?**

Das Schwerpunktthema des Buches ist die Programmierung von Windows-Anwendungen. Aber auch die umfasst viele Bereiche, die nichts mit Windows-Oberflächen zu tun haben. Beispielsweise sind Systemzugriffe, Datumsberechnungen, Zugriffe auf XML-Dateien und Datenbanken oder Dateioperationen für Web-Anwender genauso interessant wie für Windows-Programmierer. Selbst die Rezepte zum Eigenschaftsfenster können sehr nützlich sein, wenn Sie Web-Controls entwerfen, die Sie an andere Anwender weitergeben wollen.

**Wird auf ASP.NET eingegangen?**

Nein. Hierfür gibt es das ASP.NET Codebook, dem wir auch keine Konkurrenz machen wollen ☺. Web-Anwendungen und Web-Dienste werden nur insoweit besprochen, wie sie in Windows-Anwendungen genutzt werden können. Auf spezielle ASP.NET-Themen wird nicht eingegangen.

**Benötige ich Visual Studio für die Programmierung mit VB?**

Jein. Eigentlich ist alles, was Sie benötigen, das .NET Framework SDK und ein Texteditor (und eigentlich können Sie Ihre Programme auch auf Lochkarten stanzen ;-)). Aber wenn Sie bereits mit den Visual Basic 5 oder 6 gearbeitet haben, dann wollen Sie die komfortable Entwicklungsumgebung bestimmt nicht missen. Für die professionelle Arbeit mit Visual Basic .NET führt derzeit kein Weg an Microsoft Visual Studio vorbei.

**Benötige ich Visual Studio für den Einsatz der Rezepte?**

Auch hier ein Jein. Im Prinzip nicht, aber praktischer ist es schon. Alle Projekte wurden mit Visual Studio erstellt und können mit diesem natürlich auch wieder geöffnet und weiterbearbeitet werden.

**Ich habe nur VB6. Kann ich die Beispiele auch benutzen?**

Nein. Die Programmierung unter VB.NET hat nicht mehr viel mit der Programmierung unter VB6 zu tun. Die Beispiele sind nur unter VB.NET lauffähig.

### **Werden Module oder andere aus Kompatibilitätsgründen übernommene alte VB6-Techniken verwendet?**

Nein. Diese Techniken sind für Umsteiger beim Erlernen des .NET-Konzeptes eher hinderlich als förderlich. Alles, was sich mit dem Framework und den Techniken der Objektorientierten Programmierung realisieren lässt, haben wir auch so realisiert. Natürlich verwenden wir dort, wo es Sinn macht, VB-Begriffe wie z.B. Integer statt System.Int32. Auch für gängige Typ-Umwandlungen setzen wir in vielen Fällen Methoden wie CInt oder CSng ein, statt die doch sehr umständliche Typecast-Syntax von CType oder DirectCast zu verwenden.

Grundsätzlich sind wir der Meinung, dass man sich auch (oder vielleicht gerade) als ehemaliger VB6-Programmierer von den Altlasten von VB Classic befreien muss und nicht gegen die Konzepte von .NET arbeiten sollte. Die prozedurale Programmierung, in der häufig mit globalen Variablen und Methoden gearbeitet wurde, hat hier keinen Platz mehr.

### **Werden alle Themenbereiche von .NET angesprochen?**

Wenn Sie bei dieser Frage insgeheim denken »ich hoffe doch ja«, dann sollten Sie wissen, dass Sie ein solches Buch nicht mehr forttragen könnten. Nein, .NET umfasst derartig viele Bereiche, dass es völlig unmöglich ist, auch nur annähernd alles in einem Buch zu beschreiben. Wir haben die Themen herausgesucht, die unserer Meinung nach für die meisten Visual Basic-Programmierer von Belang sind und/oder sich in der Praxis, Newsgroups etc. als relevant herausgestellt haben.

### **Werden Techniken wie XML detailliert erklärt?**

Nein. Auch hier stoßen wir mit dem Buch an Grenzen. Wir erklären, wie Sie mit diesen Techniken in .NET umgehen können. Aber die Grundlagen von z.B. XML können wir nicht umfassend dokumentieren. Hierzu gibt es ausreichende Literatur, in Büchern und online.

### **Wird auch Multithreading eingesetzt?**

Wie bereits oben erwähnt, ist der Einsatz von Multithreading in Visual Basic .NET wesentlich vereinfacht worden. Allerdings warnen wir ausdrücklich davor, ohne massive Kenntnis der Grundlagen Multithreading einzusetzen. Gerade im Umgang mit Windows-Anwendungen gibt es viele Fallen, in die (mit Multithreading) unerfahrene Programmierer tappen können. Multithreading ist erheblich komplizierter, als es auf den ersten Blick den Anschein macht.

Wir haben Multithreading nur in einzelnen Ausnahmefällen eingesetzt, wenn auch wirklich ein Grund dafür gegeben war.

### **Auf welchen Plattformen laufen die entwickelten .NET-Programme?**

Theoretisch auf allen Plattformen, für die es das .NET Framework gibt. Das heißt auf Windows 98SE, Windows ME, Windows NT 4.0, Windows 2000, Windows XP und Windows 2003 Server. Allerdings gibt es für Windows 98/ME sehr viele Einschränkungen. Ähnlich sieht es bei Windows NT aus. Die vollständige Framework-Implementierung gibt es erst ab Windows 2000.

### **Auf welchen Plattformen funktionieren die Beispiele aus dem Buch?**

Entwickelt und getestet haben wir unter Windows 2000. Die Programme sollten sich unter Windows XP und Windows 2003 Server genauso verhalten. Auf den älteren Plattformen werden höchstwahrscheinlich einige Beispiele (z.B. mit Transparenzeffekten) nicht funktionieren.

### **Funktionieren die Rezepte auch auf dem Framework 1.0 und mit Visual Studio .NET 2002?**

Größtenteils wahrscheinlich schon. Wir haben es aber nicht ausprobiert. Im Sourcecode sind ggf. kleine Änderungen bei den For-Schleifen notwendig, da wir in der Regel von der Möglichkeit Gebrauch gemacht haben, die Schleifenvariablen im For-Statement zu definieren. Auch kann es in

Ausnahmefällen vorkommen, dass Erweiterungen des Frameworks benutzt werden, die in der Version 1.0 nicht enthalten sind. Ein Beispiel dafür ist die Enumeration `Environment.SpecialFolder`, die in der Version 1.1 deutlich umfangreicher ausgefallen ist.

## Die Rezepte

In 15 Kategorien finden Sie über 300 Rezepte. Was Sie in den Kategorien erwartet, haben wir hier kurz zusammengefasst.

### Basics

Hier finden Sie Grundlegendes z.B. zu Formatierungen, Bit-Operationen, Vergleichen und Sortieren von Objekten und Enumerationen. Die alte VB `Format`-Anweisung hat ausgedient. Lesen Sie nach, welche neuen weitaus komfortableren Möglichkeiten das Framework bietet, um Zahlen und andere Werte zu formatieren und zu konvertieren. Der Umgang mit Bits und Bytes ist in .NET wesentlich einfacher geworden. Nicht nur, dass die Sprache VB.NET selbst endlich Schiebeoperationen kennt, sondern auch das Framework hält einige Klassen für Konvertierungen bereit, die früher nur sehr umständlich realisierbar waren.

Enumerationen sind in .NET nicht mehr einfache benannte Konstanten, sondern bieten auch zur Laufzeit Unterstützungen, z.B. zum Abfragen aller definierten Werte und Namen. Wir zeigen Ihnen, wie Sie mit Enumerationen umgehen können, wie Sie Enums ein- und auslesen können, ohne mit den nackten Zahlenwerten arbeiten zu müssen.

Zum Sortieren von Listen und Arrays, die Objektreferenzen enthalten, gibt es allgemeingültige, vom Framework genutzte Entwurfsmuster (Design Patterns). Wir erklären die Mechanismen, die Sie benötigen, um Objekte vergleichen und suchen zu können.

(Fast) ausgedient haben auch die alten Funktionen `Asc` und `Chr`, denn sie arbeiten nur mit ASCII-Zeichen. Zur Laufzeit arbeiten .NET Strings aber grundsätzlich mit Unicode. Wie Sie die Umwandlung zwischen den verschiedenen Zeichencodierungen (ASCII, UTF8, UTF16 usw.) vornehmen können, erfahren Sie ebenfalls in dieser Kategorie.

### Datum und Zeit

Für den Umgang mit Datums- und Zeitangaben stellt das Framework umfangreiche Strukturen, Klassen und Methoden bereit. Leider gibt es auch ein paar Fehler und Fallen, z.B. bei der Berechnung der Kalenderwochen, wie sie in Europa üblich sind. Dagegen helfen unsere Rezepte ☺.

Ein immer wieder gefragtes Thema ist die Berechnung der deutschen Feiertage. Mit mehreren Rezepten zeigen wir, wie grundlegende Daten (Kirchenjahr, Osterdatum) berechnet werden, und stellen Ihnen eine Klasse für den Umgang mit Feiertagen zur Verfügung, die beliebig erweitert werden kann.

Für Historiker interessant ist die Umrechnung zwischen dem Gregorianischen Datum und der Julianischen Tageszählung.

### Anwendungen

Nach Ini-Dateien und Registry-Einträgen werden heute Konfigurationsdaten in XML-Dateien gespeichert. In der Kategorie Anwendungen stellen wir Rezepte vor, um aus der Konfigurationsdatei einer Anwendung Daten zu lesen. Wir zeigen, wie Sie selbst Konfigurationsdateien anlegen und in Ihrem Programm diese lesen und schreiben können.

Auch für die oft gestellte Frage, wie man verhindern kann, dass eine Anwendung mehrfach gestartet wird, finden Sie hier ein Rezept. Ebenso für die Klärung der Frage, wie man in einer Anwendung eine zentrale Fehlerbehandlung durchführen kann.

## GDI+ Zeichnen

GDI+ ist das Grafik-API der neuen Betriebssysteme und wird ab Windows XP mit diesem zusammen installiert. GDI steht für Graphics Device Interface. Es ist nicht Bestandteil von .NET, jedoch basieren alle grafischen Ausgaben auf GDI+. GDI+ ist somit die Plattform für alle Zeichnungen, die mit einem .NET-Programm erstellt werden. Gegenüber dem alten gewachsenen GDI bietet es eine Reihe von Vorteilen. Hier nur einige wichtige:

- ▶ Systematisch aufgebautes Klassen- und Funktionsmodell
- ▶ Einfach anwendbare Zeichengeräte wie Pen, Brush und Font
- ▶ Koordinatensysteme und -transformationen
- ▶ Transparenz über Alpha-Kanal

Heutige Grafikkarten verfügen in der Regel noch nicht über eine Hardware-Beschleunigung für GDI+. In vielen Fällen, insbesondere bei der Anwendung von Transparenz, kann es daher zu Performance-Einbußen gegenüber GDI kommen. Diese Nachteile sollten jedoch in einigen Jahren, wenn die nächsten Generationen von Grafikkarten zur Verfügung stehen, der Vergangenheit angehören.

Im Wesentlichen ersetzt GDI+ alle Funktionen von GDI. Es soll jedoch nicht verschwiegen werden, dass ein paar Möglichkeiten des alten GDI in GDI+ nicht mehr oder zumindest nur noch eingeschränkt zur Verfügung stehen. Dazu gehört z.B. die Möglichkeit, durch wiederholtes XOR-Drawing zuvor gezeichnete Figuren wieder zu löschen, ohne den Hintergrund neu zeichnen zu müssen.

Dieses Buch kann und soll nicht die Grundlagen von GDI+ erklären. Stattdessen wird an Hand einiger Beispiele der praktische Einsatz gezeigt. Da Zeichenoperationen aber zu den wichtigsten Bestandteilen einer grafischen Oberfläche zählen, gehen wir im Buch etwas ausführlicher auf sie ein. In der Kategorie GDI+ Zeichnen zeigen wir Ihnen zunächst allerlei Tricks und Kniffe im Umgang mit Schriften, Schatten und 3D-Effekten. Die Zeiten der eintönig grau in grau erscheinenden Dialogboxen sollte allmählich der Vergangenheit angehören.

## GDI+ Bildbearbeitung

Beginnend mit den Grundlagen zum Zeichnen von Bildern finden Sie Rezepte zu komplexen Überblend-Funktionen mit Clipping und Transparenz, zum Vergrößern von Bildausschnitten, zur Maximierung von Bildanzeigen und zum Drehen und Spiegeln. Wir zeigen Ihnen, wie Sie in Ihrem Programm eine Ordnerauswahl mit miniaturisierten Vorschaubildern anzeigen und Thumbnail-Bilder für Webseiten generieren können.

Des Weiteren finden Sie Rezepte zum Umgang mit verschiedenen Dateiformaten und Encodern. Oft wird gefragt, wie man Bilder im JPEG-Format speichern und dabei Einfluss auf die Qualität nehmen kann. Auch das zeigen wir hier.

Mit Hilfe von Matrix-Operationen können die Farbinformationen auch von größeren Bildern sehr schnell verändert werden. Wir haben Rezepte für Sie vorbereitet, um Farbbilder zu invertieren oder in Graustufen umzuwandeln. In einem Rezept erklären wir ein Testprogramm, mit dessen Hilfe Sie die Farbmatrix online verändern und gleichzeitig die Auswirkung sehen können.

Für technische und wissenschaftliche Auswertungen werden oft Scharfzeichnungsfilter benötigt. In mehreren Rezepten erklären wir, wie die gängigen Filteralgorithmen in Visual Basic .NET umgesetzt werden können.

## Windows Forms

Nahezu unüberschaubar sind die Möglichkeiten, die .NET zur Gestaltung von Fenstern bietet. Neu hinzugekommen sind Fenster mit nicht rechteckigen Umrissen sowie ganz oder teilweise durchsichtige Fenster. Wir zeigen an einigen Beispielen, wie Sie die neuen Effekte nutzen können. Sie finden Rezepte zu oft gestellten Fragen, wie z.B. Fenster ohne Titelleiste erzeugt und verschoben werden können und wie man per Programm ScreenShots erstellen kann.

Viele VB Classic-Programmierer haben große Probleme bei der Programmierung von Dialogfenstern unter .NET. Die Fragen in den Newsgroups zeigen nur allzu oft, dass Dialoge unter VB Classic nur selten objektorientiert programmiert worden sind, obwohl bereits VB5 eine Menge Möglichkeiten zur Kapselung von Dialogen zu bieten hatte. Wir halten es daher für außerordentlich wichtig, diese Thematik in einigen Rezepten aufzugreifen und zu erklären, wie man unter .NET Dialoge kapseln kann. Dadurch erhalten die Dialogklassen eine saubere Struktur, sind besser wartbar und können wiederverwendet werden. Durch Vererbung können Sie zusätzlich erreichen, dass die Dialogfenster eines Programms ein einheitliches Aussehen bekommen.

Ebenfalls im Zusammenhang mit Dialogfenstern wird diskutiert, wie Benutzereingaben überprüft werden können. Das Framework bietet auch hierzu einige neue Mechanismen.

Auch oft gestellte Fragen wie z.B., wie man einen Startbildschirm (Splashscreen) anzeigt oder wie man über Drag&Drop oder über die Zwischenablage Daten importieren oder exportieren kann, werden in dieser Kategorie mit mehreren Rezepten beantwortet.

Wenngleich wir die Projekte alle unter Windows 2000 erstellt haben, zeigen wir natürlich auch, wie Sie die neuen Darstellungsformate für Schaltflächen etc. unter Windows XP nutzen können.

Letztlich zeigen wir Lösungen für banal klingende Aufgabenstellungen, die sich bei näherer Betrachtung als äußerst knifflig erweisen, z.B. wie in einer TextBox die Eingabetaste abgefangen werden kann und wie man Dateipfade so kürzen kann, dass sie in noch lesbarer Form in Menüs verwendet werden können.

## Windows Controls

Dieses ist aus gutem Grund die umfangreichste Kategorie. Der Umgang mit Steuerelementen hat sich grundlegend geändert. Es gibt nicht mehr die Aufteilung in Ressource und Code-Teil, wie es in VB Classic der Fall war. Alle Steuerelemente werden per Code erzeugt. Auch die Eigenschaften werden im Code initialisiert. Beginnend mit einfachen Themen, z.B., wie die Control-Arrays aus VB Classic ersetzt werden können, über das benutzerdefinierte Zeichnen von ListBoxen und mehrspaltigen ComboBoxen gehen wir intensiv auf die Programmierung eigener Steuerelemente ein. Wir erläutern, welche Basisklassen sich für welche Steuerelemente eignen und welche Möglichkeiten es gibt, in den Nachrichtenfluss eines Controls einzugreifen.

Komplexere Steuerelemente für Verzeichnis- und Dateiauswahl werden ausführlich vorgestellt. Sie können direkt für den Bau eigener Dialoge verwendet werden, falls die von Windows bereitgestellten nicht ausreichen.

Am Beispiel eines grafischen Steuerelementes zur Anzeige von Zeitsegmenten demonstrieren wir, wie man Steuerelemente mit nicht rechteckigem Umriss definiert, wie Koordinaten transformiert werden, wie Maus-Ereignisse auch zur Entwurfszeit bearbeitet werden können und was zu tun ist, um auch im Eigenschaftsfenster die Zeitsegmente einstellen zu können.

Mehrzeilige TextBoxen werden vom Framework etwas stiefmütterlich behandelt. Einige Informationen und Einstellungen sind nur über API-Funktionen erreichbar. Auch hierzu halten wir einige Rezepte parat.

## Eigenschaftsfenster (PropertyGrid)

Eines der leistungsfähigsten Steuerelemente, die Ihnen auch für eigene Anwendungen zur Verfügung stehen, ist das `PropertyGrid`-Control (Eigenschaftsfenster). Es handelt sich dabei um das Steuerelement, das der Designer benutzt, um dem Entwickler die Möglichkeit zu geben, für ein ausgewähltes Objekt (Steuerelement, Fenster, Menü etc.) dessen Eigenschaften anzuzeigen und zu ändern (siehe Abbildung 1). Sicher haben auch Sie es schon oft in Verbindung mit dem Designer verwendet.

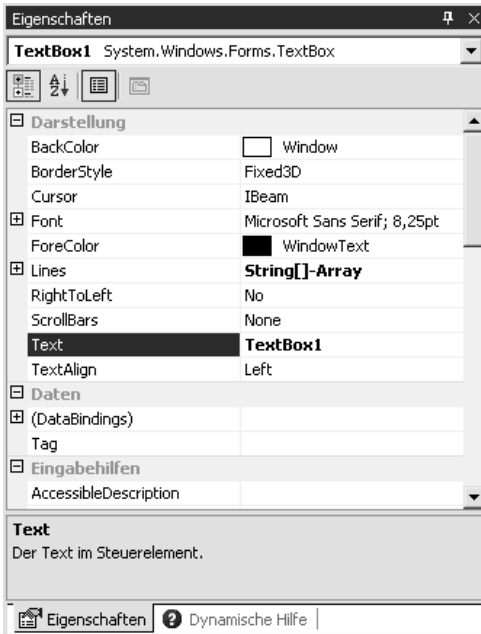


Abbildung 1: Das Eigenschaftsfenster (PropertyGrid) ist eines der leistungsfähigsten Steuerelemente und lässt sich auch in eigene Anwendungen einbinden

Egal, ob Sie im Designer die Eigenschaften Ihrer eigenen Steuerelemente korrekt anzeigen wollen oder ob Sie zur Laufzeit das `PropertyGrid`-Control in Ihren eigenen Anwendungen dazu benutzen möchten, die Eigenschaften beliebiger Objekte anzuzeigen, es stehen Ihnen vielfältige Möglichkeiten zur Verfügung, das Verhalten des Eigenschaftsfensters gezielt zu steuern. Insbesondere für die professionelle Entwicklung von Steuerelementen ist es unerlässlich, sich näher mit dem `PropertyGrid` zu beschäftigen, um dem Anwender alle erdenklichen Hilfestellungen und Vereinfachungen für den Umgang mit den Steuerelementen zur Verfügung stellen zu können.

Eigentlich wollten wir einem einzelnen Steuerelement kein eigenes Kapitel widmen. Das `PropertyGrid`-Control ist aber leistungsstark wie kaum ein anderes, wird oft eingesetzt und ist in der MSDN-Dokumentation nur lückenhaft beschrieben. Auch im Internet findet man nur sehr verstreut die benötigten Informationen. Deswegen haben wir an dieser Stelle eine zusammenhängende, viele Details umfassende Sammlung von Rezepten zum Eigenschaftsfenster für Sie erarbeitet.

Anfangen mit Rezepten zur Steuerung des `PropertyGrid` über Attribute für Kategorien, Beschreibungen, Standardwerten und Standardeigenschaften führen wir Sie in Techniken zum Anzeigen eigener Editoren ein, die den Anwender bei der Eingabe der Daten unterstützen können.

Wir erklären den Umgang mit Auflistungen und Enumerationen und wie Sie das Eigenschaftsfenster um zusätzliche Hyperlink-Schaltflächen und Tab-Seiten erweitern können.

Das starre Anzeigen der vorgegebenen Eigenschaftsnamen kann in mehrsprachigen Programmen störend sein. Daher finden Sie hier auch ein Rezept, das beschreibt, wie Sie die angezeigten Texte lokalisieren können.

## **Dateisystem**

Dateisystemverwaltung zählt zu den grundlegenden Eigenschaften, die ein Betriebssystem unterstützen muss. Ohne die Möglichkeit, Dateien und Verzeichnisse erstellen, verändern und löschen zu können, wären alle Informationen mit dem Abschalten des Rechners verloren. Was bliebe, wären Lochstreifen und Lochkarten. Im Laufe der Jahre haben sich die Anforderungen an ein Dateisystem stetig erhöht.

In den Rezepten zum Dateisystem finden Sie Rezepte für eben diese Möglichkeiten. Da das Betriebssystem aber neben »normalen« Dateien ebenfalls im Dateisystem abgespeichert wird, zeigen einige Rezepte auf, in welchen Verzeichnissen ein Anwender diese Informationen bei der Installation des Betriebssystems hinterlegt hat.

Da ein modernes Dateisystem ein recht dynamisches Gebilde ist, kann man manchmal die Überwachung von bestimmten Dateien oder Verzeichnissen in einem Programm nicht vermeiden. Auch findet heute der Datenaustausch zwischen unterschiedlichen Plattformen teilweise noch immer über Dateien statt. Ein Rezept stellt die Möglichkeiten vor, wie man innerhalb eines Programms Änderungen am Dateisystem überwachen kann.

## **Netzwerk**

Ein Netzwerk ist in der modernen EDV nicht mehr wegzudenken. War ein Netzwerk bis vor kurzer Zeit noch ein zusätzliches Programm oder ein zusätzlicher Treiber, der dem Betriebssystem bekannt gemacht werden musste, gehören Netzwerkfunktionen heute bereits zum Betriebssystem. Dementsprechend unterscheiden viele Funktionen nicht mehr zwischen lokal und global.

Wir haben in diesem Abschnitt des Buches Rezepte aufgeführt, die trotz allem mehr dem Netzwerk zuzuordnen sind, z.B. welche Netzwerkadapter sind in einem Rechner eingebaut. Da die IP-Adressen der Version 6 immer bekannter werden, sind auch zwei Rezepte zur Umrechnung alt auf neu und umgekehrt hier zu finden.

Um das Thema Web-Services nicht allein den ASP-Programmierern zu überlassen ☺ haben wir an dieser Stelle auch ein Rezept für die Erstellung eines Google-Web-Clients für VB.NET aufgeführt.

## **System**

Dieser Abschnitt basiert zu einem nicht kleinen Teil auf einer Technik, die relativ unbekannt, aber trotz allem sehr effektiv genutzt werden kann: WMI. Eine kleine Einführung in die Thematik haben wir in den Anhang aufgenommen.

Das System kann grob in zwei Bereiche unterteilt werden, die Hardware und die Software. Für den Bereich Hardware finden Sie Rezepte, die es Ihnen ermöglichen, den Plattenplatz zu ermitteln oder das BIOS auszulesen. Aber auch der Prozessor und seine Eigenschaften oder die Auflösung des Monitors finden Sie in den Rezepten wieder. Auf der Softwareseite finden Sie Rezepte, wie Sie feststellen können, welche Software auf dem System installiert ist oder wie Sie die Registry mit .NET bearbeiten können.

Natürlich dürfen Dienste und deren Steuerung in einem solchen Kontext nicht fehlen. Zwei Rezepte haben hier Eingang gefunden, die mehr mit der Kommunikation zu tun haben, aber viel-

fach als Systemfunktionalität in einem Programm Einzug finden: das Versenden von Fax und eMail aus einer Anwendung heraus.

## Datenbanken

In diesem Teil des Buches finden Sie hauptsächlich Rezepte, die mit der Verwaltung einer Datenbank zu tun haben. Da es genügend Bücher zum Thema ADO.NET gibt, aber eher selten etwas über die Verwaltung einer Datenbank über ein eigenes Programm ausgeführt wird, finden Sie hier Rezepte, mit denen Sie Datenbanken aus einer Anwendung heraus oder als eigenständiges Programm steuern können. Dies ist umso wichtiger, da mit der MSDE kein entsprechendes Programm von Microsoft ausgeliefert wird.

In diesem Abschnitt des Buches finden Sie Rezepte zur Erstellung von Datensicherungen (mit oder ohne Device-Kontext) und zur Automatisierung dieser Tätigkeiten. Sie können dies nicht nur mit der lokalen Datenbank, sondern mit jeder SQL-Datenbank, für die Sie entsprechende Rechte im Netzwerk haben. Natürlich wird auch der umgekehrte Weg, also die Rücksicherung in den Rezepten gezeigt.

Einige Rezepte beschäftigen sich aber auch mit anderen Themen als der Datenbankverwaltung. So finden Sie ein Rezept zur Feststellung der MDAC-Version auf einem Rechner, wie man Bilder in einer Datenbank abspeichern kann oder wie Excel als Datenbank angesprochen wird.

## XML

XML ist eine der tragenden Säulen in .NET. Deswegen wird XML auch in vielen Bereichen des Frameworks eingesetzt und in vielfältiger Weise unterstützt. Wir können leider nicht sehr tief auf die Grundlagen von XML eingehen (dazu gibt es Bücher, die durchaus dicker sind als dieses), aber wir zeigen Ihnen die wichtigsten Vorgehensweisen im Umgang mit XML-Daten und -Dateien auf.

Sie finden in dieser Kategorie einige Rezepte zu grundlegenden Klassen wie `XmlTextWriter` und `XmlTextReader` oder zum Lesen und Schreiben von Bildern und Binärdaten ebenso wie solche zum Umgang mit dem Document Object Model (DOM).

XPath-Ausdrücke sind leistungsstarke Konstrukte für XML-Abfragen. Auch sie werden ausführlich behandelt, besonders in Zusammenhang mit Namensräumen, die bei XPath einigen zusätzlichen Aufwand erfordern. Wir geben Ihnen Rezepte zu den Klassen `XmlDocument` und `XPathDocument` und erläutern die Unterschiede.

Viele Nachrichtenagenturen, Zeitschriftenverlage und auch das MSDN stellen im Internet XML-Dateien (Stichwörter RSS und RDF) mit den aktuellen Nachrichten zur Verfügung. Zwei Rezepte zeigen auf, wie Sie diese Nachrichten verarbeiten und anzeigen können.

Auch Techniken, wie die Validierung von XML-Dateien über ein XML-Schema oder die komfortablen Aufrufmöglichkeiten von Stylesheet-Transformationen (XSLT) haben wir in Rezeptform für Sie aufgearbeitet. Sie finden hier auch ein Rezept, wie Sie Code für Klassen erzeugen können, ohne selbst programmieren zu müssen.

Ein interessantes Thema ist auch der Parallelbetrieb von `DataSet` und `XmlDataDocument`, bei dem Sie wechselweise mit XPath- oder SQL-Abfragen arbeiten können.

Abschließend werden in dieser Kategorie die Serialisierung beliebiger Objekte von und nach XML erläutert. Sie finden Rezepte zu den Klassen `XmlSerializer` und `SoapFormatter`.

## Wissenschaftliche Berechnungen und Kurvendiagramme

Physikalisch technische Maße werden in der Programmierung meist als simple Gleitkommazahlen abgehandelt. Eine Typsicherheit, um hier nicht Längenmaße und Geschwindigkeiten durcheinander zu werfen, wird nur selten vorgesehen.



Am Beispiel einiger Maße (Längenmaß, Flächenmaß usw.) zeigen wir, wie Sie die Objektorientierte Programmierung gezielt einsetzen können, um zum einen die Typsicherheit zu gewährleisten und zum anderen, wie Angaben in verschiedenen Einheiten automatisch in eine Standard-Einheit umgerechnet werden können.

Zur wissenschaftlichen Aufarbeitung von Mess- und Simulationswerten gehört auch die Darstellung in Form von Kurvendiagrammen. Wir beschreiben, wie Sie die Skalierungen der Achsen berechnen können und wie die Kurven gezeichnet werden; insbesondere, wie kontinuierlich durchlaufende T/Y-Diagramme programmiert werden können.

Falls Ihnen die Genauigkeit nicht reicht, mit der das Framework die Zahl Pi berechnet, finden Sie hier auch ein Rezept, mit dem Sie Pi auf 2400 Stellen genau berechnen können 😊.

## Verschiedenes

In der letzten Kategorie haben wir all die Rezepte gesammelt, die für die Kategorie Basics nicht einfach genug sind und die nicht zu den Themen der anderen Kategorien passen.

Eine oft gestellte Frage ist, wie man unter .NET Sound-Dateien abspielen kann. Da das Framework dafür bislang keine Unterstützung bietet, zeigen wir, wie Sie wahlweise mit API-Funktionen oder über DirectX die Aufgabe lösen können.

Wenig bekannte Debug-Möglichkeiten, z.B. die Steuerung von Ausgaben über die Konfigurationsdatei der Anwendung, die Ausgabe von Debug- und Trace-Ausgaben in TextBoxen, Dateien oder Eventlogs oder die Erstellung und Nutzung eigener Leistungsindikatoren (Performance-Counter) waren uns auch einige Rezepte wert. Auch wenn Sie wissen möchten, wie Sie auf Eventlog-Ausgaben anderer Anwendungen reagieren können, finden Sie hier Lösungen.

Ein weiteres Rezept gibt Ratschläge, wie Sie Zeiten mit hoher Auflösung messen können und was dabei zu beachten ist. Auch, wie Sie vorgehen können, wenn Sie Fehlermeldungen der API-Funktionen im Klartext anzeigen wollen.

## Typografische Konventionen

Die folgenden typografischen Konventionen werden in diesem Buch verwendet:

- ▶ Schlüsselwörter von Visual Basic sowie Bezeichner für Variablen, Methoden, Klassen usw. werden innerhalb des Fließtextes in der Schriftart *Courier* dargestellt.

Beispiel: die Klasse `System.Object` und der Wert `True`

- ▶ Listings werden in folgender Schriftart gedruckt. VB-Schlüsselwörter werden fett gedruckt:

---

```
Dim i As Integer = 123
```

---

- ▶ Kommentare in Listings werden kursiv geschrieben:

---

```
' Das ist ein Kommentar
```

---

- ▶ Dateinamen und Verzeichnisnamen werden kursiv formatiert:

Beispiel: *Text.txt*

- ▶ Internetadressen sehen so aus: *www.addison-wesley.de*

- ▶ Texte der Bedienoberflächen, Menüpunkte, Schaltflächenbeschriftungen etc. werden in Kapitälchen formatiert:

Beispiel: Menüpunkt DATEI/ÖFFNEN

## Inhalt der Buch-CD

Auf der Buch-CD befindet sich das bereits erwähnte Repository, mit dessen Hilfe Sie die abgedruckten Quelltexte auffinden und über die Zwischenablage in Ihre Projekte übernehmen können. Zusätzlich finden Sie auf der CD alle Beispielprojekte und eine Querverweisliste, der Sie die Zuordnung von Rezepten zu Projekten entnehmen können.

## Errata

Keine Qualitätskontrolle kann hundertprozentig verhindern, dass Fehler übersehen werden. So verhält es sich auch bei einem Codebook mit mehr als 800 Seiten. Natürlich haben wir alle Beispiele sorgfältig getestet, aber wir können nicht ausschließen, dass uns Fehler unterlaufen sind oder dass in manchen Situationen ein Beispiel nicht so funktioniert, wie es beschrieben wird.

Falls Sie also Fehler oder Unstimmigkeiten finden sollten oder Anregungen und Verbesserungsvorschläge haben, lassen Sie es uns wissen. Unsere Internet-Adresse, über die Sie den Kontakt herstellen können, finden Sie im Vorwort.

## Anhang / Referenzteil

Im Anhang haben wir für Sie wichtige Hintergrundinformationen zusammengefasst. In einer kurzen Zusammenfassung erläutern wir die wichtigsten Themen und Begriffe rund um die Objektorientierte Programmierung mit Visual Studio .NET. Wir zeigen, was sich gegenüber VB6 wesentlich verändert hat. Auch Arrays und Auflistungen werden hier näher erörtert.

Ein weiteres Schwerpunktthema im Referenzteil ist der Umgang mit Visual Studio .NET 2003. Nicht, wie Sie ein Projekt anlegen (das sollten Sie bereits beherrschen), sondern kleine, aber hilfreiche Details, die oft ungenutzt bleiben, werden hier vorgestellt.

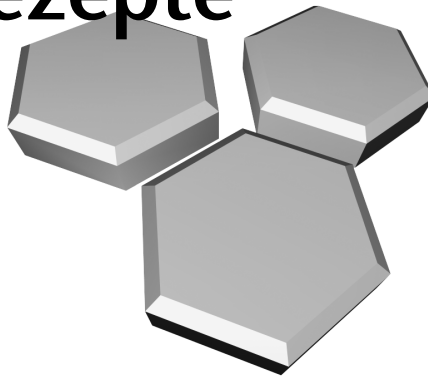
Natürlich nennen wir auch einige Quellen im Internet, in denen Sie zusätzliche Informationen finden können. Auch erklären wir, wie Sie in den gängigen Newsgroups Hilfe finden können, wie Sie mit Google in den Newsgroups recherchieren oder selber an den Diskussionen teilnehmen können.

Einige Grundlagenthemen, die aus Platzgründen nicht in die Rezepte aufgenommen werden konnten, werden im Referenzteil ebenfalls kurz aufgegriffen und erläutert.

Da das .NET Framework nicht alles abdecken kann, werden in einigen Rezepten Windows-API-Funktionen verwendet. Die verwendeten Funktionen, Strukturen und Konstanten haben wir in einer Klasse gekapselt, die ebenfalls im Referenzteil abgedruckt ist.

# TEIL II

## Rezepte





Zu Beginn wollen wir Ihnen ein paar grundlegende Vorgehensweisen vorstellen, die einige Leser vielleicht schon kennen. Aber die Erfahrung zeigt, dass viele Umsteiger von VB6 mit den neuen Features von VB.NET und den grundlegenden Framework-Klassen und -Methoden nicht vertraut sind. Daher beginnen wir mit einfachen Dingen wie Zahlenformaten und gehen auch auf Bit-Operationen und Vergleiche von Objekten ein.

## 1 Zahlen-, Zeichen- und String-Literale

Aufgrund der in .NET allgegenwärtigen Typprüfungen ist es oft wichtig, bereits die Literale mit dem richtigen Typ zu definieren. Auch wenn VB.NET im Umgang mit Literalen etwas großzügiger ist, als es sein dürfte, sollten Sie wissen, wie ein Literal für den jeweiligen Typ definiert wird. In Tabelle 1 sind die wichtigsten Literal-Formate aufgeführt.

| Datentyp | Postfix | Beispiel  | Frameworktyp   |
|----------|---------|-----------|----------------|
| Integer  | -       | 123       | System.Int32   |
| Integer  | I       | 123I      | System.Int32   |
| Short    | S       | 123S      | System.Int16   |
| Long     | L       | 123L      | System.Int64   |
| Double   | -       | 123.456   | System.Double  |
| Double   | -       | 123.45E45 | System.Double  |
| Single   | F       | 123.45F   | System.Single  |
| Decimal  | D       | 123.45D   | System.Decimal |
| String   | -       | "A"       | System.String  |
| Char     | C       | "A"c      | System.Char    |

Tabelle 1: Definition von Literalen in VB.NET

### Die Methode PrintInfo

```
Public Sub PrintInfo(ByVal o As Object)
    Debug.WriteLine(o.ToString() & " [" & o.GetType.Name & "]")
End Sub
```

erzeugt bei den folgenden Aufrufen

```
PrintInfo(3.2)
PrintInfo(3.2F)
PrintInfo(3D)
PrintInfo(3.2E+25)
PrintInfo(3)
PrintInfo(3L)
```

---

```
PrintInfo(3S)
PrintInfo("A")
PrintInfo("A"c)
```

---

diese Ausgaben:

---

```
3,2 [Double]
3,2 [Single]
3 [Decimal]
3,2E+25 [Double]
3 [Int32]
3 [Int64]
3 [Int16]
A [String]
A [Char]
```

---

Wollen Sie bei Methodenaufrufen Literale übergeben, dann können Sie durch den entsprechenden Postfix gezielt eine Überladung für einen bestimmten Typ aufrufen. Bei dieser Methoden-Überladung:

---

```
Public Sub Compute(ByVal value As Integer)
    ' Berechnungen mit Integer-Wert
End Sub

Public Sub Compute(ByVal value As Short)
    ' Berechnungen mit Short-Wert
End Sub
```

---

können Sie gezielt eine der beiden Methoden mit einem Literal aufrufen:

---

```
Compute(123)    ' ruft die erste Variante auf
Compute(123S)   ' ruft die zweite Variante auf
```

---

Alternativ können Sie auch den Typ mit einem `TypeCast` anpassen, entweder allgemein mit `CType` oder speziell mit den VB-Typumwandlungen `CSng`, `CShort`, `CChar` usw.

Hexadecimal-Literale werden definiert, wie schon früher in VB6: mit vorangestelltem `&H`:

---

```
Dim h As Integer = &H1000    ' weist h den Wert 4096 zu
```

---

Analog gilt für Oktal-Literale ein vorangestelltes `&O`:

---

```
Dim o As Integer = &O1000    ' weist o den Wert 512 zu
```

---

Datums-Literale werden in amerikanischer Notation angegeben (# Monat / Tag / Jahr# bzw. # Stunde : Minute : Sekunde# oder Kombinationen hieraus). Beispiele:

```
Dim d As DateTime
d = #10/20/2003#
d = #2:20:30 PM#
d = #9/10/2004 2:20:30 PM#
```

## 2 Ganzzahlen dual, oktal oder hexadezimal darstellen

Benötigen Sie einen String, der die duale, oktale oder hexadezimale Repräsentation einer ganzen Zahl darstellt, dann können Sie diesen durch Aufruf der statischen Methode `Convert.ToString (Wert, Basis)` abrufen. `Basis` gibt die Darstellungsbasis an. Erlaubt sind die Werte 2 (dual), 8 (oktal) und 16 (hexadezimal).

Die Aufrufe

```
Debug.WriteLine(Convert.ToString(1024, 2))
Debug.WriteLine(Convert.ToString(1024, 8))
Debug.WriteLine(Convert.ToString(1024, 16))
```

erzeugen die Ausgaben

```
100000000000
2000
400
```

## 3 String mit dualer, oktaler oder hexadezimaler Darstellung in Zahlenwert wandeln

Umgekehrt geht die Wandlung selbstverständlich auch. Je nach benötigtem Datentyp können Sie eine der `ToXXX`-Methoden von `Convert` verwenden:

- ▶ `Convert.ToByte (text, base)`
- ▶ `Convert.ToInt16 (text, base)`
- ▶ `Convert.ToInt32 (text, base)`
- ▶ `Convert.ToInt64 (text, base)`

`base` gibt hier wieder die Darstellungsbasis an und kann die Werte 2, 8 und 16 annehmen. Beispielsweise wird mit

```
Dim b As Byte = Convert.ToByte("0000111", 2)
```

der Variablen `b` der Wert 7 zugewiesen.

## 4 Zahlenwerte formatieren

Jeder numerische Datentyp verfügt über eine eigene Überschreibung der Methode `ToString` mit verschiedenen Überladungen. Eine der Überladungen nimmt als Parameter einen Format-String an, mit dem Sie definieren können, wie der Zahlenwert ausgegeben werden soll. Wird kein Format-String übergeben, wird eine Standard-Formatierung vorgenommen (`Format General`). Die

möglichen Formatierungen sind von Typ zu Typ unterschiedlich. Grundsätzlich gibt es standardisierte Formatbezeichner wie D, E, X usw. sowie benutzerdefinierte Formate wie #.## und 0000. Hier einige Beispiele:

Verschiedene Formatierungen für Integer-Werte:

---

```
Dim value As Integer = 876
Debug.WriteLine(value.ToString("00000"))
Debug.WriteLine(value.ToString("d"))
Debug.WriteLine(value.ToString("e"))
Debug.WriteLine(value.ToString("X"))
```

---

erzeugen die Ausgabe

---

```
00876
876
8,760000e+002
36C
```

---

Oder für Double-Werte:

---

```
Dim value As Double = 0.43219
Debug.WriteLine(value.ToString("G"))
Debug.WriteLine(value.ToString("00.00"))
Debug.WriteLine(value.ToString("##.##"))
Debug.WriteLine(value.ToString("0.00%"))
Debug.WriteLine(value.ToString("E"))
```

---

mit den Ausgaben

---

```
0,43219
00,43
,43
43,22%
4,321900E-001
```

---

Bei der Verwendung von Dezimal- und Tausendertrennzeichen werden die länderspezifischen Einstellungen des Betriebssystems berücksichtigt. Die Beispielausgaben wurden auf einem deutschen Betriebssystem erzeugt.

Zur Formatierung von Datum und Uhrzeit siehe [2.36](#)((Umgang mit Datum und Uhrzeit)).

## 5 Positive und negative Zahlen unterschiedlich formatieren

In manchen Fällen müssen positive und negative Werte unterschiedlich behandelt werden. Hierfür lässt sich der Format-String in Abschnitte aufteilen. Diese Abschnitte werden mit einem Semikolon getrennt. Zwei oder drei Abschnitte sind möglich.



Bei zwei Abschnitten gibt der erste das Format für positive Zahlen und Null an, der zweite das Format für negative Zahlen. Bei drei Abschnitten gibt der erste das Format für positive Zahlen, der zweite für negative Zahlen und der dritte für den Wert Null an.

Die Ausgabe in der For-Schleife

```
For i As Integer = -1 To 1
    Debug.WriteLine(i.ToString("Positiv: +00;Negativ: -00;Null: 00"))
Next
```

erzeugt die Ausgaben:

```
Negativ: -01
Null:    00
Positiv: +01
```

## 6 Zusammengesetzte Formatierungen

Mit Hilfe der Methode `String.Format` können Sie auf einfache Weise komplexe Werte formatiert zusammenführen. Als ersten Parameter übergeben Sie hierzu einen Format-String. Die benötigten Werte sind die nachfolgenden Parameter:

```
String.Format ( Formatstring, Wert1, Wert2, Wert3 ... )
```

Der Format-String kann beliebigen Text beinhalten. Um an einer bestimmten Position einen der übergebenen Werte einzusetzen, geben Sie dessen Index in geschweiften Klammern an. Beispiel:

```
Dim d As Double = 54.293
Dim i As Integer = 200
Dim x As Integer = 1023
Dim p As Double = 0.55
Dim t As String
t = String.Format("d: {0}, i: {1}, x: {2}, p:{3}", d, i, x, p)
Debug.WriteLine(t)
```

Erzeugt die Ausgabe:

```
d: 54,293, i: 200, x: 1023, p:0,55
```

Für jeden Parameter können Sie, abgetrennt durch einen Doppelpunkt, die Formatierung wählen. Der Format-String

```
"d: {0:0.0}, i: {1:0000}, x: {2:X}, p:{3:0.0%}"
```

eingesetzt in obigem Code führt zu folgender Ausgabe:

```
d: 54,3, i: 0200, x: 3FF, p:55,0%
```

## 7 Format-Provider für eigene Klassen definieren

Auch für Ihre eigenen Klassen können Sie Formatierungen durch Format-Strings unterstützen. Hierzu müssen Sie lediglich die Schnittstelle `IFormattable` implementieren und die Methode `ToString` mit den entsprechenden Parametern überladen. Sie können beliebige Format-Strings unterstützen. Einzig das Format `G` ist Pflicht.

In Listing 1 sehen Sie ein Beispiel für die Klasse `Vector`, die die Formate `X`, `Y`, `Z` und `G` unterstützt. Für `X` und `Y` wird jeweils der Zahlenwert der betreffenden Komponente zurückgegeben, bei `Z` eine spezielle Formatierung mit spitzen Klammern und bei `G` und allen anderen die Standard-Formatierung, die auch die `ToString`-Überladung ohne Parameter zurückgibt.

---

```
Public Class Vector
    Implements IFormattable

    Public X, Y As Double

    Public Sub New(ByVal x As Double, ByVal y As Double)
        Me.X = x
        Me.Y = y
    End Sub

    Public Overloads Overrides Function ToString() As String
        Return String.Format("{0},{1}", X, Y)
    End Function

    Public Overloads Function ToString(ByVal format As String, _
        ByVal formatProvider As System.IFormatProvider) As String _
        Implements System.IFormattable.ToString

        ' Gewünschtes Format berücksichtigen
        Select Case format
            Case "X", "x" : Return X.ToString("0.00")
            Case "Y", "y" : Return Y.ToString("0.00")
            Case "Z", "z" : Return String.Format( _
                "<{0:0.00} - {1:0.00}>", X, Y)
            Case "G", "g" :
            Case Else : Return Me.ToString()
        End Select
    End Function
End Class
```

---

*Listing 1: Unterstützen von Format-Anweisungen durch Implementierung der Schnittstelle `IFormattable`*

### Die Ausgaben

---

```
Dim V1 As New Vector(4.283, 6.733)
Dim V2 As New Vector(21.4, 55.2)
```

---

---

```
Debug.WriteLine(String.Format("{0} und {1}", V1, V2))
Debug.WriteLine(String.Format("{0:z} und {1:z}", V1, V2))
Debug.WriteLine(String.Format("{0:x} und {0:y}", V1))
```

---

führen zu folgendem Ergebnis:

---

```
(4,283,6,733) und (21,4,55,2)
<4,28 - 6,73> und <21,40 - 55,20>
4,28 und 6,73
```

---

## 8 Ausgaben in länderspezifischen Formaten

Formatierte Zahlen- und Datumsausgaben erfolgen stets in den kulturspezifischen Formaten, wie sie in den Ländereinstellungen des Betriebssystems vorgegeben worden sind. Sie können diese Einstellung aber temporär für den laufenden Thread ändern, um nachfolgende Ausgaben in einem länderspezifischen Format eines anderen Landes zu formatieren.

Zunächst benötigen Sie hierzu die Referenz des `CultureInfo`-Objektes der gewünschten Kultur. Diese erhalten Sie beispielsweise über die Methode `CultureInfo.CreateSpecificCulture`. Übergeben können Sie den aus Sprachcode und Landescode bestehenden Namen (siehe RFC 1766), z.B. de-DE, en-US, nl-NL usw.

Anschließend weisen Sie die Referenz der `CurrentCulture`-Eigenschaft des laufenden Threads zu. Danach erfolgen alle Formatierungen mit dem neu eingestellten Format, bis Sie wieder ein anderes `CultureInfo`-Objekt zuweisen.

Ein Beispiel hierzu: Der folgende Code

---

```
Debug.WriteLine("ISO-Datum: " & DateTime.Now.ToString("s"))
Debug.WriteLine("Datum : " & DateTime.Now.ToString())
Debug.WriteLine("Double : " & 123456.78.ToString())
Debug.WriteLine("Double : " & 123456.78.ToString("#,###.##"))
Debug.WriteLine("Währung: " & 123.456.ToString("C"))
```

---

erzeugt auf einem Betriebssystem mit der Ländereinstellung DEUTSCH-DEUTSCHLAND diese Ausgabe:

---

```
ISO-Datum: 2003-10-13T13:35:06
Datum : 13.10.2003 13:35:06
Double : 123456,78
Double : 123.456,78
Währung: 123,46 _
```

---

Möchten Sie nun die gleiche Ausgabe im Format ENGLISCH-AUSTRALIEN vornehmen, dann weisen Sie dem aktuellen Thread das für Australien spezifische `CultureInfo`-Objekt zu:

---

```
Imports System.Globalization
Imports System.Threading
```

---

...

```
' CultureInfo-Objekt anlegen
Dim cultInfo As CultureInfo
cultInfo = CultureInfo.CreateSpecificCulture("en-AU")

' Aktuelle Kultur ändern
Thread.CurrentThread.CurrentCulture = cultInfo
```

Die Ausgabe des obigen Codes sieht danach so aus:

```
ISO-Datum: 2003-10-13T13:34:14
Datum   : 13/10/2003 1:34:14 PM
Double  : 123456.78
Double  : 123,456.78
Währung : $123.46
```

Wenn Sie später wieder auf das ursprüngliche Format zurückschalten wollen, speichern Sie sich am besten in einer Hilfsvariablen vor der ersten Änderung die Referenz der aktuellen Einstellung und weisen diese später `Thread.CurrentThread.CurrentCulture` wieder zu.

Auf der Buch-CD finden Sie ein Beispielprogramm, mit dem Sie die Ausgaben in unterschiedlichen Ländereinstellungen ausprobieren können (siehe Abbildung 1). Wie Sie eine TreeView mit den Namen der Ländereinstellungen füllen, erfahren Sie im nächsten Rezept.

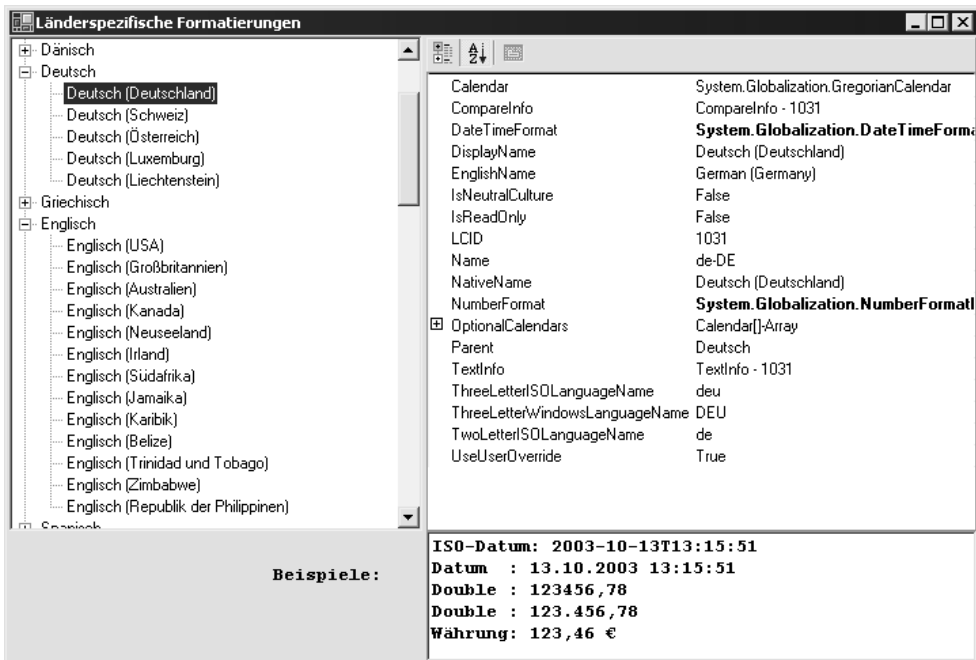


Abbildung 1: Länderspezifische Formatierungen von Zahlen- und Datumswerten

## 9 Informationen zu länderspezifischen Einstellungen abrufen

Alle von Windows vorgesehenen länderspezifischen Informationen und Formate lassen sich auch im Programmcode abrufen. Eine Liste der `CultureInfo`-Objekte erhalten Sie durch den Aufruf der statischen Methode `CultureInfo.GetCultures`. Über einen Parameter vom Typ `CultureTypes` können Sie festlegen, welche Art von `CultureInfo`-Objekten als Array zurückgegeben wird. Tabelle 2 gibt Aufschluss über die Bedeutung der möglichen Konstanten.

| Konstante                           | Bedeutung                                                                                                                |
|-------------------------------------|--------------------------------------------------------------------------------------------------------------------------|
| <code>AllCultures</code>            | Alle Kulturen                                                                                                            |
| <code>InstalledWin32Cultures</code> | Alle Kulturen, die im Betriebssystem installiert sind. In der Regel sind das weniger als die vom Framework unterstützten |
| <code>NeutralCultures</code>        | Kulturen, die nur einer Sprache zugeordnet sind, aber keinem bestimmten Land (z.B. Deutsch).                             |
| <code>SpecificCultures</code>       | Einem bestimmten Land zugeordnete Kulturen (z.B. Deutsch-Deutschland)                                                    |

Tabelle 2: Enumerationskonstanten für den Aufruf von `CultureInfo.GetCultures`

Um ein `TreeView`-Steuerelement mit den Namen der verfügbaren Kulturen zu füllen, wie in Abbildung 1 und Abbildung 2 zu sehen ist, werden zunächst alle neutralen Kulturen, also alle, die keinem spezifischen Land oder Region zugeordnet sind, abgerufen und als Stammknoten der `TreeView` hinzugefügt (siehe Listing 2). Für die Anzeige des Namens können Sie wählen, welche Darstellung der Bezeichnung Sie verwenden möchten:

- ▶ Betriebssystemspezifische Darstellung (Name in der Sprache des Betriebssystems, Eigenschaft `DisplayName`)
- ▶ Englische Bezeichnung (Eigenschaft `EnglishName`)
- ▶ Den Namen in der Landessprache (z.B. `ة يبرعل ا` für Arabisch, Eigenschaft `NativeName`)
- ▶ Das Zwei/Vier-Buchstaben-Kürzel (z.B. `en` für Englisch bzw. `en-us` für Englisch-USA, Eigenschaft `Name`)
- ▶ Das Drei-Buchstaben-Kürzel (z.B. `DEU` oder `deu` für Deutsch, Eigenschaften `ThreeLetterWindowsLanguageName` bzw. `ThreeLetterISOLanguageName`)
- ▶ sowie weitere weniger wichtige.

Anschließend werden alle länderspezifischen `CultureInfo`-Objekte abgefragt. Für jedes Objekt dieser Liste wird in der `TreeView` der zugehörige Stammknoten gesucht (das neutrale `CultureInfo`-Objekt mit der gleichen Sprachkennung, Eigenschaft `ThreeLetterISOLanguageName`).

Für jeden Knoten wird in der `Tag`-Eigenschaft die Referenz des zugehörigen `CultureInfo`-Objektes für den späteren Zugriff gespeichert.

```
Dim tn As TreeNode

' Alle neutralen Kulturen durchlaufen und als Stammknoten
' in der TreeView eintragen
```

Listing 2: Füllen einer `TreeView` mit den Namen der verfügbaren Kulturen

|                           |
|---------------------------|
| Basics                    |
| Datum/<br>Zeit            |
| Anwen-<br>dungen          |
| Zeich-<br>nen             |
| Bild-<br>bearbei-<br>tung |
| Win-<br>dows<br>Forms     |
| Controls                  |
| Propor-<br>tivityGrid     |
| Datei-<br>system          |
| Netz-<br>werk             |
| System                    |
| Daten-<br>banken          |
| XML                       |
| Wissen-<br>schaft         |
| Ver-<br>schiede-<br>nes   |

---

```

For Each cult As CultureInfo In CultureInfo.GetCultures( _
    CultureTypes.NeutralCultures)

    ' Displayname im Baum anzeigen
    tn = New TreeNode(cult.DisplayName)

    ' Verweis auf CultureInfo-Objekt speichern
    tn.Tag = cult

    ' Knoten hinzufügen
    TVCultures.Nodes.Add(tn)

```

Next

```

' Alle länderspezifischen CultureInfo-Objekte durchlaufen
For Each cult As CultureInfo In CultureInfo.GetCultures( _
    CultureTypes.SpecificCultures)

    ' Displayname für TreeView-Anzeige
    tn = New TreeNode(cult.DisplayName)

    ' Verweis auf CultureInfo-Objekt speichern
    tn.Tag = cult

    ' Stammknoten suchen, der die neutrale Kultur enthält
    For Each n As TreeNode In TVCultures.Nodes

        ' CultureInfo-Objekt des Stammknotens
        Dim ncult As CultureInfo = DirectCast(n.Tag, CultureInfo)

        ' Ist es der richtige?
        If ncult.ThreeLetterISOLanguageName = _
            cult.ThreeLetterISOLanguageName Then

            ' Ja, hier den Knoten einfügen
            n.Nodes.Add(tn)

            ' Weiter mit dem nächsten länderspez. CultureInfo-Objekt
            Exit For
        End If
    Next
Next

```

---

Listing 2: Füllen einer TreeView mit den Namen der verfügbaren Kulturen (Forts.)

Jedes `CultureInfo`-Objekt verfügt über eine Reihe weiterer Informationen. Diese Informationen lassen sich leicht mit Hilfe eines Eigenschaftsfensters (`PropertyGrid-Control`) darstellen. Dazu wird der Eigenschaft `SelectedObject` des `PropertyGrid-Controls` die Referenz des ausgewählten `CultureInfo`-Objektes zugewiesen (Listing 3). Näheres zum `PropertyGrid`-Steuerelement erfahren Sie in Kapitel 9 ((Eigenschaftsfenster (`PropertyGrid`))).

Von besonderem Interesse im Zusammenhang mit den Zahlen- und Datumsformaten sind noch die Eigenschaften `DateTimeFormat` und `NumberFormat`. Sie verweisen auf Objekte vom Typ

DateTimeFormatInfo bzw. NumberFormatInfo, die viele weitere Details zu den jeweiligen Formaten bereitstellen. In Abbildung 2 sehen Sie zwei weitere Eigenschaftsfenster, die diese zusätzlichen Informationen anzeigen. Die Informationen stehen jedoch nur zur Verfügung, wenn die ausgewählte Kultur nicht neutral ist.

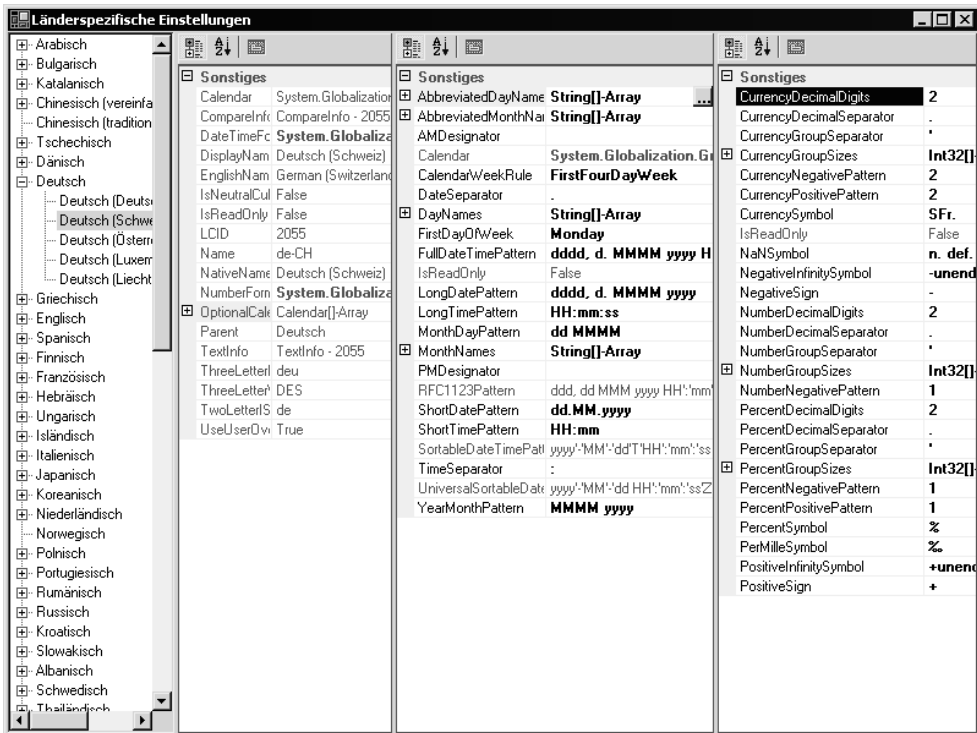


Abbildung 2: Länderspezifische Informationen abrufen und anzeigen

```
Private Sub TVCultures_AfterSelect(ByVal sender As System.Object, _
    ByVal e As System.Windows.Forms.TreeViewEventArgs) _
    Handles TVCultures.AfterSelect

    ' Knoten expandieren
    e.Node.Expand()

    ' Eigenschaften des CultureInfo-Objektes anzeigen
    PGCultureInfo.SelectedObject = e.Node.Tag

    ' Wenn es sich nicht um ein neutrales CultureInfo-Objekt handelt,
    ' dann auch die Informationen zu Datums- und Zahlenformaten
    ' anzeigen
    Dim ncult As CultureInfo = DirectCast(e.Node.Tag, CultureInfo)
    If ncult.IsNeutralCulture Then
        PGDateTimeInfo.SelectedObject = Nothing
        PGNumberInfo.SelectedObject = Nothing
    End If
End Sub
```

Listing 3: Anzeigen weiterer Informationen bei Auswahl einer Kultur in der TreeView

## Basics

### Datum/ Zeit

### Anwen- dungen

### Zeich- nen

### Bild- bearbei- tung

### Win- dows Forms

### Controls

### Proper- tyGrid

### Datei- system

### Netz- werk

### System

### Daten- banken

### XML

### Wissen- schaft

### Ver- schiede- nes

---

```

Else
    PGDateTimeInfo.SelectedObject = ncult.DateTimeFormat
    PGNumberInfo.SelectedObject = ncult.NumberFormat
End If

End Sub

```

---

*Listing 3: Anzeigen weiterer Informationen bei Auswahl einer Kultur in der TreeView (Forts.)*

Der auf der Buchseite zur Verfügung stehende Platz ist leider begrenzt, so dass die Spalten in der Abbildung nur eingeschränkt sichtbar sind. Das vollständige Programm steht Ihnen aber auf der Buch-CD zur Verfügung. Alle Steuerelemente sind durch Splitter-Controls voneinander getrennt, so dass Sie die Spaltenbreiten anpassen können.

## 10 Größter und kleinster Wert eines numerischen Datentyps

Jeder der numerischen Basistypen verfügt über die konstanten Felder `MinValue` und `MaxValue`, die Auskunft über seinen Wertebereich geben. Wie in Listing 4 dargestellt, können Sie so zur Laufzeit diese Werte ermitteln und nutzen. Der Code im Listing füllt eine `ListView` mit den abgefragten Werten ().

---

```

Dim lvi As ListViewItem

' Byte
lvi = LVMinMax.Items.Add("Byte")
lvi.SubItems.Add(Byte.MinValue.ToString())
lvi.SubItems.Add(Byte.MaxValue.ToString())

' Short
lvi = LVMinMax.Items.Add("Short")
lvi.SubItems.Add(Short.MinValue.ToString())
lvi.SubItems.Add(Short.MaxValue.ToString())

' Integer
lvi = LVMinMax.Items.Add("Integer")
lvi.SubItems.Add(Integer.MinValue.ToString())
lvi.SubItems.Add(Integer.MaxValue.ToString())

' Long
lvi = LVMinMax.Items.Add("Long")
lvi.SubItems.Add(Long.MinValue.ToString())
lvi.SubItems.Add(Long.MaxValue.ToString())

' Single
lvi = LVMinMax.Items.Add("Single")
lvi.SubItems.Add(Single.MinValue.ToString())
lvi.SubItems.Add(Single.MaxValue.ToString())

```

---

*Listing 4: ListView mit Informationen zu den Wertebereichen der numerischen Datentypen füllen*



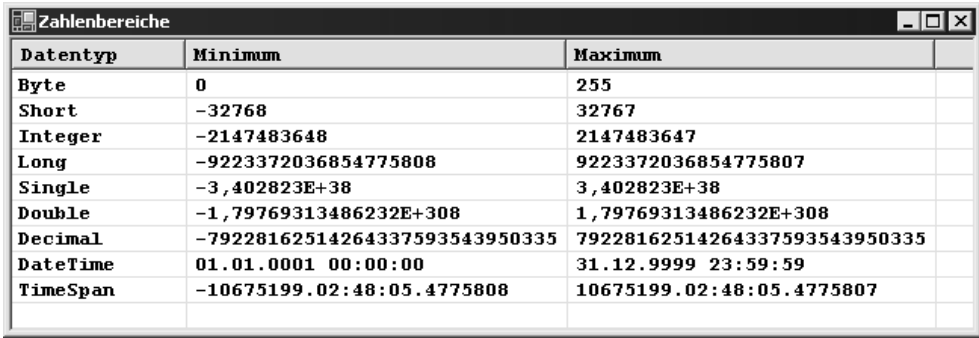
```
' Double
lvi = LVMinMax.Items.Add("Double")
lvi.SubItems.Add(Double.MinValue.ToString())
lvi.SubItems.Add(Double.MaxValue.ToString())

' Decimal
lvi = LVMinMax.Items.Add("Decimal")
lvi.SubItems.Add(Decimal.MinValue.ToString())
lvi.SubItems.Add(Decimal.MaxValue.ToString())

' DateTime
lvi = LVMinMax.Items.Add("DateTime")
lvi.SubItems.Add(DateTime.MinValue.ToString())
lvi.SubItems.Add(DateTime.MaxValue.ToString())

' TimeSpan
lvi = LVMinMax.Items.Add("TimeSpan")
lvi.SubItems.Add(TimeSpan.MinValue.ToString())
lvi.SubItems.Add(TimeSpan.MaxValue.ToString())
```

Listing 4: ListView mit Informationen zu den Wertebereichen der numerischen Datentypen füllen



| Datentyp | Minimum                        | Maximum                       |
|----------|--------------------------------|-------------------------------|
| Byte     | 0                              | 255                           |
| Short    | -32768                         | 32767                         |
| Integer  | -2147483648                    | 2147483647                    |
| Long     | -9223372036854775808           | 9223372036854775807           |
| Single   | -3,402823E+38                  | 3,402823E+38                  |
| Double   | -1,79769313486232E+308         | 1,79769313486232E+308         |
| Decimal  | -79228162514264337593543950335 | 79228162514264337593543950335 |
| DateTime | 01.01.0001 00:00:00            | 31.12.9999 23:59:59           |
| TimeSpan | -10675199.02:48:05.4775808     | 10675199.02:48:05.4775807     |

Abbildung 3: Die Zahlenbereiche der numerischen Datentypen

# 11 Berechnen der signifikanten Vorkommastellen

In manchen Situationen ist es erforderlich, Zahlenformate dynamisch an veränderliche Wertebereiche anzupassen. Ein Beispiel hierfür ist der Aufbau der Koordinatenskalierung eines Diagramms. Für Zahlen im Bereich von 0.0 bis 10.0 muss ein anderes Format eingestellt werden als für Zahlen im Bereich von -5000 bis +5000. Um ein solches Format zur Laufzeit festlegen zu können, muss die Anzahl der signifikanten Vorkommastellen ermittelt werden.

Der Zehner-Logarithmus eines Wertes gibt einen ersten Anhaltspunkt für die Anzahl der Vorkommastellen (Dekade). Für Zahlen, die keine Zehnerpotenz darstellen, liegt er immer unterhalb der gesuchten Dekade und immer oberhalb der nächstkleineren Dekade. In diesen Fällen genügt es, auf die nächste ganze Zahl aufzurunden, um die Anzahl der Stellen zu berechnen.

Handelt es sich bei dem Wert um eine Zehnerpotenz, dann muss auf den Wert des Logarithmus 1 aufaddiert werden (Beispiel:  $\text{Log}_{10}(100) = 2$ , Anzahl Stellen = 3).

Die vollständige Implementierung der Funktion sehen Sie in Listing 5. Nach Entfernen des Vorzeichens wird für Werte, die kleiner sind als 10, der Wert 1 zurückgegeben. Für alle anderen Werte wird der Logarithmus berechnet. Ist er identisch mit dem aufgerundeten Ergebnis, dann handelt es sich um eine glatte Zehnerpotenz, so dass der um 1 erhöhte Wert zurückgegeben werden muss. Anderenfalls wird der aufgerundete Logarithmuswert zurückgegeben.

---

```
Public Shared Function SignificantDigits(ByVal value As Double) _
    As Integer

    ' Absolutwert bestimmen
    Dim val As Double = Math.Abs(value)

    ' Werte kleiner als 10 haben immer eine Vorkommastelle
    If val < 10.0 Then Return 1

    ' Logarithmus und Dekade berechnen
    Dim log As Double = Math.Log10(val)
    Dim decade As Double = Math.Ceiling(log)

    ' Glatte Werte (für 10, 100, 1000 etc.) müssen um 1 erhöht werden
    If decade = log Then Return CInt(decade + 1)

    ' Ansonsten passt es
    Return CInt(decade)

End Function
```

---

Listing 5: Berechnen der signifikanten Vorkommastellen eines Double-Wertes

### Ein kleines Testbeispiel

---

```
Private Sub Output(ByVal value As Double)
    ' Testausgabe formatieren
    Debug.WriteLine(String.Format( _
        "{0} hat {1} signifikante Vorkommastellen", _
        value, SignificantDigits(value)))
End Sub
```

---

mit den folgenden Aufrufen

---

```
Output(0.5)
Output(7.9)
Output(10.0)
Output(99.9999)
Output(100)
Output(101.5)
Output(123456789.0)
Output(-987654321.1)
```

---

erzeugt diese Ausgabe:

---

```
0,5 hat 1 signifikante Vorkommastellen
7,9 hat 1 signifikante Vorkommastellen
10 hat 2 signifikante Vorkommastellen
99,9999 hat 2 signifikante Vorkommastellen
100 hat 3 signifikante Vorkommastellen
101,5 hat 3 signifikante Vorkommastellen
123456789 hat 9 signifikante Vorkommastellen
-987654321,1 hat 9 signifikante Vorkommastellen
```

---

Die beschriebene Implementierung funktioniert bis ca. 13 Stellen. Darüber hinaus ergeben sich Fehler aufgrund der Rechengenauigkeiten.

## 12 Lange einzeilige Texte umbrechen

Lange Textzeilen, die zum Beispiel aus Internet-Quellen geladen wurden, lassen sich schlecht in einem ToolTip oder in einer MessageBox verwenden, da sie nicht automatisch umgebrochen werden. Den Umbruch können Sie mit Hilfe der Methode `BreakString` (Listing 6) selbst vornehmen.

Die Methode erwartet als Parameter den umzubrechenden Text sowie eine Angabe, nach wie vielen Zeichen (ungefähr) der Umbruch erfolgen soll. In einer Schleife wird ab der angegebenen Position nach dem nächsten Leerzeichen gesucht und an dieser Stelle ein Zeilenumbruch eingefügt. Der neu erzeugte Text wird als Funktionswert zurückgegeben.

---

```
Public Function BreakString(ByVal theString As String, _
    ByVal approxLineWidth As Integer) As String

    Dim newString As String
    Dim pos As Integer

    ' Umbrechen nur, solange der Text länger ist als die
    ' vorgegebene Breite
    Do While theString.Length > approxLineWidth

        ' Nächstes Leerzeichen suchen
        pos = theString.IndexOf(" ", approxLineWidth)

        ' Wenn keins mehr gefunden wird,
        If pos = -1 Then Return newString & theString

        ' Textfragment und Zeilenumbruch anhängen
        newString = newString & theString.Substring(0, pos) _
            & Environment.NewLine

        ' Rest weiterverarbeiten
        theString = theString.Substring(pos + 1)

    Loop
```

---

Listing 6: Umbrechen langer Textzeilen

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

```
' Text anhängen und zurückgeben
Return newString & theString
```

```
End Function
```

---

*Listing 6: Umbrechen langer Textzeilen (Forts.)*

Die überladene Methode `BreakString(string)` (Listing 7) arbeitet mit einer voreingestellten Umbruchspostition.

---

```
Public Function BreakString(ByVal theString As String) As String
    Return BreakString(theString, 70)
End Function
```

---

*Listing 7: Umbrechen nach ca. 70 Zeichen*

## 13 Zahlenwerte kaufmännisch runden

Das mathematische Runden, das die Methode `Math.Round` implementiert, gehorcht folgenden Regeln:

1. Folgt auf die letzte beizubehaltende Ziffer eine 0, 1, 2, 3 oder 4, so wird sie unverändert übernommen.
2. Folgt auf die letzte beizubehaltende Ziffer eine 9, 8, 7, 6 oder eine 5, gefolgt von weiteren Ziffern, die nicht alle Null sind, so wird die letzte beizubehaltende Ziffer um eins erhöht. Ist die letzte beizubehaltende Ziffer eine 9, so wird sie durch 0 ersetzt und der Wert der vorhergehenden Ziffer um eins erhöht (usw.).
3. Folgt auf die letzte beizubehaltende Ziffer lediglich eine 5 oder eine 5, auf die nur Nullen folgen, so wird abgerundet, falls die letzte beizubehaltende Ziffer einen geraden Ziffernwert besitzt, sonst wird aufgerundet.

Aus 1,25 wird somit 1,2 und aus 1,15 wird ebenfalls 1,2.

Kaufleute allerdings runden anders: Im Falle von Regel 3 wird grundsätzlich aufgerundet (positive Zahlen). Negative Zahlen werden gerundet, indem man den Betrag rundet und anschließend wieder das Vorzeichen vorweg schreibt.

Listing 8 zeigt eine mögliche Implementierung des kaufmännischen Rundens. Neben dem zu runden Wert wird angegeben, auf wie viele Nachkommastellen gerundet werden soll. Der Wert wird mit der entsprechenden Zehnerpotenz multipliziert, um die beizubehaltenden Ziffern nach links vor das Komma zu schieben. Anschließend wird ein Offset von 0,5 aufaddiert und der Nachkommateil abgeschnitten. Die Division durch die Zehnerpotenz, mit der zuvor multipliziert wurde, ergibt dann das gesuchte Ergebnis. Gerechnet wird grundsätzlich mit dem Absolutwert. Das Vorzeichen wird erst bei der Rückgabe des Ergebnisses berücksichtigt.

---

```
Public Shared Function RoundCommercial(ByVal value As Double, _
    ByVal decimals As Integer) As Double

    ' Vorzeichen merken
    Dim sign As Double = Math.Sign(value)
```

---

*Listing 8: Kaufmännisches Runden*

```

' Mit Absolutwert weiterrechnen
value = Math.Abs(value)

' Zehnerpotenz zum Verschieben der Ziffern nach links
Dim decFact As Double = Math.Pow(10, decimals)

' Alle bleibenden Stellen nach links schieben
value *= decFact

' 0.5 addieren
value += 0.5

' Nachkommastellen abschneiden und durch Zehnerpotenz teilen
Return sign * Math.Floor(value) / decFact

```

End Function

Listing 8: Kaufmännisches Runden (Forts.)

Nachfolgend einige Beispielberechnungen mit der Methode `RoundCommercial`:

```

1,1 auf 1 Stellen gerundet: 1,1
1,5 auf 1 Stellen gerundet: 1,5
1,5 auf 0 Stellen gerundet: 2
1,15 auf 1 Stellen gerundet: 1,2
1,25 auf 1 Stellen gerundet: 1,3
1,15363 auf 2 Stellen gerundet: 1,15
1,15 auf 2 Stellen gerundet: 1,15
1,14999 auf 2 Stellen gerundet: 1,15
100,125 auf 2 Stellen gerundet: 100,13
100,135 auf 2 Stellen gerundet: 100,14
-1,449 auf 1 Stellen gerundet: -1,4
-1,45 auf 1 Stellen gerundet: -1,5
-1,46 auf 1 Stellen gerundet: -1,5
12345,5 auf 0 Stellen gerundet: 12346
12345 auf -1 Stellen gerundet: 12350
12345 auf -2 Stellen gerundet: 12300

```

Beachten Sie, dass `RoundCommercial` auch Vorkommastellen runden kann. In den letzten beiden Ausgabzeilen wurden negative Werte für die Kommastellen angegeben. Dadurch wurden die letzten ein bzw. zwei Ziffern vor dem Komma gerundet.

## 14 Überprüfen, ob ein Bit in einem Integer-Wert gesetzt ist

Bitoperationen werden in der neuen Version von VB.NET dadurch vereinfacht, dass nun auch endlich die Verschiebe-Operatoren unterstützt werden. Mit `<<` lässt sich ein Wert um eine vorgegebene Anzahl von Bits nach links schieben, mit `>>` nach rechts. Mit Hilfe dieser Operatoren lassen sich schnell Bitmasken erstellen, die für weitere Operationen benötigt werden.

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

Listing 9 zeigt die Methode `TestBit`, die überprüft, ob bei einem übergebenen Wert vom Typ `Integer` das im Parameter `bitNumber` angegebene Bit gesetzt ist. Die Nummerierung der Bits ist beginnend mit dem niederwertigsten (0) bis zum hochwertigsten (31). Mit Hilfe des Verschiebeoperators wird eine Bitmaske generiert, die an der angegebenen Bitposition ein gesetztes Bit und an allen anderen Positionen nicht gesetzte Bits aufweist.

Die `And`-Verknüpfung der Maske mit dem Wert liefert als Ergebnis 0, wenn das Bit nicht gesetzt ist, und einen Wert ungleich 0, wenn das Bit gesetzt ist. Der Vergleich auf »ungleich 0« liefert somit das gewünschte Ergebnis.

---

```
Public Shared Function TestBit(ByVal value As Integer, _
    ByVal bitNumber As Integer) As Boolean

    ' Bitmaske für Verknüpfung erzeugen
    Dim mask As Integer = 1 << bitNumber

    ' Testen, ob Bit gesetzt ist
    Return (value And mask) <> 0
```

End Function

---

*Listing 9: Testen eines Bitwertes in einem Integer-Wert*

Das Beispiel steht exemplarisch für die Vorgehensweise. Sie können die Methode leicht für andere Typen (`Byte`, `Short` und `Long`) überladen. Diese Methode und auch die folgenden sind in der Klasse `BitOP` definiert.

Beispiel für Aufrufe:

---

```
Dim b As Boolean
b = BitOP.TestBit(7, 0)
b = BitOP.TestBit(7, 3)
If TestBit(X, 8) Then ...
```

---

## 15 Bit in einem Integer-Wert setzen

Um ein einzelnes Bit in einem Integer-Wert zu setzen, erzeugen Sie eine Bitmaske und setzen das Bit durch eine `Or`-Verknüpfung (Listing 10). Auch diese Methode können Sie für andere ganzzahlige Datentypen überladen.

---

```
Public Shared Sub SetBit(ByRef value As Integer, _
    ByVal bitNumber As Integer)

    ' Bitmaske für Verknüpfung erzeugen
    Dim mask As Integer = 1 << bitNumber

    ' Bit setzen
    value = value Or mask
```

End Sub

---

*Listing 10: Setzen eines Bits*

Beispiel für Aufrufe:

```
Dim i1 As Integer = 8
BitOP.SetBit(i1, 0)
```

## 16 Bit in einem Integer-Wert löschen

Ähnlich wie beim Setzen eines Bits wird eine Verknüpfung mit einer Maske ausgeführt. Die Verknüpfung ist hier eine And-Operation mit der negierten Maske (Listing 11).

```
Public Shared Sub ClearBit(ByRef value As Integer, _
    ByVal bitNumber As Integer)

    ' Bitmaske für Verknüpfung erzeugen
    Dim mask As Integer = 1 << bitNumber

    ' Bit löschen
    value = value And Not mask

End Sub
```

Listing 11: Löschen eines Bits

Beispiele für Aufrufe:

```
Dim i1 As Integer = 9
BitOP.ClearBit(i1, 0)
```

## 17 Bit in einem Integer-Wert einen bestimmten Zustand zuweisen

Soll ein Bit einen als booleschen Wert vorgegebenen Zustand annehmen, wird eine Kombination aus SetBit und ClearBit benötigt. Diese lässt sich wie in Listing 12 gezeigt zusammenfassen.

```
Public Shared Sub SetBit(ByRef value As Integer, _
    ByVal bitNumber As Integer, ByVal bitValue As Boolean)

    ' Bitmaske für Verknüpfung erzeugen
    Dim mask As Integer = 1 << bitNumber

    If bitValue Then
        ' Bit setzen
        value = value Or mask
    Else
        ' Bit löschen
        value = value And Not mask
    End If

End Sub
```

Listing 12: Überladung der Methode SetBit, um ein Bit je nach Vorgabe zu setzen oder zu löschen

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

Beispiel für Aufrufe:

---

```
Dim i1 As Integer = 8
BitOP.SetBit(i1, 0, True)
BitOP.SetBit(i1, 1, False)
```

---

## 18 Bit in einem Integer-Wert umschalten (togglen)

Mit der gleichen Herangehensweise wie in den beiden Methoden `SetBit` und `ClearBit` lässt sich auch der Wert eines einzelnen Bits umkehren. Hierzu wird der Integer-Wert mit einer `Xor`-Operation mit der Maske verknüpft (Listing 13).

---

```
Public Shared Sub ToggleBit(ByRef value As Integer, _
    ByVal bitNumber As Integer)

    ' Bitmaske für Verknüpfung erzeugen
    Dim mask As Integer = 1 << bitNumber

    ' Bit umschalten
    value = value Xor mask

End Sub
```

---

*Listing 13: Umschalten eines Bits*

Beispiele für Aufrufe:

---

```
Dim i1 As Integer = 9
BitOP.ToggleBit(i1, 0)
BitOP.ToggleBit(i1, 0)
```

---

## 19 Boolean-Array aus Bit-Informationen eines Integer-Wertes erzeugen

Die 32 Bits einer Integer-Zahl lassen sich leicht in ein `Boolean`-Array mit 32 Werten überführen. Zum Einsatz kommt die Vorgehensweise, die bereits bei der Methode `TestBit` erläutert wurde. Es wird eine Maske verwendet, bei der zunächst nur das niederwertigste Bit gesetzt ist. In einer Schleife wird über eine `And`-Operation überprüft, ob das Bit im Integer-Wert gesetzt ist, und das Ergebnis in einem zuvor angelegten `Boolean`-Array gespeichert (Listing 14). Nach der Überprüfung wird die Maske um ein Bit nach links geschoben, so dass im nächsten Schleifendurchlauf das nächsthöhere Bit betrachtet wird.

---

```
Public Shared Function BitsToBooleanArray( _
    ByVal value As Integer) As Boolean()

    ' Array mit Boolean-Werten (Größe für Integer)
    Dim bits(31) As Boolean
```

---

*Listing 14: In einer Schleife werden die Werte aller Bits überprüft*



```

' Bitmaske für das unterste Bit
Dim mask As Integer = 1

' Schleife über alle Bits
For i As Integer = 0 To 31

    ' Bitwert prüfen und speichern
    bits(i) = (value And mask) <> 0

    ' Maske für nächstes Bit setzen
    mask <<= 1

```

Next

```

' Fertiges Array zurückgeben
Return bits

```

End Function

*Listing 14: In einer Schleife werden die Werte aller Bits überprüft (Forts.)*

Beispiel für Aufrufe:

```
Dim boolArr() As Boolean = BitOP.BitsToBooleanArray(&HC056)
```

Auch diese Methode kann für andere Typen (Byte, Short und Long) überladen werden. Beachten Sie dabei bitte, dass auch die Zahlenwerte der Konstanten (im Beispiel die Konstante 31) geändert werden müssen.

## 20 Integer-Wert aus Bit-Informationen eines Boolean-Arrays zusammensetzen

Aus einem Boolean-Array (siehe vorhergehendes Rezept), das genau 32 Werte aufweisen muss, soll ein Integer-Wert zusammengesetzt werden. Die Wertigkeit eines Bits entspricht dabei dem Index im Array.

Wieder wird in einer Schleife über alle Bits eine Maske mitgeführt, die in jedem Schritt ein Bit nach links geschoben wird (Listing 15). Mit einer Or-Operation wird das Bit im Ergebnisspeicher (value) gesetzt, wenn der betreffende Wert im Boolean-Array den Wert True hat.

```

Public Shared Function BooleanArrayToIntegerBits( _
    ByVal bits() As Boolean) As Integer

    ' Sicherstellen, dass das Array genau 32 Werte enthält
    If bits.Length <> 32 Then Throw New _
        ArgumentException("Array muss genau 32 Elemente haben")

    ' Maske für Bit-Operationen
    Dim mask As Integer = 1

```

*Listing 15: Bits gemäß Boolean-Array zusammenfügen*

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

```

' Zusammengesetzter Wert
Dim value As Integer = 0

' Für alle Bits
For i As Integer = 0 To 31
    If bits(i) Then value = value Or mask
    mask <<= 1
Next
Return value
End Function

End Class

```

---

*Listing 15: Bits gemäß Boolean-Array zusammenfügen (Forts.)*

Beispiel für Aufrufe:

---

```

Dim bitArr(31) As Boolean
bitArr(1) = True
bitArr(4) = True
bitArr(6) = True
Dim v As Integer = BitOP.BooleanArrayToIntegerBits(bitArr)

```

---

## 21 Gesetzte Bits eines Integer-Wertes abfragen

Eine Methode, die Ihnen die Ordinalzahlen der gesetzten Bits eines Integer-Wertes als Array zurückliefert, zeigt Listing 16. Einer zu Beginn leeren Liste (`foundBits`, Typ `ArrayList`) werden in einer Schleife die Ordinalzahlen der Bits hinzugefügt, die gesetzt sind. Abschließend wird die Liste in ein Integer-Array konvertiert und zurückgegeben.

---

```

Public Shared Function GetBitsSet(ByVal value As Integer) _
    As Integer()

    ' Maske für Bit-Operationen
    Dim mask As Integer = 1

    ' Liste der gesetzten Bits
    Dim foundBits As New ArrayList(31)

    ' Schleife über alle Bits
    For i As Integer = 0 To 31

        ' Wenn ein Bit gesetzt ist, dessen Nummer in der Liste aufnehmen
        If (value And mask) <> 0 Then
            foundBits.Add(i)
        End If

        ' 1 Bit nach links schieben
        mask <<= 1
    Next

```

---

*Listing 16: Array mit Ordinalzahlen der gesetzten Bits erzeugen*

```
' ArrayList in Integer-Array konvertieren
Return DirectCast(foundBits.ToArray(GetType(Integer)), Integer())
```

End Function

Listing 16: Array mit Ordinalzahlen der gesetzten Bits erzeugen (Forts.)

Beispiel für Aufrufe:

```
Dim bitsSet() As Integer = BitOP.GetBitsSet(81)
```

## 22 Nicht gesetzte Bits eines Integer-Wertes abfragen

Dieses Rezept unterscheidet sich zum vorhergehenden lediglich in der Überprüfung des Bitwertes. Nur die Bits, die nicht gesetzt sind, werden hier berücksichtigt. Die Implementierung der Methode `GetBitsNotSet` zeigt Listing 17.

```
Public Shared Function GetBitsNotSet(ByVal value As Integer) _
    As Integer()

    ' Maske für Bit-Operationen
    Dim mask As Integer = 1

    ' Liste der gesetzten Bits
    Dim foundBits As New ArrayList(31)

    ' Schleife über alle Bits
    For i As Integer = 0 To 31

        ' Wenn ein Bit nicht gesetzt ist, dessen Ordinalzahl in
        ' der Liste aufnehmen
        If (value And mask) = 0 Then
            foundBits.Add(i)
        End If

        ' 1 Bit nach links schieben
        mask <<= 1
    Next

    ' ArrayList in Integer-Array konvertieren
    Return DirectCast(foundBits.ToArray(GetType(Integer)), Integer())

End Function
```

Listing 17: Array mit Ordinalzahlen der nicht gesetzten Bits erzeugen

Beispiel für Aufrufe:

```
Dim bitsNSet() As Integer = BitOP.GetBitsNotSet(81)
```

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

## 23 Konvertierungen zwischen 8-Bit, 16-Bit, 32-Bit und 64-Bit Datentypen

Beim Aufruf von API-Funktionen werden oft Byte- und Word-Werte in 32-Bit-Werten zusammengefasst. Um solche Werte zusammenfassen und wieder separieren zu können, haben wir für Sie die Klasse `ByteConverter` (Listing 18 und Listing 19) entworfen.

Die Klasse speichert intern die Daten in einem Array aus acht Byte. Diese Daten werden bei der Instanzierung gesetzt und können nicht mehr geändert werden. In Listing 18 sehen Sie eine Reihe von Konstruktoren zum Setzen der Daten. Sie können ein Objekt initialisieren mit

- ▶ einem 16-Bit-Wert (`Short`)
- ▶ einem 32-Bit-Wert (`Integer`)
- ▶ einem 64-Bit-Wert (`Long`)
- ▶ `LowWord` / `HighWord` (2 x `Short`)
- ▶ `LowDoubleWord` / `HighDoubleWord` (2 x `Integer`)
- ▶ einem Byte-Array mit 8 Byte Länge.

Je nach Typ der übergebenen Daten werden alle oder nur einige Bytes des Arrays gesetzt. Nicht gesetzte haben den Standardwert 0. Zum Kopieren der Byte-Werte wird die Methode `GetBytes` der Klasse `BitConverter` genutzt. Diese generiert ein Byte-Array, dessen Werte an die entsprechenden Positionen im Array Bytes kopiert werden.

---

```
Public Class ByteConverter

    ' Zentrale Datenhaltung in 8 Byte
    Protected Bytes(7) As Byte

    ' Initialisierung mit einem 16-Bit-Wert
    Public Sub New(ByVal value As Short)
        BitConverter.GetBytes(value).CopyTo(Bytes, 0)
    End Sub

    ' Initialisierung mit einem 32-Bit-Wert
    Public Sub New(ByVal value As Integer)
        BitConverter.GetBytes(value).CopyTo(Bytes, 0)
    End Sub

    ' Initialisierung mit einem 64-Bit-Wert
    Public Sub New(ByVal value As Long)
        BitConverter.GetBytes(value).CopyTo(Bytes, 0)
    End Sub

    ' Initialisierung mit zwei 16-Bit-Werten
    Public Sub New(ByVal loWord As Short, ByVal hiWord As Short)
        BitConverter.GetBytes(loWord).CopyTo(Bytes, 0)
        BitConverter.GetBytes(hiWord).CopyTo(Bytes, 2)
    End Sub

    ' Initialisierung mit zwei 32-Bit-Werten
```

---

Listing 18: Daten und Konstruktoren der Klasse `ByteConverter`

```

Public Sub New(ByVal LoDWord As Integer, ByVal HiDWord As Integer)
    BitConverter.GetBytes(LoDWord).CopyTo(Bytes, 0)
    BitConverter.GetBytes(HiDWord).CopyTo(Bytes, 4)
End Sub

' Initialisierung mit Byte-Array
Public Sub New(ByVal bytes() As Byte)
    ' Sicherheitsabfrage
    If bytes.Length <> 8 Then Throw New ArgumentException( _
        "Das Array muss exakt 8 Bytes lang sein")

    ' Bytes kopieren
    bytes.CopyTo(Me.Bytes, 0)
End Sub
...

End Class

```

Listing 18: Daten und Konstruktoren der Klasse ByteConverter (Forts.)

Nachdem eine Instanz angelegt und das Byte-Array mit Daten gefüllt worden ist, können Sie die Werte komplett oder in Fragmenten lesen. Mehrere Eigenschaften ermöglichen das Lesen der Werte aus dem Array (siehe Listing 19 und Tabelle 3). Beachten Sie auch, dass die Eigenschaft `AllBytes` eine Kopie des Arrays und nicht das Array `Bytes` selbst zurückgibt, damit nicht von außen die Werte des Arrays verändert werden können.

| Eigenschaft            | Wert                                                      |
|------------------------|-----------------------------------------------------------|
| <code>LoWord</code>    | gibt die unteren 16 Bit (2 Byte) zurück                   |
| <code>HiWord</code>    | gibt die oberen 16 Bit eines Word (32 Bit, Short) zurück  |
| <code>LoDWord</code>   | gibt das untere Doppelwort (32 Bit, Integer) zurück       |
| <code>HiDWord</code>   | gibt das obere Doppelwort (32 Bit, Integer) zurück        |
| <code>LongValue</code> | gibt den Wert als 64-Bit Wert (Long) zurück               |
| <code>ByteN</code>     | gibt das über den Parameter Index ausgewählte Byte zurück |
| <code>AllBytes</code>  | gibt eine Kopie des Arrays <code>Bytes</code> zurück      |

Tabelle 3: Eigenschaften der Klasse ByteConverter für den gezielten Zugriff auf die gespeicherten Bytes

```

Public Class ByteConverter
    ...

    ' Unteren 16-Bit-Wert lesen (Byte 0-1)
    Public ReadOnly Property LoWord() As Short
        Get
            Return BitConverter.ToInt16(Bytes, 0)
        End Get
    End Property

```

Listing 19: Eigenschaften zum Lesen der Werte in der Klasse ByteConverter

```
' Oberen 16-Bit-Wert lesen (Byte 2-3)
Public ReadOnly Property HiWord() As Short
    Get
        Return BitConverter.ToInt16(Bytes, 2)
    End Get
End Property

' Unteren 32-Bit-Wert lesen (Byte 0-3)
Public ReadOnly Property LoDWord() As Integer
    Get
        Return BitConverter.ToInt32(Bytes, 0)
    End Get
End Property

' Oberen 32-Bit-Wert lesen (Byte 4-7)
Public ReadOnly Property HiDWord() As Integer
    Get
        Return BitConverter.ToInt32(Bytes, 4)
    End Get
End Property

' 64-Bit-Wert lesen (Byte 0-7)
Public ReadOnly Property LongValue() As Long
    Get
        Return BitConverter.ToInt64(Bytes, 0)
    End Get
End Property

' Einzelnes Byte lesen
Public ReadOnly Property ByteN(ByVal index As Integer) As Byte
    Get
        Return Bytes(index)
    End Get
End Property

' Alle Bytes als Array zurückgeben
Public ReadOnly Property AllBytes() As Byte()
    Get
        ' Bytes kopieren und zurückgeben
        Dim arr(7) As Byte
        Bytes.CopyTo(arr, 0)
        Return arr
    End Get
End Property

End Class
```

Listing 19: Eigenschaften zum Lesen der Werte in der Klasse *ByteConverter* (Forts.)

Im folgenden Beispielcode wird eine Instanz mit einem Long-Wert instanziiert und anschließend werden die Daten mit Hilfe der verschiedenen Eigenschaften ausgegeben:

```

Dim bc As New ByteConverter(&H123456789ABCDEF0L)
Debug.WriteLine("LongValue: " & bc.LongValue.ToString("X"))
Debug.WriteLine("LoWord   : " & bc.LoWord.ToString("X"))
Debug.WriteLine("HiWord   : " & bc.HiWord.ToString("X"))
Debug.WriteLine("LoDWord  : " & bc.LoDWord.ToString("X"))
Debug.WriteLine("HiDWord  : " & bc.HiDWord.ToString("X"))
For i As Integer = 0 To 7
    Debug.WriteLine("Byte " & i.ToString() & " : " & _
        bc.ByteN(i).ToString("X"))
Next

```

Ausgabe dieses Codefragmentes:

```

LongValue: 123456789ABCDEF0
LoWord   : DEFO
HiWord   : 9ABC
LoDWord  : 9ABCDEF0
HiDWord  : 12345678
Byte 0   : FO
Byte 1   : DE
Byte 2   : BC
Byte 3   : 9A
Byte 4   : 78
Byte 5   : 56
Byte 6   : 34
Byte 7   : 12

```

Um zwei 16-Bit-Werte zu einem 32-Bit-Wert zusammenzufügen benötigen Sie eine Sequenz wie:

```

Dim vl As Short = ...
Dim vh As Short = ...
Dim bc As New ByteConverter(vl, vh)
Dim v As Integer = bc.LoDWord

```

Um aus zwei 32-Bit-Werten einen 64-Bit-Wert zusammenzusetzen können Sie diese Sequenz verwenden:

```

Dim vl As Integer = ...
Dim vh As Integer = ...
Dim bc As New ByteConverter(vl, vh)
Dim v As Long = bc.LongValue

```

## 24 Basistyp für Enumeration festlegen

Eine `Enum`-Definition ist ein Typ, der eine Reihe von Zahlenkonstanten definiert. Wird kein Typ explizit angegeben, dann basiert der `Enum`-Typ auf dem Typ `Integer`:

```

Public Enum Rooms
    Livingroom
    ...

```

### Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung  
Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schie-  
denes

---

```
End Enum
```

```
Dim r As Rooms
Dim i As Integer = r
```

---

Der Wert der Variablen `r` kann direkt als Integer-Wert weiterverwendet werden. Sie können aber auch andere Basistypen für eine Enumeration vorgeben. Erlaubt sind `Byte`, `Short`, `Integer` und `Long`. Wenn Ihnen also beispielsweise für Bitfelder der Typ `Integer` zu klein ist, weil Sie mehr als 32 Bit für unabhängige Konstanten benötigen, dann können Sie die Enumeration explizit vom Typ `Long` ableiten: (Listing 20)

---

```
Public Enum Rooms As Long
    Entrancehall = &H800000000
    Floor1Livingroom = &H1
    Floor1Kitchen = &H2
    Floor1Diningroom = &H4
    ...
    Floor2Livingroom = &H1000
    Floor2Kitchen = &H2000
    Floor2Diningroom = &H4000
    Floor2Sleepingroom = &H8000
    Floor2Corridor = &H10000
    ...
    Floor2Toilette = &H200000
    Floor3...
End Enum
```

---

*Listing 20: Auf dem Typ `Long` basierende Enumeration*

## 25 Enum-Werte ein- und ausgeben

Einen Enum-Wert können Sie durch Aufruf der Methode `ToString` in einen String wandeln. Verschiedene Formate lassen sich anwenden.

---

```
Dim r As Rooms = Rooms.Entrancehall
Debug.WriteLine(r.ToString())
Debug.WriteLine(r.ToString("D"))
Debug.WriteLine(r.ToString("X"))
```

---

Dieser Code erzeugt für die Enumeration aus Listing 20 folgende Ausgabe:

---

```
Entrancehall
34359738368
0000000800000000
```

---

Interessant ist, dass die Ausgabe im Standardformat den Namen der Enum-Konstanten wiedergibt. Das ändert sich allerdings, wenn dem vorliegenden Wert keine Konstante zugeordnet ist. Dieser Fall tritt beispielsweise ein, wenn der Variablen `r` eine Kombination verschiedener Räume zugeordnet wird:



```
Dim r As Rooms = Rooms.Entrancehall Or Rooms.Floor1Bath
...
```

erzeugt jetzt die Ausgabe:

```
34359738624
34359738624
0000000800000100
```

Um dieses Verhalten zu ändern und auch bei der Ausgabe die Namen einzelner Komponenten zu berücksichtigen, können Sie das `Flags`-Attribut einsetzen (Listing 21).

```
<Flags(>> Public Enum Rooms As Long
    Entrancehall = &H800000000
    Floor1Livingroom = &H1
    Floor1Kitchen = &H2
    Floor1Diningroom = &H4
    Floor1Sleepingroom = &H8
    ...
End Enum
```

*Listing 21: Enumeration mit Flags-Attribut*

Derselbe Code von oben gibt dann Folgendes aus:

```
Floor1Bath, Entrancehall
34359738624
0000000800000100
```

Für jedes gesetzte Bit wird der zugehörige Konstantenname gesucht und in einer Komma-getrennten Liste ausgegeben.

Das `Flags`-Attribut wird nur bei der Umwandlung in einen String berücksichtigt. Bit-Operationen können Sie unabhängig vom Vorhandensein dieses Attributs ausführen, da der Typ intern ohnehin als `Ganzzahl`-Typ verarbeitet wird. Eine andere Klasse, die auch das `Flags`-Attribut bei Enumerationen berücksichtigt, ist `PropertyGrid`. Hierzu finden Sie in Kapitel 0((

Eigenschaftsfenster (`PropertyGrid`)) nähere Informationen.

Gelegentlich ist es notwendig, einen vorliegenden String in einen Enumerationstyp zu wandeln. Das Framework stellt Ihnen hierfür die Methode `Enum.Parse` zur Verfügung. `Enum.Parse` erwartet als Parameter den Typ der Enumeration, in den gewandelt werden soll, sowie den zu wandelnden String. Eine zweite Überladung nimmt als zusätzlichen Parameter einen booleschen Wert entgegen, mit dem Sie angeben können, ob Groß-/Kleinschrift berücksichtigt werden soll oder nicht. Der Standard ist, dass Groß- und Kleinschrift unterschieden wird.

In VB.NET gibt es allerdings ein kleines syntaktisches Problem: Da `Enum` ein Schlüsselwort ist, kann es nicht direkt als Klassenname verwendet werden. Stattdessen muss der Bezeichner in eckige Klammern eingeschlossen werden. Mit diesem Code wird ein Text in den oben beschriebenen `Enum`-Typ `Rooms` gewandelt:

---

```
Dim r As Rooms
r = DirectCast([Enum].Parse(GetType(Rooms), "Floor1Child1"), Rooms)
```

---

Da `Parse` eine `Object`-Referenz zurückgibt, muss ein `TypeCast` mit `DirectCast` oder `CType` vorgenommen werden. Auch das Einlesen von Kombinationen ist möglich. Interessanterweise funktioniert das auch dann, wenn das `Flags`-Attribut nicht gesetzt ist:

---

```
r = DirectCast([Enum].Parse(GetType(Rooms),
    "Floor1Child1, Floor1Corridor"), Rooms)
```

---

Letztlich können Sie auch jederzeit einer `Enum`-Variablen einen `Integer`-Wert zuweisen:

---

```
r = CType(4, Rooms)
```

---

weist der Variablen `r` den Wert `Floor1Diningroom` zu.

## 26 Bezeichner und Werte eines Enum-Typs abfragen

Hin und wieder wird eine Liste aller zu einer Enumeration gehörenden Konstanten benötigt. Sowohl die Namen als auch die Werte können von Interesse sein. Auch hierfür stellt die Klasse `Enum` Methoden bereit.

Die Methode `GetNames` gibt ein `Array` mit den Bezeichnern der Konstanten zurück, die Methode `GetValues` ein `Array` mit den Werten. Beide sind aufsteigend nach den Werten sortiert, so dass die jeweiligen `Array`-Elemente mit gleichem Index zusammengehören. In Listing 22 sehen Sie ein Beispiel, wie die Namen und Werte der in Listing 21 abgebildeten Enumeration abgerufen und ausgegeben werden können.

---

```
' Array mit Bezeichnern abfragen
Dim names() As String = [Enum].GetNames(GetType(Rooms))

' Array mit Werten abfragen
Dim arr As Array = [Enum].GetValues(GetType(Rooms))

' In Integer-Array wandeln
Dim values() As Rooms = DirectCast(arr, Rooms())

' Ausgabe in For-Schleife
For i As Integer = 0 To names.GetUpperBound(0)
    Debug.WriteLine(names(i).PadRight(20) & " : " & _
        values(i).ToString("X"))
Next
```

---

Listing 22: Abrufen der Bezeichner und der Werte eines Enumerationstyps

Der Code des Listings gibt den nachfolgenden Text aus:

---

```
Floor1Livingroom    : 0000000000000001
Floor1Kitchen       : 0000000000000002
Floor1Diningroom    : 0000000000000004
```

---

---

```
Floor1Sleepingroom : 0000000000000008
Floor1Corridor     : 0000000000000010
Floor1Child1       : 0000000000000020
Floor1Child2       : 0000000000000040
...
```

---

Die beiden Arrays können Sie selbstverständlich auch für den Einsatz in Steuerelementen, z.B. ListBoxen oder ComboBoxen verwenden. Der Code

---

```
LBRooms.Items.AddRange(names)
```

---

füllt die in Abbildung 4 gezeigte ListBox.



Abbildung 4: Enum-Bezeichner in einer ListBox

## 27 Prüfen, ob ein Zahlenwert als Konstante in einer Enumeration definiert ist

Wenn Sie einen Zahlenwert vorliegen haben und wissen möchten, ob diesem in einer bestimmten Enumeration eine Konstante zugewiesen ist, dann können Sie hierzu die Methode `Enum.IsDefined` aufrufen. Für die Enumeration `Rooms` aus Listing 21 ergibt

---

```
Debug.WriteLine([Enum].IsDefined(GetType(Rooms), 1L))
Debug.WriteLine([Enum].IsDefined(GetType(Rooms), 3L))
```

---

im ersten Fall `True` (`Floor1Livingroom`) und im zweiten Fall `False` (keine Konstante vorhanden). Der Typ des übergebenen Zahlenwertes muss zum Typ der Enumeration passen – daher die Literale vom Typ `Long` (1L, 3L).

## 28 Prüfen, ob ein bestimmter Enumerationswert in einer Kombination von Werten vorkommt

Die Festlegung der Enumerationswerte auf Zweierpotenzen, wie sie in Listing 21 vorgenommen wurde, erlaubt eine beliebige Oder-Verknüpfung der Werte. Schon mehrfach wurden Definitionen wie

---

```
Dim r As Rooms = Rooms.Entrancehall Or Rooms.Floor1Bath
```

---

angesprochen. Da es sich letztlich um Zahlenwerte handelt, können Sie auch alle Bit-Operationen auf Enum-Werte anwenden. Wenn Sie also beispielsweise überprüfen wollen, ob in der Variablen `r` das Bit für `Rooms.Entrancehall` gesetzt ist, dann genügt eine simple And-Verknüpfung:

---

```
If (r And Rooms.Floor1Child1) <> 0 Then ...
```

---

Verwechseln Sie bitte nicht diese Art der Bit-Operation mit dem Aufruf von `Enum.IsDefined` aus dem vorigen Beispiel. Beide haben nichts miteinander zu tun.

## 29 Objekte eigener Klassen vergleichbar und sortierbar machen

Im Framework finden Sie viele Klassen und Methoden, die Aktionen wie Sortieren und Suchen beliebiger Objekte mit Hilfe allgemeiner Object-Referenzen unterstützen. Da eine Methode wie `Array.Sort` die Objekte, die sie sortieren soll, bzw. die zugrunde liegenden Klassen nicht kennen kann, ist sie für den Vergleich zweier Objekte auf fremde Hilfe angewiesen. Zur Bereitstellung einer solchen Hilfe gibt es grundsätzlich zwei Varianten:

1. Ein zusätzliches Vergleichs-Objekt führt den Vergleich zweier Objekte durch. Dieses Objekt implementiert die Schnittstelle `IComparer`. Methoden wie `Sort` beauftragen den Vergleichs-Objekt mit der Durchführung und verwenden das Ergebnis für den weiteren Ablauf des Algorithmus. Ein Beispiel hierzu finden Sie in Rezept [7.15](#) ((`ListView` des Explorers-Steuerelementes sortieren)).
2. Die Objekte selbst können sich mit anderen Objekten vergleichen. Hierzu implementiert die betreffende Klasse die Schnittstelle `IComparable`. In der einzigen Methode dieser Schnittstelle (`CompareTo`) wird der Vergleich ausprogrammiert.

Der Ablauf einer Sortierung mit der 2. Variante erfolgt etwa nach folgendem Muster:

Ganz gleich, auf welchem Algorithmus die Sortierung basiert, es müssen immer zwei Objekte miteinander verglichen werden. Das Ergebnis dieses Vergleichs beeinflusst den weiteren Ablauf und somit die Sortierung. Für die Sortieralgorithmen ist es völlig unerheblich, welche Art von Objekten sortiert werden sollen und welchen Inhalt diese Objekte haben. Einzig von Interesse ist, ob ein Objekt A größer, kleiner oder gleich einem Objekt B ist.

Um das zu ermitteln, werden zwei Objekte in der Methode `Sort` verglichen. Das geschieht intern über das `IComparable`-Interface in etwa folgendermaßen:

---

```
Dim o1 As IComparable = Objekt1 aus Liste
Dim o2 As Object = Objekt2 aus Liste
Select Case o1.CompareTo(o2)
...

```

---

Das Ergebnis des Vergleichs ist ein Integerwert mit folgenden Werten:

- ▶ `< 0`, wenn gilt: `o1 < o2`
- ▶ `= 0`, wenn gilt: `o1 = o2`
- ▶ `> 0`, wenn gilt: `o1 > o2`

Der absolute Zahlenwert spielt dabei keine Rolle. Sie werden auch selten in die Lage kommen, außer der Null für die Identität selbst Zahlenwerte vorzugeben. Denn in der Regel werden die Vergleiche an die zu vergleichenden Datentypen delegiert.

Listing 23 zeigt die Klasse `Animal` als Beispiel für die Implementierung der Schnittstelle `IComparable`. Zwei Member-Variablen, `Family` und `Name`, werden für den Vergleich herangezogen. Zunächst sollen die Member-Variablen `Family` miteinander verglichen werden. Wenn diese identisch sind, entscheidet der Vergleich der Member-Variablen `Name` über das Ergebnis.

Der Vergleich ist in der Schnittstellen-Methode `CompareTo` formuliert. Zwei Hilfsvariablen, `animal1` und `animal2`, dienen nur zur Veranschaulichung, welche Objekte miteinander verglichen werden. Zum Vergleich der `Family`-Felder wird die Aufgabe an die `CompareTo`-Methode der Klasse `String` weitergegeben. Auch diese implementiert die Schnittstelle `IComparable` und verfügt somit über die benötigte Methode.

Das Ergebnis des ersten Vergleichs wird in der Variablen `cmp` zwischengespeichert. Ist `cmp` gleich Null, dann sind die Namen der Tierfamilien identisch. In diesem Fall wird der Vergleich der `Name`-Felder durchgeführt. Wieder wird die Durchführung an die `CompareTo`-Methode der Klasse `String` delegiert. Das Ergebnis wird als Funktionswert zurückgegeben.

Wenn das Ergebnis des ersten Vergleichs nicht Null ist, dann unterscheiden sich die Tierfamilien. In diesem Fall müssen die `Name`-Felder nicht mehr verglichen werden, sondern der Wert von `cmp` kann als Endergebnis zurückgegeben werden.

```
Public Class Animal
    Implements IComparable

    Public ReadOnly Family As String
    Public ReadOnly Name As String

    Public Sub New(ByVal family As String, ByVal name As String)
        Me.Family = family
        Me.Name = name
    End Sub

    Public Function CompareTo(ByVal obj As Object) As Integer _
        Implements System.IComparable.CompareTo

        ' Sicherheitsabfrage
        If Not TypeOf obj Is Animal Then Throw New ArgumentException( _
            "Objekte vom Typ Animal können nur mit anderen Objekten " & _
            "vom Typ Animal verglichen werden")
        ' Die beiden zu vergleichenden Tierarten
        Dim animal1 As Animal = Me
        Dim animal2 As Animal = DirectCast(obj, Animal)

        ' Vergleichen der Familie
        Dim cmp As Integer = animal1.Family.CompareTo(animal2.Family)

        ' Sind die Familien identisch?
        If cmp = 0 Then
```

Listing 23: Die Klasse `Animal` implementiert die Schnittstelle `IComparable` und ermöglicht so den Vergleich von Objekten dieser Klasse untereinander

---

```

        ' Ja, dann Namen vergleichen und Ergebnis zurückgeben
        Return animal1.Name.CompareTo(animal2.Name)
    Else
        ' Nein, dann steht das Ergebnis bereits fest
        Return cmp
    End If

End Function

Public Overrides Function ToString() As String
    Return Me.Family.PadRight(10) & " (" & Me.Name & ")"
End Function

```

End Class

---

*Listing 23: Die Klasse Animal implementiert die Schnittstelle IComparable und ermöglicht so den Vergleich von Objekten dieser Klasse untereinander (Forts.)*

Praktischerweise implementieren alle Basistypen des Frameworks wie String, Double, Integer usw. die Schnittstelle IComparable. So können Sie stets den Vergleich auf deren CompareTo-Implementierungen abwälzen und brauchen sich keine Gedanken über die zurückzugebenden Zahlenwerte zu machen.

Ein kleines Beispiel mit der Klasse Animal zeigt, wie ein Array sortiert werden kann.

Definition des Arrays:

---

```

Dim Animals() As Animal = {New Animal("Bear", "Grizzly"), _
                             New Animal("Cat", "Tiger"), _
                             New Animal("Cat", "Lion"), _
                             New Animal("Bear", "Barribal"), _
                             New Animal("Bear", "Polar Bear"), _
                             New Animal("Whale", "Orca"), _
                             New Animal("Bear", "Raccoon"), _
                             New Animal("Cat", "Panther")}

```

---

Eine Ausgabe mit folgender Schleife

---

```

For Each creature As Animal In Animals
    Debug.WriteLine(creature)
Next

```

---

erzeugt vor der Sortierung folgende Ausgabe:

---

```

Bear      (Grizzly)
Cat        (Tiger)
Cat        (Lion)
Bear      (Barribal)
Bear      (Polar Bear)
Whale     (Orca)
Bear      (Raccoon)
Cat        (Panther)

```

---

Nach der Sortierung wird daraus:

|       |              |
|-------|--------------|
| Bear  | (Barribal)   |
| Bear  | (Grizzly)    |
| Bear  | (Polar Bear) |
| Bear  | (Raccoon)    |
| Cat   | (Lion)       |
| Cat   | (Panther)    |
| Cat   | (Tiger)      |
| Whale | (Orca)       |

Wenn Sie einmal nachvollziehen möchten, wann und in welcher Reihenfolge welches Objekt mit welchem anderen verglichen wird, ergänzen Sie die `CompareTo`-Methode um folgende Ausgabe:

```
Debug.WriteLine(animal1.ToString() & " verglichen mit " & _
    animal2.ToString())
```

Sie erhalten dann Einblick in den Sortieralgorithmus, in etwa in der Form:

|      |                                 |              |
|------|---------------------------------|--------------|
| Bear | (Grizzly) verglichen mit Bear   | (Barribal)   |
| Bear | (Barribal) verglichen mit Cat   | (Panther)    |
| Bear | (Barribal) verglichen mit Bear  | (Raccoon)    |
| Bear | (Barribal) verglichen mit Whale | (Orca)       |
| Bear | (Barribal) verglichen mit Bear  | (Polar Bear) |
| Cat  | (Tiger) verglichen mit Bear     | (Barribal)   |
| Bear | (Barribal) verglichen mit Cat   | (Lion)       |
| ...  |                                 |              |

Welche Vergleichsvariante (`IComparer` oder `IComparable`) Sie verwenden, hängt von der Art der Objekte und den Vergleichskriterien ab. Wenn vorab entschieden werden kann, nach welchen Kriterien zwei Objekte miteinander verglichen werden können, dann ist `IComparable` die einfachere Lösung. Können sich die Kriterien zur Laufzeit ändern, dann ist die `IComparer`-Variante die einzig mögliche.

### 30 Binäre Suche in Arrays und Auflistungen

Arrays und Auflistungen wie `ArrayList` implementieren mit der Methode `BinarySearch` die binäre Suche nach beliebigen Objekten. Hierzu wird eine sortierte Liste fortlaufend halbiert, bis das gewünschte Element gefunden worden ist. Der Algorithmus ist schnell und effektiv. Er setzt, wie die Methode `Sort`, voraus, dass zwei Objekte miteinander verglichen werden können. In diesem Fall wird das vorgegebene gesuchte Objekt mit anderen Objekten der Liste verglichen. Stellt der Vergleich die Identität der Objekte fest, dann ist die Suche erfolgreich abgeschlossen.

Auch hier kommen wieder die beiden Schnittstellen `IComparer` und `IComparable` ins Spiel (siehe vorhergehendes Rezept), für die es unterschiedliche Überladungen der Methode `BinarySearch` gibt. Ein kleines Beispiel mit der Klasse `Animal` aus dem vorigen Rezept verdeutlicht die Suche mit Hilfe der Schnittstellenimplementierung `IComparable`:

|                           |
|---------------------------|
| <b>Basics</b>             |
| Datum/<br>Zeit            |
| Anwen-<br>dungen          |
| Zeich-<br>nen             |
| Bild-<br>bearbei-<br>tung |
| Win-<br>dows<br>Forms     |
| Controls                  |
| Proper-<br>tyGrid         |
| Datei-<br>system          |
| Netz-<br>werk             |
| System                    |
| Daten-<br>banken          |
| XML                       |
| Wissen-<br>schaft         |
| Ver-<br>schiede-<br>nes   |

Zunächst wird das Array sortiert:

---

```
' Sortieren  
Array.Sort(Animals)
```

---

Dann wird ein Objekt erstellt, das die Werte enthält, nach denen gesucht werden soll:

---

```
' Danach wird gesucht:  
Dim animalToSearch As Animal = _  
    New Animal(TBFamily.Text, TBName.Text)
```

---

Nun kann gesucht werden:

---

```
' Suchen  
Dim index As Integer = Array.BinarySearch(Animals, animalToSearch)
```

---

Wird das Objekt gefunden, ist das Ergebnis der Index im Array. Wird es nicht gefunden, wird eine negative Zahl zurückgegeben.

---

```
If index < 0 Then  
    MessageBox.Show("Tierart nicht gefunden. Rückgabewert: " & _  
        index.ToString("X"))  
Else  
    MessageBox.Show("Tierart gefunden an Position " & index)  
End If
```

---

Ein Beispiel für das Suchergebnis zeigt Abbildung 5.

Die Suche erfolgt durch den wiederholten Aufruf von `CompareTo` und nicht durch den Vergleich der Referenzen. Wäre letzteres der Fall, dann würde das Beispielprogramm nie einen Eintrag finden können, da das Such-Objekt immer eine andere Referenz hat als die Objekte, auf die das Array verweist.

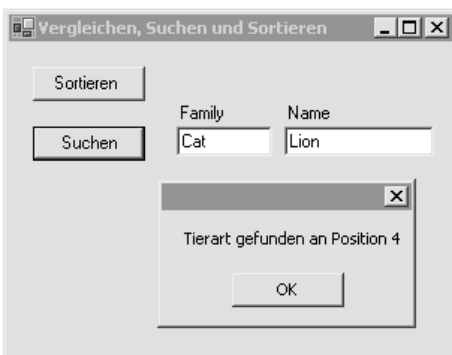


Abbildung 5: Ergebnis einer Suche mit `BinarySearch`



Hinweis

Beachten Sie bei der Verwendung von `BinarySearch` bitte unbedingt, dass die Liste vorher sortiert werden muss. Der Algorithmus der binären Suche funktioniert nur mit sortierten Auflistungen. Ist die Auflistung nicht sortiert, kann es vorkommen, dass Objekte nicht gefunden werden, obwohl sie in der Liste enthalten sind.

Die vorherige Sortierung dient also nicht der Performance-Verbesserung, wie in manchen Büchern zu lesen ist, sondern ist zwingend für das Funktionieren des Algorithmus erforderlich.

### 31 Strings in Byte-Arrays konvertieren und vice versa

Vorbei sind die Zeiten, da ein Byte genau ein ASCII-Zeichen repräsentierte. Verschiedene länder-spezifische Kodierungen und Unicode-Darstellungen lassen eine 1:1-Umsetzung nicht mehr zu. Intern arbeitet .NET immer mit Unicode-Zeichen, d.h. jedes Zeichen belegt zwei Byte. Externe Daten liegen aber oft in anderen Formaten vor und müssen entsprechend konvertiert werden. Auf unterster Ebene lassen sich diese Konvertierungen über die jeweiligen Encoder vornehmen. So können Sie einen String in einer bestimmten Kodierung in ein Byte-Array umwandeln und ein Byte-Array, das einen String repräsentiert, wieder in ein String-Objekt überführen.

Nicht alle Kodierungen unterstützen alle Zeichen. Zeichen, die z.B. der ASCII-Encoder nicht verarbeiten kann, werden durch Fragezeichen ersetzt. Sicher kennen Sie dieses leidige Problem vom täglichen Umgang mit Textdokumenten in eMails etc.

Die Konvertierungsmethoden sind schnell erklärt. Jeder Encoder unterstützt die Methode `GetBytes`. Diese liefert ein Byte-Array zurück, das einen als Parameter übergebenen Text in der Kodierung des Encoders repräsentiert:

```
Dim bArr() As Byte
bArr = System.Text.Encoding.ASCII.GetBytes("Guten Tag")
```

Für die Gegenrichtung stellt ein Encoder die Methode `GetString` zur Verfügung, die aus einem übergebenen Byte-Array wieder einen .NET-String erzeugt:

```
Dim text As String = System.Text.Encoding.ASCII.GetString(bArr)
```



Abbildung 6: Kodierung eines Zeichens in ein Byte-Array und zurück

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Propert-  
yGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

Ein kleines Beispielprogramm (Abbildung 6) ruft diese beiden Methoden für die gängigen Encoder auf. In der oberen TextBox kann ein beliebiges Zeichen eingegeben werden. Für dieses Zeichen wird die Repräsentation als Byte-Array für alle Kodierungen angezeigt (mittlere TextBox-Spalte). In der rechten TextBox-Spalte wird aus dem jeweiligen Byte-Array wieder das Zeichen reproduziert. So können Sie direkt erkennen, mit welchen Encodern das Zeichen darstellbar ist. Listing 24 zeigt die Implementierung unter Verwendung verschiedener Encoder.

---

```
Private Sub TBChar_TextChanged(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles TBChar.TextChanged

    ' Zeichen in TextBox selektieren
    TBChar.SelectionStart = 0
    TBChar.SelectionLength = 1

    ' Zeichen lesen
    Dim t As String = TBChar.Text

    Dim bArr() As Byte

    ' Kodierung in ASCII
    bArr = System.Text.Encoding.ASCII.GetBytes(t)
    TBAscii.Text = Format(bArr)
    TB1C.Text = System.Text.Encoding.ASCII.GetString(bArr)

    ' Kodierung in Big Endian Unicode
    bArr = System.Text.Encoding.BigEndianUnicode.GetBytes(t)
    TBBigEndian.Text = Format(bArr)
    TB2C.Text = System.Text.Encoding.BigEndianUnicode.GetString(bArr)

    ' Kodierung mit Standard-Encoder, hier iso-8859-1 (Windows-1252)
    bArr = System.Text.Encoding.Default.GetBytes(t)
    TBDefault.Text = Format(bArr)
    TB3C.Text = System.Text.Encoding.Default.GetString(bArr)

    ' Kodierung in 16-Bit Unicode
    bArr = System.Text.Encoding.Unicode.GetBytes(t)
    TBUnicode.Text = Format(bArr)
    TB4C.Text = System.Text.Encoding.Unicode.GetString(bArr)

    ' Kodierung in 7-Bit Unicode
    bArr = System.Text.Encoding.UTF7.GetBytes(t)
    TBUTF7.Text = Format(bArr)
    TB5C.Text = System.Text.Encoding.UTF7.GetString(bArr)

    ' Kodierung in 8-Bit Unicode
    bArr = System.Text.Encoding.UTF8.GetBytes(t)
    TBUTF8.Text = Format(bArr)
    TB6C.Text = System.Text.Encoding.UTF8.GetString(bArr)

End Sub
```

---

Listing 24: Verwendung verschiedener Encoder für die Konvertierung zwischen String und Byte-Array

## 32 Ersatz für unveränderliche (konstante) Zeichenketten-Arrays

Vielleicht haben Sie schon einmal versucht, ein `String`-Array so zu definieren, dass es zur Laufzeit nicht mehr verändert werden kann. Dieses trivial klingende Problem lässt sich leider nicht so einfach mit einem Schlüsselwort lösen. `Const` ist im Zusammenhang mit Arrays nicht erlaubt und `ReadOnly` hilft auch nicht richtig weiter:

---

```
Public ReadOnly NonConstantStringArray() As String _
    = {"Red", "Green", "Yellow", "Blue"}
```

---

Eine solche Definition schützt zwar die Referenz-Variable `NonConstantStringArray` vor späteren Zuweisungen, aber das Array, auf das sie verweist, ist keineswegs geschützt:

---

```
Dim color As String

color = NonConstantStringArray(0)
Debug.WriteLine("Vorher: " & color)

NonConstantStringArray(0) = "Gangster-Black"

color = NonConstantStringArray(0)
Debug.WriteLine("Nachher: " & color)
```

---

Die Zuweisung der Gangster-Farbe ist syntaktisch zulässig, so dass der Code folgende Ausgabe liefert:

---

```
Vorher: Red
Nachher: Gangster-Black
```

---

Einen wirklichen Schutz kann man nur erreichen, wenn auch die Zugriffe auf die Array-Elemente auf Lese-Operationen beschränkt werden können. Die Syntax von VB.NET sieht hierfür jedoch nichts vor. Stattdessen kann man sich behelfen, indem man eine Klasse definiert, die den Zugriff auf ein Array nur über eine `ReadOnly`-Eigenschaft gestattet. Listing 25 zeigt eine mögliche Implementierung hierfür.

---

```
Public Class ConstStringArray

    ' Geschützte Referenz des String-Arrays
    Protected Strings() As String

    Public Sub New(ByVal strings() As String)
        ' Wichtig! Array muss geklont werden
        Me.Strings = DirectCast(strings.Clone(), String())
    End Sub

    ' Die Standard-Eigenschaft erlaubt den Zugriff über den Index
```

---

Listing 25: Work-Around zur Realisierung konstanter String-Arrays

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

```

Default Public ReadOnly Property Item(ByVal index As Integer) _
    As String
    Get
        Return Strings(index)
    End Get
End Property

```

```
End Class
```

---

*Listing 25: Work-Around zur Realisierung konstanter String-Arrays (Forts.)*

In der Klasse wird eine geschützte Referenz-Variable definiert (`Strings`), die als einzige auf das zugrunde liegende `String`-Array verweist. Um sicherzustellen, dass außerhalb der Klasse keine Referenzen existieren, wird das `String`-Array im Konstruktor angenommen und dupliziert (geklont).

Der Zugriff auf das Array erfolgt über die schreibgeschützte Standard-Eigenschaft `Item`. Der Qualifizierer `ReadOnly` sorgt dafür, dass die Eigenschaft wirklich nur gelesen werden kann, während `Default` den Aufruf der Eigenschaft ohne Nennung des Namens ermöglicht. Dadurch sieht ein Aufruf von `Item` syntaktisch genauso aus, als wäre es eine Array-Indizierung:

---

```

Public ReadOnly ReallyConstantStringArray As New ConstStringArray _
    (New String() {"Red", "Green", "Yellow", "Blue"})

...

color = ReallyConstantStringArray(0)

```

---

Eine Zuweisung wie

---

```
ReallyConstantStringArray(0) = "Gangster-Black"
```

---

ist jetzt nicht mehr zulässig und wird vom Compiler abgelehnt.

Die Klasse `ConstStringArray` lässt sich so für beliebig viele invariante `String`-Arrays verwenden. Der einzige Mehraufwand besteht in der Deklarationszeile, da dort im Konstruktor-Aufruf ein Array mit `Strings` übergeben werden muss.

Jeder, der sich schon einmal intensiver mit Berechnungen von Uhrzeiten oder Kalendertagen beschäftigt hat, kennt die vielen Problematiken und Fallen, die dieses Thema mit sich bringt. Einige Probleme sind astronomischer Natur. So gibt es beispielsweise kein ganzzahliges Verhältnis zwischen der Zeit, die die Erde benötigt, um einmal um die Sonne zu kreisen, und der Zeit, die die Erde für eine Umdrehung um sich selbst benötigt. Bis dieser Missstand mit einem (intergalaktischen) Service Release behoben wird, müssen wir uns mit Annäherungen durch Schaltjahre begnügen.

Viele Besonderheiten hat die Menschheit aber selbst erfunden. Als Konsequenz aus den Beobachtungen des Sonnenstandes haben sich die Einheiten Tage und Jahre ergeben. Die Einteilung in Wochen und Monate hat zwar auch einen astronomischen Hintergrund, erscheint bei oberflächlicher Betrachtung aber eher willkürlich. So manches geht auch sprichwörtlich nach dem Mond. Eine ganze Reihe von Feiertagen richten sich nach dem ersten Vollmond im Frühling.

Und dann wären da noch die Zeitzonen, die sich (zumindest theoretisch) nach den Längengraden bemessen. Extrem problematisch kann auch der Umgang mit der Sommerzeit werden. Insbesondere die Übergänge zu Beginn und am Ende der Sommerzeit sind typische Fehlerquellen.

Dieses Kapitel stellt Ihnen einige Rezepte für den richtigen Umgang mit Kalendern, Datum und Uhrzeit bereit. Basis der Berechnungen ist der Gregorianische Kalender, der für unsere Kultur maßgeblich ist. Im Framework finden Sie aber auch Klassen für den Umgang mit Kalendern anderer Kulturen, z.B. `HebrewCalendar`, `JapaneseCalendar`, `TaiwanCalendar` usw., die hier aber nicht behandelt werden sollen.

## 33 Umgang mit Datum und Uhrzeit

Drei Datentypen werden für den Umgang mit Datums- und Zeitinformationen vom Framework bereitgestellt (Tabelle 4). Die meisten Berechnungen erfolgen mit Hilfe der Strukturen `DateTime` und `TimeSpan`. Der alte VB-Datentyp `Date` ist in VB.NET ein Synonym für `System.DateTime`.

| Name                           | Typ      | Bedeutung                 |
|--------------------------------|----------|---------------------------|
| <code>DateTime</code>          | Struktur | Absolutes Datum           |
| <code>TimeSpan</code>          | Struktur | Zeitspanne, relative Zeit |
| <code>GregorianCalendar</code> | Klasse   | Gregorianischer Kalender  |

Tabelle 4: Datentypen für den Umgang mit Zeiten und Kalenderdaten

Für einen intuitiven Umgang mit den beiden Strukturen wäre es praktisch, wenn einige Operatoren überladen werden könnten, was die VB-Syntax aber ja leider nicht hergibt. Statt dessen muss man sich in VB mit dem Aufruf von Funktionen zufrieden geben. So lassen sich leicht die in Tabelle 5 dargestellten typischen Aufgaben lösen.

Zahlreiche Überladungen der Konstruktoren und zusätzlicher statischer Methoden stehen Ihnen zur Definition von `DateTime` (Tabelle 6)-und `TimeSpan` (Tabelle 7)-Objekten zur Verfügung.

| Aufgabe                              | Ergebnistyp | Lösung                                                                              |
|--------------------------------------|-------------|-------------------------------------------------------------------------------------|
| Zeit jetzt                           | DateTime    | DateTime.Now                                                                        |
| Tag heute                            | DateTime    | DateTime.Today                                                                      |
| Zeit zwischen dem 1.1.2003 und heute | TimeSpan    | New DateTime(2003, 1, 1).Subtract(DateTime.Now)                                     |
| Gestern um die gleiche Zeit          | DateTime    | DateTime.Now.AddDays(-1)                                                            |
| Gestern 0:00 Uhr                     | DateTime    | DateTime.Now.AddDays(-1).Date                                                       |
| Zeitspanne seit Mitternacht          | TimeSpan    | DateTime.Now.TimeOfDay                                                              |
| Zeitspanne ts in Tagen, Stunden usw. | Double      | ts.TotalDays, ts.TotalHours, ts.TotalMinutes, ts.TotalSeconds, ts.TotalMilliseconds |
| Aktuelle Standardzeit                | DateTime    | DateTime.UtcNow                                                                     |
| Aktuelles Jahr                       | Integer     | DateTime.Now.Year                                                                   |

Tabelle 5: Eine Auswahl typischer Berechnungen zu Datum und Uhrzeit

| Methode                       | Bedeutung                                    |
|-------------------------------|----------------------------------------------|
| Konstruktor                   | Sieben Überladungen mit Integer-Parametern   |
| FromFileTime, FromFileTimeUtc | Umrechnung aus Zeitformaten des Dateisystems |
| FromFileTime                  | Umrechnung für OLE-Automatisierung           |
| Parse, ParseExact             | Konvertierung aus einer Zeichenkette         |

Tabelle 6: Methoden zur Konstruktion von DateTime-Werten

| Methode                  | Bedeutung                                |
|--------------------------|------------------------------------------|
| Konstruktor              | Vier Überladungen mit Integer-Parametern |
| FromDays, FromHours etc. | Zeitspanne aus Tagen, Stunden usw.       |
| Parse                    | Konvertierung aus einer Zeichenkette     |

Tabelle 7: Methoden zur Konstruktion von TimeSpan-Werten

Für Vergleiche stehen die üblichen Operatoren zur Verfügung, die Sie in VB.NET leider nur indirekt nutzen können (s. Tabelle 8). Alternativ können Sie die Methoden `Compare` und `CompareTo` einsetzen. Ein Vergleich zweier `DateTime`-Strukturen `dt1` und `dt2` sieht dann so aus:

statt

---

```
dt1 <= dt2
```

---

verwenden Sie in VB.NET

---

```
DateTime.op_LessThanOrEqual(dt1, dt2)
```

---

oder

```
dt1.CompareTo(dt2) <= 0
```

| Methode               | Operator               |
|-----------------------|------------------------|
| op_Equality           | =                      |
| op_GreaterThan        | >                      |
| op_GreaterThanOrEqual | >=                     |
| op_Inequality         | <>                     |
| op_LessThan           | <                      |
| op_LessThanOrEqual    | <=                     |
| op_UnaryNegation      | unäres- (nur TimeSpan) |
| op_Addition           | +                      |
| op_Subtraction        | -                      |

Tabelle 8: Diese Methoden stehen hinter den Operatoren von DateTime und TimeSpan

Zur Ausgabe der vielen verschiedenen internationalen Datums- und Zeitformate stellt die Struktur DateTime eine Reihe von Methoden bereit. Einige dieser Methoden basieren auf den Formaten der im Betriebssystem eingestellten länderspezifischen Definitionen, andere sind frei konfigurierbar.

| Methode                             | Bedeutung                            |
|-------------------------------------|--------------------------------------|
| ToString                            | Vier Überladungen, beliebige Formate |
| ToLongDateString, ToShortDateString | Länderspezifische Datumsformate      |
| ToLongTimeString, ToShortTimeString | Länderspezifische Zeitformate        |

Tabelle 9: Formatierung von DateTime-Werten für die Ausgabe

Wesentlich seltener wird die Klasse GregorianCalendar (siehe auch nächstes Rezept) benutzt. Sie bietet einige kalenderspezifische Umrechnungsmethoden, etwa für die Umrechnung einer zweistelligen Jahreszahl in eine vierstellig.

### 34 Schaltjahre

Schaltjahre sind definitionsgemäß die Jahre, deren Jahreszahl sich ohne Rest durch 4, aber nicht durch 100 teilen lässt, sowie die Jahre, deren Jahreszahl durch 400 teilbar ist.

Wollen Sie ermitteln, ob ein bestimmtes Jahr ein Schaltjahr ist, dann können Sie hierfür auf die Methode IsLeapYear der Klasse GregorianCalendar zurückgreifen. Sie liefert für eine angegebene Jahreszahl das gewünschte boolesche Ergebnis.

Zwei zusätzliche Funktionen, IsLeapMonth und IsLeapDay geben an, ob es sich um den Monat Februar innerhalb eines Schaltjahres bzw. um den 29. Februar handelt.

|                           |
|---------------------------|
| Basics                    |
| Datum/<br>Zeit            |
| Anwen-<br>dungen          |
| Zeich-<br>nen             |
| Bild-<br>bearbei-<br>tung |
| Win-<br>dows<br>Forms     |
| Controls                  |
| Proper-<br>tyGrid         |
| Datei-<br>system          |
| Netz-<br>werk             |
| System                    |
| Daten-<br>banken          |
| XML                       |
| Wissen-<br>schaft         |
| Ver-<br>schiede-<br>nes   |

## 35 Wochentag berechnen

Die Struktur `DateTime` kennt die Eigenschaft `DayOfWeek`, die die Berechnung des Wochentages durchführt. Leider ziemlich amerikanisch. Denn die Woche beginnt bei dieser Berechnung mit dem Sonntag (Tabelle 10). Listing 26 zeigt die Implementierung der Umrechnungsfunktion in die europäische Norm.

| Tag        | DayOfWeek | Norm |
|------------|-----------|------|
| Montag     | 1         | 1    |
| Dienstag   | 2         | 2    |
| Mittwoch   | 3         | 3    |
| Donnerstag | 4         | 4    |
| Freitag    | 5         | 5    |
| Samstag    | 6         | 6    |
| Sonntag    | 0         | 7    |

Tabelle 10: Nummerierung der Wochentage nach Microsoft und europäischer Norm

---

```
' Berechnung des Wochentages nach europäischer Norm
Public Shared Function GetDayOfWeek(ByVal [Date] As DateTime) _
    As Integer

    ' 1 = Montag ... 7 = Sonntag
    Return ([Date].DayOfWeek + 6) Mod 7 + 1

End Function
```

---

Listing 26: `GetDayOfWeek` in der europäischen Version

## 36 Beginn einer Kalenderwoche berechnen

Der Umgang mit Kalenderwochen ist ein wenig tückisch. Zwar gibt es in der Klasse `GregorianCalendar` die Methode `GetWeekOfYear`, die auch für europäische Kalenderwochen funktionieren sollte, jedoch hat Microsoft es bislang nicht geschafft, diese Methode korrekt zu implementieren (siehe Kasten). Im Hinweiskasten sehen Sie die Definition der Kalenderwoche, wie sie in Ihren Berechnungen berücksichtigt werden muss.

Achtung

Verwenden Sie nicht die Methode `GregorianCalendar.GetWeekOfYear`, um nach europäischer Norm Berechnungen zu Kalenderwochen anzustellen! Diese Methode ist fehlerhaft und liefert für Tage, die um den Jahreswechsel herum liegen, teilweise falsche Werte. Beispielsweise liegt der 30.12.2002 in der ersten Kalenderwoche des Jahres 2003 und nicht in der 53. Woche des Jahres 2002, wie die Funktion berechnet. Der Fehler ist schon seit längerem bei Microsoft bekannt und unter der Nummer Q200299 in der Knowledge Base dokumentiert, wurde bislang aber nicht behoben.



## Hinweis

Die Berechnung der europäischen Kalenderwoche ist genormt. International in der ISO 8601, europäisch in der EN 28601 und in Deutschland entsprechend in der DIN 1355. Danach beginnt eine Kalenderwoche immer mit einem Montag. Die erste Kalenderwoche ist die Woche, in der der vierte Januar liegt.

Ein Jahr kann somit 52 oder 53 Kalenderwochen haben. Die Tage vor dem 4.1. können in die letzte Kalenderwoche des Vorjahres fallen, die Tage nach dem 28.12. in die erste Kalenderwoche des Folgejahres.

Für die Berechnung des ersten Tages einer Kalenderwoche ist es somit notwendig, den Beginn der ersten Kalenderwoche zu berechnen. Ausgangspunkt ist der vierte Januar. Mit Hilfe der Eigenschaft `DayOfWeek` wird ermittelt, auf welchen Wochentag dieser fällt. Aus dieser Information wird dann der Montag der ersten Woche ermittelt.

Der Beginn der n. Woche ergibt sich dann aus der Addition von n-1 Wochen auf den Beginn der ersten Woche. Listing 27 zeigt die Implementierung der Funktion.

```
Public Shared Function GetStartOfCalendarWeek( _
    ByVal year As Integer, ByVal calendarWeek As Integer) _
    As DateTime

    ' 4. Januar des betreffenden Jahres
    Dim january4 As New DateTime(year, 1, 4)

    ' Nummer des Wochentags des 4. Januars 1=Mo..7=So
    Dim weekdayJan4 As Integer
    weekdayJan4 = GetDayOfWeek(january4)

    ' Dann beginnt die KW1 x Tage vorher
    Dim dateOfFirstWeek As DateTime
    dateOfFirstWeek = january4.AddDays(1 - weekdayJan4)

    ' Die gesuchte KW beginnt dann KW-1 Wochen später:
    Return dateOfFirstWeek.AddDays((calendarWeek - 1) * 7)

End Function
```

Listing 27: Berechnung des ersten Tages einer Kalenderwoche

## 37 Anzahl der Kalenderwochen eines Jahres bestimmen

Wie bereits erwähnt, kann ein Jahr 52 oder 53 Kalenderwochen umfassen. Die genaue Anzahl lässt sich aus der Differenz in Tagen zwischen dem Beginn der ersten Woche des Folgejahres und dem Beginn der ersten Woche des angegebenen Jahres berechnen (Listing 28).

```
Public Shared Function GetNumberOfCalendarWeeks( _
    ByVal year As Integer) As Integer

    ' 1. KW des Folgejahres - 1. KW des betrachteten Jahres
```

Listing 28: Wie viele Kalenderwochen hat ein Jahr?

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Propert-  
yGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

---

```

Dim diff As TimeSpan = _
    GetStartOfCalendarWeek(year + 1, 1).Subtract _
    (GetStartOfCalendarWeek(year, 1))

Return diff.Days \ 7

End Function

```

---

*Listing 28: Wie viele Kalenderwochen hat ein Jahr? (Forts.)*

## 38 Berechnung der Kalenderwoche zu einem vorgegebenen Datum

Mit Hilfe der vorangegangenen Rezepte lässt sich leicht und übersichtlich zu einem vorgegebenen Datum die Kalenderwoche ermitteln. Allerdings reicht die Wochennummer als alleiniger Rückgabewert nicht aus. Denn es kann ja sein, dass die berechnete Kalenderwoche zu einem anderen Jahr gehört und die Nummer unter Umständen nicht eindeutig ist. Als Ergebnis können Sie eine Kalenderwoche erhalten, die im gleichen Jahr liegt wie das angegebene Datum, die letzte Kalenderwoche des Vorjahres oder die erste Kalenderwoche des Folgejahres.

Um eine Unterscheidung treffen zu können, wird eine Struktur definiert, die sowohl die Wochennummer als auch das Jahr enthält (Listing 29). Die Überschreibung der Methode `ToString` ermöglicht die direkte Ausgabe der Werte in Textform.

---

```

Public Structure CalendarWeek

    Public ReadOnly Week, Year As Integer

    Public Sub New(ByVal week As Integer, ByVal year As Integer)
        Me.Week = week
        Me.Year = year
    End Sub

    Public Overrides Function ToString() As String
        Return String.Format("KW {0} {1}", Week, Year)
    End Function

End Structure

```

---

*Listing 29: Hilfsstruktur für den Umgang mit Kalenderwochen*

Zwei Stichtage sind somit relevant: Der Beginn der ersten Woche des betrachteten Jahres (`day1`, siehe Listing 30) und der Beginn der ersten Woche des Folgejahres (`day2`). Ist der Wert des betrachteten Tags (`dateOfInterest`) größer als `day2`, dann gehört der Tag zur ersten Woche des Folgejahres.

Ansonsten muss ermittelt werden, ob `dateOfInterest` kleiner ist als `day1`. Wenn ja, dann ist die gesuchte Woche die letzte Woche des Vorjahres (`GetNumberOfCalendarWeeks(year - 1)`). Wenn nein, dann errechnet sich die Nummer der Woche aus der Differenz zum Beginn der ersten Woche in Tagen (`dateOfInterest.Subtract(day1).Days \ 7 + 1`).

```

Public Shared Function GetCalendarWeek( _
    ByVal dateOfInterest As DateTime) As CalendarWeek

    ' Betrachtetes Jahr
    Dim year As Integer = dateOfInterest.Year

    ' Erster Tag der ersten Woche dieses Jahres
    Dim day1 As DateTime = GetStartOfCalendarWeek(year, 1)

    ' Erster Tag der ersten Woche des Folgejahres
    Dim day2 As DateTime = GetStartOfCalendarWeek(year + 1, 1)

    ' Gehört der Tag zur ersten Woche des Folgejahres?
    If DateTime.op_GreaterThanOrEqual(dateOfInterest, day2) Then
        ' Ja, KW 1 des Folgejahres zurückgeben
        Return New CalendarWeek(1, year + 1)
    Else
        ' Gehört der Tag zu einer Kalenderwoche des gleichen Jahres?
        If DateTime.op_GreaterThanOrEqual(dateOfInterest, day1) Then
            ' Ja, KW aus Differenz zum 1. Tag der ersten Woche berechnen
            Return New CalendarWeek( _
                dateOfInterest.Subtract(day1).Days \ 7 + 1, year)
        Else
            ' Nein, letzte KW des Vorjahres zurückgeben
            Return New CalendarWeek( _
                GetNumberOfCalenderWeeks(year - 1), year - 1)
        End If
    End If
End Function

```

Listing 30: Berechnung der Kalenderwoche zu einem vorgegebenen Datum

An dieser Stelle sei nochmals auf die Anmerkungen zu `GregorianCalendar.GetWeekOfYear` in Rezept 2.39 ((Beginn einer Kalenderwoche berechnen)) hingewiesen. Im Gegensatz zu dieser Funktion wird die Kalenderwoche vom `MonthCalendar`-Steuerelement richtig angezeigt. Es kann daher zur Überprüfung der Berechnungen herangezogen werden (Abbildung 7 / Listing 31).



Abbildung 7: Überprüfung der Berechnungen mit Hilfe des `MonthCalendar`-Steuerelementes

---

```

Private Sub MonthCalendar1_DateChanged( _
    ByVal sender As System.Object, ByVal e As _
    System.Windows.Forms.DateRangeEventArgs) _
    Handles MonthCalendar1.DateChanged

    LBLKW.Text = EuropeanCalendarCalculations.GetCalendarWeek( _
        MonthCalendar1.SelectionStart).ToString()

End Sub

```

---

*Listing 31: Ausgabe der berechneten Kalenderwoche zur Auswahl im Kalender-Control*

## 39 Berechnung des Osterdatums

Ausgangspunkt für die Berechnung vieler Feiertage ist der Ostersonntag. Von kirchlicher Seite wurde vor langer Zeit festgelegt, dass Ostern stets an dem Sonntag zu feiern sei, der unmittelbar dem Tag des ersten Vollmonds nach der Tag-und-Nacht-Gleiche im März (Frühjahrsbeginn) folgt bzw. auf ihm liegt. Was die Kirche seinerzeit nicht bedachte, war, dass die Tag-und-Nacht-Gleiche nicht immer auf den 21. März fällt, sondern durchaus ein oder zwei Tage früher eintreten kann. So ergaben sich unterschiedliche Berechnungsformeln, die auch für manche Jahre verschiedene Ergebnisse liefern.

Die heute allgemein gebräuchliche Formel ist die in Listing 32 abgedruckte von Jean Meeus (Journal of the British Astronomical Association, Band 88 (1977)). Sie berechnet das korrekte Osterdatum für den Gregorianischen Kalender, also ab dem Jahr 1583. Zur Funktionsweise der Formel, zu Risiken und Nebenwirkungen, befragen Sie bitte den genannten Autor ;-).

---

```

Public Shared Function GetEasterDate(ByVal year As Integer) _
    As DateTime

    ' Berechnung des Ostersonntags nach einer Formel aus dem Buch
    ' "Astronomical Formulae For Calculators" des Belgiers
    ' Jean Meeus, erschienen 1982 im Willmann-Bell-Verlag,
    ' Richmond, Virginia
    ' Gültig für den Gregorianischen Kalender (ab 1583)

    If year < 1 Then Throw New _
        ArgumentOutOfRangeException( _
            "Berechnung des Osterdatums nur ab dem Jahr 1 möglich")

    Dim a, b, c, d, e, f, g, h, i, k, l, m, n, p As Integer
    If year > 1582 Then

        ' Gregorianischer Kalender

        a = year Mod 19
        b = year \ 100
        c = year Mod 100
        d = b \ 4
        e = b Mod 4

```

---

*Listing 32: Berechnung des Osterdatums nach einer alten Näherungsformel*

```

f = (b + 8) \ 25
g = (b - f + 1) \ 3
h = (19 * a + b - d - g + 15) Mod 30
i = c \ 4
k = c Mod 4
l = (32 + 2 * e + 2 * i - h - k) Mod 7
m = (a + 11 * h + 22 * l) \ 451
n = (h + l - 7 * m + 114) \ 31
p = (h + l - 7 * m + 114) Mod 31

Return New DateTime(year, n, p + 1)

```

Else

*' Julianischer Kalender*

```

a = year Mod 4
b = year Mod 7
c = year Mod 19
d = (19 * c + 15) Mod 30
e = (2 * a + 4 * b - d + 34) Mod 7
f = (d + e + 114) \ 31
g = (d + e + 114) Mod 31
Return New Date(year, f, g + 1)

```

End If

End Function

*Listing 32: Berechnung des Osterdatums nach einer alten Näherungsformel (Forts.)*

## 40 Berechnung der deutschen Feiertage

Für den Umgang mit den deutschen Feiertagen stellen wir Ihnen eine Auflistungsklasse (Holiday-Collection) zur Verfügung, die folgende Funktionalitäten bietet:

- ▶ Berechnung der Feiertage eines Jahres und Speicherung in einer Hashtable-Liste
- ▶ Nachschlagen eines oder mehrerer Feiertage in der Liste
- ▶ Erzeugung eines sortierten Arrays aller Feiertage

Für den Umgang mit Feiertagen wird eine Hilfsklasse (Holiday) definiert, die für einen Feiertag dessen Bezeichnung und Datum enthält (s. Listing 33). Die Implementierung der Schnittstelle IComparable ermöglicht das Sortieren von Instanzen dieser Klasse nach Datum. Dadurch lässt sich beispielsweise ein Array von Holiday-Referenzen direkt mit Array.Sort sortieren.

```

Public Class Holiday
    Implements IComparable

    Public ReadOnly Name As String
    Public ReadOnly [Date] As DateTime

```

*Listing 33: Hilfsklasse Holiday speichert Name und Datum eines Feiertags*

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

```

Public Sub New(ByVal name As String, ByVal [Date] As DateTime)
    Me.Name = name
    Me.Date = [Date]
End Sub

Public Function CompareTo(ByVal obj As Object) As Integer _
    Implements System.IComparable.CompareTo

    If Not TypeOf obj Is Holiday Then Throw New ArgumentException( _
        "Ein Holiday-Objekt kann nur mit einem anderen Holiday-" _
        & "Objekt verglichen werden")

    ' Vergleich auf Basis des Datums
    Dim hd2 As Holiday = DirectCast(obj, Holiday)
    Return Me.Date.CompareTo(hd2.Date)

End Function

Public Overrides Function ToString() As String
    Return String.Format("{0} ({1})", Me.Date.ToLongDateString(), _
        Me.Name)
End Function

End Class

```

---

Listing 33: Hilfsklasse Holiday speichert Name und Datum eines Feiertags (Forts.)

In Listing 34 sehen Sie die Funktions-Header der implementierten Methoden und Eigenschaften. Eine Feiertagsliste ist immer einem eindeutigen Jahr zugeordnet, das über die Eigenschaft `Year` abgefragt werden kann. Die Hauptarbeit, nämlich die Berechnung der Feiertage, erfolgt im öffentlichen Konstruktor. Die Implementierung diverser Hilfsfunktionen und Eigenschaften wie `Item`, `Keys` etc. erfolgt nach dem üblichen Muster für das Ableiten einer Listenklasse von `DictionaryBase` und soll hier nicht weiter beschrieben werden. Sie finden den vollständigen Code auf der CD. Interessant sind hingegen die spezifischen Funktionen, die einen gezielten Abruf von Feiertagen als Array oder Liste ermöglichen.

---

```

Public Class HolidayCollection
    Inherits DictionaryBase

    ' Jahr, für das die Feiertage berechnet wurden
    Public ReadOnly Property Year() As Integer

    ' Konstruktoren
    Protected Sub New()
    Public Sub New(ByVal year As Integer)

    ' Implementierungen für DictionaryBase-Ableitung
    Default Public Property Item(ByVal key As String) As Holiday
    Public ReadOnly Property Keys() As ICollection

```

---

Listing 34: Eigenschaften und Methoden der Klasse HolidayCollection

---

```

Public ReadOnly Property Values() As ICollection
Public Sub Add(ByVal key As String, ByVal value As Holiday)
Public Function Contains(ByVal key As String) As Boolean
Public Sub Remove(ByVal key As String)

' Spezifische Kalenderfunktionen
Public Function ToArray() As Holiday()
Public Function GetHolidayCollection(ByVal holidays() As String) _
    As HolidayCollection
Public Function GetHolidayDates() As DateTime()
Public Function GetHolidayByDate(ByVal [Date] As DateTime) _
    As Holiday

End Class

```

---

Listing 34: Eigenschaften und Methoden der Klasse HolidayCollection (Forts.)

Zur Instanzierung der Liste muss dem öffentlichen Konstruktor ein Jahr übergeben werden. Für dieses Jahr werden die Feiertage berechnet. In Listing 35 sehen Sie den Rahmen des Konstruktors mit der Definition der Hilfsvariablen. Insbesondere das Osterdatum ist für die Berechnung einiger Feiertage von besonderer Bedeutung. Die einzelnen Berechnungen werden mit den nachfolgenden Listings beschrieben.

---

```

Public Sub New(ByVal year As Integer)

    Me.CalendarYear = year

    ' Feiertage für angegebenes Jahr bestimmen

    ' Hilfsvariablen

    Dim hd As Holiday

    ' Ostersonntag
    Dim easterday As DateTime = _
        EuropeanCalendarCalculations.GetEasterDate(year)

    Dim d As DateTime

    ' Beginn des Kirchenjahrs
    Dim beginnKirchenjahr As DateTime

    Dim wd As Integer
    Dim diff As Integer

    ' Berechnung der Feiertage siehe nachfolgende Listings

End Sub

```

---

Listing 35: Beginn der Feiertagsberechnung

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Propert-  
yGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

## Feste Feiertage

Feste Feiertage haben in jedem Jahr das gleiche Datum. Für jeden Feiertag wird eine Instanz von `Holiday` angelegt und der Auflistung hinzugefügt (siehe Listing 36). Das Datum errechnet sich aus dem vorgegebenen Tag sowie dem Jahr, für das die Liste erstellt werden soll.

Da es sich bei der internen Auflistung um eine Key-Value-Liste handelt, muss für jeden Eintrag ein Schlüssel angegeben werden. Als Schlüssel wird hier die Bezeichnung des Feiertags angegeben, so dass später über diese Bezeichnung das Datum des Feiertags ermittelt werden kann.

---

```
' *** Neujahr ***  
' 01. Januar  
  
hd = New Holiday("Neujahr", New DateTime(year, 1, 1))  
Dictionary.Add(hd.Name, hd)  
  
' *** Heilige 3 Könige ***  
' 06. Januar  
  
hd = New Holiday("Heilige 3 Könige", New DateTime(year, 1, 6))  
Dictionary.Add(hd.Name, hd)  
  
' *** Valentinstag ***  
' 14. Februar  
  
hd = New Holiday("Valentinstag", New DateTime(year, 2, 14))  
Dictionary.Add(hd.Name, hd)  
  
' *** Tag der Arbeit ***  
' 01. Mai  
  
hd = New Holiday("Tag der Arbeit", New DateTime(year, 5, 1))  
Dictionary.Add(hd.Name, hd)  
  
' *** Maria Himmelfahrt ***  
' 15. August  
  
hd = New Holiday("Maria Himmelfahrt", New DateTime(year, 8, 15))  
Dictionary.Add(hd.Name, hd)  
  
' *** Tag der Deutschen Einheit ***  
' 3. Oktober  
  
hd = New Holiday("Tag der Deutschen Einheit", _  
    New DateTime(year, 10, 3))  
Dictionary.Add(hd.Name, hd)  
  
' *** Reformationstag ***  
' 31. Oktober  
  
hd = New Holiday("Reformationstag", New DateTime(year, 10, 31))  
Dictionary.Add(hd.Name, hd)
```

---

Listing 36: Feste Feiertage



```

' *** Allerheiligen ***
' 1. November

hd = New Holiday("Allerheiligen", New DateTime(year, 11, 1))
Dictionary.Add(hd.Name, hd)

' *** Heiligabend ***
' 24. Dezember

hd = New Holiday("Heiligabend", New DateTime(year, 12, 24))
Dictionary.Add(hd.Name, hd)

' *** 1. Weihnachtsfeiertag ***
' 25. Dezember

hd = New Holiday("1. Weihnachtsfeiertag", _
    New DateTime(year, 12, 25))
Dictionary.Add(hd.Name, hd)

' *** 2. Weihnachtsfeiertag ***
' 26. Dezember

hd = New Holiday("2. Weihnachtsfeiertag", _
    New DateTime(year, 12, 26))
Dictionary.Add(hd.Name, hd)

' *** Silvester ***
' 31. Dezember

hd = New Holiday("Silvester", New DateTime(year, 12, 31))
Dictionary.Add(hd.Name, hd)

```

Listing 36: Feste Feiertage (Forts.)

## Bewegliche Feiertage

Die meisten unserer Feiertage haben kein festes Datum, sondern richten sich entweder nach Ostern oder nach dem Kirchenjahr. Die Berechnung des Osterdatums wurde bereits in Rezept [2.42](#) ((Berechnung des Osterdatums)) erläutert. In Listing 37 sehen Sie die Berechnung der Feiertage, die von Ostern abhängen. Das Datum ergibt sich daher durch Addition oder Subtraktion von Tagen auf das Datum des Ostersonntags. Auch der Muttertag, der regelmäßig auf den zweiten Sonntag im Mai fällt, ist von Ostern abhängig. Fällt der zweite Sonntag mit Pfingsten zusammen, ist der Muttertag eine Woche vorher.

```

' *** Rosenmontag ***
' 48 Tage vor Ostersonntag

hd = New Holiday("Rosenmontag", easterday.AddDays(-48))
Dictionary.Add(hd.Name, hd)

```

Listing 37: Feiertage, die vom Osterdatum abhängen

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

```
' *** Aschermittwoch ***  
' 46 Tage vor Ostersonntag  
  
hd = New Holiday("Aschermittwoch", easterday.AddDays(-46))  
Dictionary.Add(hd.Name, hd)  
  
' *** Karfreitag ***  
' 2 Tage vor Ostersonntag  
  
hd = New Holiday("Karfreitag", easterday.AddDays(-2))  
Dictionary.Add(hd.Name, hd)  
  
' *** Ostersonntag ***  
  
hd = New Holiday("Ostersonntag", easterday)  
Dictionary.Add(hd.Name, hd)  
  
' *** Ostermontag ***  
' 1 Tag nach Ostersonntag  
  
hd = New Holiday("Ostermontag", easterday.AddDays(1))  
Dictionary.Add(hd.Name, hd)  
  
' *** Weißer Sonntag ***  
' 7 Tage nach Ostersonntag  
  
hd = New Holiday("Weißer Sonntag", easterday.AddDays(7))  
Dictionary.Add(hd.Name, hd)  
  
' *** Christi Himmelfahrt ***  
' 39 Tage nach Ostersonntag  
  
hd = New Holiday("Christi Himmelfahrt", easterday.AddDays(39))  
Dictionary.Add(hd.Name, hd)  
  
' *** Pfingstsonntag ***  
' 49 Tage nach Ostersonntag  
  
Dim pfingstsonntag As DateTime = easterday.AddDays(49)  
hd = New Holiday("Pfingstsonntag", pfingstsonntag)  
Dictionary.Add(hd.Name, hd)  
  
' *** Pfingstmontag ***  
' 50 Tage nach Ostersonntag  
  
hd = New Holiday("Pfingstmontag", easterday.AddDays(50))  
Dictionary.Add(hd.Name, hd)  
  
' *** Muttertag ***  
' 2. Sonntag im Mai, es sei denn, dass dann Pfingsten ist  
' dann 1. Sonntag im Mai
```

---

Listing 37: Feiertage, die vom Osterdatum abhängen (Forts.)

```

' Wochentag des ersten Mai (0 = Montag)
wd = (New DateTime(year, 5, 1).DayOfWeek + 6) Mod 7

' Tage bis zum zweiten Sonntag
diff = 14 - wd

d = New DateTime(year, 5, diff)

' Bei Pfingstsonntag eine Woche abziehen
If d = pfingstsonntag Then d = d.AddDays(-7)

hd = New Holiday("Muttertag", d)
Dictionary.Add(hd.Name, hd)

' *** Fronleichnam ***
' 60 Tage nach Ostersonntag

hd = New Holiday("Fronleichnam", easterday.AddDays(60))
Dictionary.Add(hd.Name, hd)

```

Listing 37: Feiertage, die vom Osterdatum abhängen (Forts.)

Einige Feiertage im November und Dezember hängen vom Beginn des Kirchenjahres ab. Das Kirchenjahr beginnt am ersten Advent, dem vierten Sonntag vor dem Weihnachtstag (25.12.). Die in Listing 38 berechneten Feiertage liegen eine feste Anzahl von Tagen vor oder nach diesem Zeitpunkt.

```

' *** Beginn des Kirchenjahres ***
' 4. Sonntag vor dem 25. Dezember

d = New DateTime(year, 12, 25)
' Wochentag des 25. Dezember (0 = Montag)
wd = (d.DayOfWeek + 6) Mod 7

' Abstand zum vorausgegangenen Sonntag
diff = -wd - 1

beginnKirchenjahr = (d.AddDays(diff - 21))

' *** Buß- und Betttag ***
' Mittwoch vor dem letzten Sonntag des Kirchenjahres

hd = New Holiday("Buß- und Betttag", _
    beginnKirchenjahr.AddDays(-11))
Dictionary.Add(hd.Name, hd)

' *** 1. Advent ***
' Beginn des Kirchenjahres

```

Listing 38: Feiertage, die vom Beginn des Kirchenjahres abhängen

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Propor-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

```

hd = New Holiday("1. Advent", beginnKirchenjahr)
Dictionary.Add(hd.Name, hd)

' *** 2. Advent ***
' Beginn des Kirchenjahres + 1 Woche

hd = New Holiday("2. Advent", beginnKirchenjahr.AddDays(7))
Dictionary.Add(hd.Name, hd)

' *** 3. Advent ***
' Beginn des Kirchenjahres + 2 Wochen

hd = New Holiday("3. Advent", beginnKirchenjahr.AddDays(14))
Dictionary.Add(hd.Name, hd)

' *** 4. Advent ***
' Beginn des Kirchenjahres + 3 Wochen

hd = New Holiday("4. Advent", beginnKirchenjahr.AddDays(21))
Dictionary.Add(hd.Name, hd)

```

---

*Listing 38: Feiertage, die vom Beginn des Kirchenjahres abhängen (Forts.)*

Die dokumentierte Implementierung berücksichtigt nur die derzeit üblichen deutschen Feiertage, ohne Berücksichtigung, ob diese arbeitsfrei sind oder nicht. Sie können die Liste leicht für weitere Feiertage anderer Länder erweitern und die Klasse so auch für den internationalen Einsatz aufbereiten.

### Ausgabe aller Feiertage eines Jahres

Das Durchlaufen der Liste genügt, um alle Feiertage eines Jahres behandeln zu können. Sie können die Formatierung des Ausgabetextes der ToString-Überschreibung der Klasse Holiday überlassen:

---

```

Dim hdc As New HolidayCollection(2003)
For Each hd As Holiday In hdc.Values
    Debug.WriteLine(hd)
Next

```

---

erzeugt folgende (unsortierte) Ausgabe:

---

```

Sonntag, 20. April 2003 (Ostersonntag)
Montag, 9. Juni 2003 (Pfingstmontag)
Sonntag, 8. Juni 2003 (Pfingstsonntag)
Samstag, 1. November 2003 (Allerheiligen)
Mittwoch, 31. Dezember 2003 (Silvester)
Sonntag, 7. Dezember 2003 (2. Advent)
Mittwoch, 5. März 2003 (Aschermittwoch)
Freitag, 3. Oktober 2003 (Tag der Deutschen Einheit)
Freitag, 26. Dezember 2003 (2. Weihnachtsfeiertag)
Montag, 3. März 2003 (Rosenmontag)
Donnerstag, 25. Dezember 2003 (1. Weihnachtsfeiertag)

```

---

---

Mittwoch, 19. November 2003 (Buß- und Betttag)  
 Donnerstag, 1. Mai 2003 (Tag der Arbeit)  
 Sonntag, 21. Dezember 2003 (4. Advent)  
 Sonntag, 30. November 2003 (1. Advent)  
 Donnerstag, 29. Mai 2003 (Christi Himmelfahrt)  
 Freitag, 31. Oktober 2003 (Reformationstag)  
 Freitag, 18. April 2003 (Karfreitag)  
 Sonntag, 27. April 2003 (Weißer Sonntag)  
 Sonntag, 11. Mai 2003 (Muttertag)  
 Montag, 6. Januar 2003 (Heilige 3 Könige)  
 Mittwoch, 24. Dezember 2003 (Heiligabend)  
 Sonntag, 14. Dezember 2003 (3. Advent)  
 Donnerstag, 19. Juni 2003 (Fronleichnam)  
 Freitag, 15. August 2003 (Maria Himmelfahrt)  
 Freitag, 14. Februar 2003 (Valentinstag)  
 Mittwoch, 1. Januar 2003 (Neujahr)  
 Montag, 21. April 2003 (Ostermontag)

---

### Abfrage eines bestimmten Feiertags

Möchten Sie das Datum eines bestimmten Feiertags ermitteln, können Sie sich die Schlüsselliste zu Nutze machen. Über die Eigenschaft `Item` lässt sich gezielt das Datum des gesuchten Tages abfragen:

---

```

Dim hdc As New HolidayCollection(2003)
Dim hd As Holiday = hdc.Item("Ostermontag")
If hd Is Nothing Then
    Debug.WriteLine("Nicht vorhanden")
Else
    Debug.WriteLine(hd)
End If
  
```

---

ergibt die Ausgabe:

---

Montag, 21. April 2003 (Ostermontag)

---

Ein Nachteil im praktischen Einsatz könnte sein, dass die exakte Schreibweise der Feiertagsbezeichnung entscheidend für die Suche ist. Bereits ein kleiner Schreibfehler verhindert, dass der Schlüssel gefunden werden kann. Wenn Sie oft derartige Suchfunktionen benötigen, sollten Sie zusätzlich eine Enumeration aufbauen, die jedem Feiertag eine eindeutige Zahl zuordnet. Enumerationskonstanten können mit Hilfe der Auto-Vervollständigen-Funktionen des Visual Studios sicherer eingegeben werden als nicht überprüfbare Zeichenketten.

### Abfrage einer Liste von Feiertagen

Wollen Sie z.B. für ein bestimmtes Bundesland nur die Feiertage abfragen, die arbeitsfrei sind, dann können Sie hierzu die Methode `GetHolidayCollection` aufrufen und ein String-Array mit den Bezeichnungen der in Frage kommenden Feiertage übergeben. Für Nordrhein-Westfalen könnte der Aufruf so aussehen:

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

```

Dim hdcAll As New HolidayCollection(2003)
Dim wanted() As String = {"Neujahr", "Christi Himmelfahrt", _
    "Karfreitag", "Ostermontag", "Tag der Arbeit", _
    "Pfingstmontag", "Fronleichnam", "Tag der Deutschen Einheit", _
    "Allerheiligen", "1. Weihnachtsfeiertag", _
    "2. Weihnachtsfeiertag"}

Dim hdcNRW As HolidayCollection = _
    hdcAll.GetHolidayCollection(wanted)

For Each hd As Holiday In hdcNRW.Values
    Debug.WriteLine(hd)
Next

```

---

Der Rückgabewert der Methode `GetHolidayCollection` ist wiederum vom Typ `HolidayCollection`, die Liste beschränkt sich jedoch auf die gesuchten Feiertage. Für den obigen Aufruf wird folgendes Ergebnis ausgegeben:

---

```

Freitag, 3. Oktober 2003 (Tag der Deutschen Einheit)
Montag, 21. April 2003 (Ostermontag)
Donnerstag, 19. Juni 2003 (Fronleichnam)
Samstag, 1. November 2003 (Allerheiligen)
Donnerstag, 29. Mai 2003 (Christi Himmelfahrt)
Donnerstag, 1. Mai 2003 (Tag der Arbeit)
Freitag, 18. April 2003 (Karfreitag)
Montag, 9. Juni 2003 (Pfingstmontag)
Mittwoch, 1. Januar 2003 (Neujahr)
Freitag, 26. Dezember 2003 (2. Weihnachtsfeiertag)
Donnerstag, 25. Dezember 2003 (1. Weihnachtsfeiertag)

```

---

Die Implementierung der Methode zeigt Listing 39. Das übergebene Array mit den Bezeichnungen der gesuchten Feiertage wird durchlaufen und jeder gefundene Feiertag einer neu instanziierten Liste hinzugefügt.

---

```

Public Function GetHolidayCollection(ByVal holidays() As String) _
    As HolidayCollection

    ' Keine Suche, wenn die Liste leer ist
    If holidays.Length <= 0 Then Return Nothing

    ' Neue leere Auflistung erstellen
    Dim list As New HolidayCollection

    ' Jahr der Liste festlegen
    list.CalendarYear = Me.Year

    Dim hd As Holiday

    ' Suchliste (Array) durchlaufen

```

---

Listing 39: Zusammenstellen einer Liste bestimmter Feiertage

---

```

For Each hdbname As String In holidays
    ' Feiertag suchen
    hd = Me.Item(hdbname)

    ' und der neuen Liste hinzufügen
    list.Dictionary.Add(hd.Name, hd)
Next

Return list

End Function

```

---

*Listing 39: Zusammenstellen einer Liste bestimmter Feiertage (Forts.)*

Auch diese Liste ist unsortiert, da für die Values-Auflistung des Dictionary-Objektes keine Sortierung vorgesehen ist.

### Liste als sortiertes Array ausgeben

Wird eine nach Datum sortierte Liste benötigt, dann kann die Methode `ToArray` (Listing 40) angewandt werden. Nach Dimensionierung des Arrays wird mittels `CopyTo` das Array mit den Werten des Dictionary-Objektes gefüllt. `Array.Sort` verwendet die `IComparable`-Implementierung der Klasse `Holiday` und sorgt so für eine Sortierung nach Datum.

---

```

Public Function ToArray() As Holiday()

    ' Array definieren
    Dim hda(Dictionary.Count - 1) As Holiday

    ' Werte kopieren
    Dictionary.Values.CopyTo(hda, 0)

    ' Nach Datum sortieren
    Array.Sort(hda)

    Return hda

End Function

```

---

*Listing 40: Erzeugen eines sortierten Arrays mit den in der Liste enthaltenen Feiertagen*

Für die Auflistung der Feiertage von Nordrhein-Westfalen lässt sich leicht eine nach Datum sortierte Liste ausgeben:

---

```

For Each hd As Holiday In hdcNRW.ToArray()
    Debug.WriteLine(hd)
Next

```

---

ergibt:

---

```

Mittwoch, 1. Januar 2003 (Neujahr)
Freitag, 18. April 2003 (Karfreitag)
Montag, 21. April 2003 (Ostermontag)

```

---

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung  
Win-  
dows  
Forms

Controls

Propert-  
yGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

```

Donnerstag, 1. Mai 2003 (Tag der Arbeit)
Donnerstag, 29. Mai 2003 (Christi Himmelfahrt)
Montag, 9. Juni 2003 (Pfingstmontag)
Donnerstag, 19. Juni 2003 (Fronleichnam)
Freitag, 3. Oktober 2003 (Tag der Deutschen Einheit)
Samstag, 1. November 2003 (Allerheiligen)
Donnerstag, 25. Dezember 2003 (1. Weihnachtsfeiertag)
Freitag, 26. Dezember 2003 (2. Weihnachtsfeiertag)

```

---

### Abfrage, ob ein bestimmtes Datum ein Feiertag ist

GetHolidayByDate (Listing 41) steht Ihnen zur Verfügung, wenn Sie abfragen möchten, ob ein vorgegebenes Datum ein Feiertag ist. Hierzu wird die Liste der Feiertage durchlaufen und für jeden Eintrag überprüft, ob es sich um das Datum handelt. Wird das Datum gefunden, dann wird das zugehörige Holiday-Objekt zurückgegeben, ansonsten Nothing.

---

```

Public Function GetHolidayByDate(ByVal [Date] As DateTime) _
    As Holiday

    ' Liste durchsuchen und Datum vergleichen
    For Each hd As Holiday In Dictionary.Values
        If hd.Date = [Date] Then Return hd
    Next

    Return Nothing

End Function

```

---

*Listing 41: Überprüfen, welcher Feiertag auf das angegebene Datum fällt*

### Array mit Datumsinformationen generieren

Sind nur die Tage, nicht jedoch die Bezeichnungen der Feiertage von Bedeutung, dann kann mit Hilfe der Methode GetHolidayDates ein Array vom Typ DateTime abgerufen werden (Listing 42). Eine praktische Anwendung sehen Sie im nächsten Rezept.

---

```

Public Function GetHolidayDates() As DateTime()

    ' Array definieren
    Dim dates(Me.Count - 1) As DateTime

    ' Liste durchlaufen und Array füllen
    Dim i As Integer = 0
    For Each hd As Holiday In Dictionary.Values
        dates(i) = hd.Date
        i += 1
    Next

    Return dates

End Function

```

---

*Listing 42: Erstellen eines Arrays, das nur das jeweilige Datum der Feiertage enthält*



## 41 Darstellung der Feiertage im Kalender-Steuerelement

In Verbindung mit den oben beschriebenen Feiertagsberechnungen lässt sich schnell eine Klasse von `MonthCalendar` ableiten, die alle Feiertage fett darstellt und den Namen des Feiertags als ToolTip anzeigt, wenn der Anwender den Mauszeiger über dem betreffenden Tag positioniert. Die Klasse `MonthCalendar` bietet mit der Eigenschaft `BoldedDates` die Möglichkeit, bestimmte Tage im Monatskalender durch Fettschrift hervorzuheben. `BoldedDates` kann ein beliebiges Array vom Typ `DateTime` zugewiesen werden. Hier bietet sich der Einsatz der oben beschriebenen Methode `GetHolidayDates` der Klasse `HolidayCollection` an.

Da die Feiertagsliste immer nur für ein bestimmtes Jahr gültige Daten enthält, muss die Eigenschaft `BoldedDates` neu gesetzt werden, wenn sich in der Kalenderansicht das Jahr geändert hat. In `OnDateChanged` wird daher überwacht, ob der ausgewählte Monat in einem anderen Jahr liegt, und die Liste gegebenenfalls neu aufgebaut. Leider ist die Implementierung des `MonthCalendar`-Steuerelementes fehlerhaft. Wird die `BoldedDates`-Eigenschaft innerhalb des `DateChanged`-Ereignisses neu gesetzt, beschäftigt sich das Steuerelement anschließend mit sich selbst und ändert im Sekundenrhythmus das angezeigte Datum. Eine Entkopplung mit einem Timer hilft als Workaround, zumindest, wenn das Datum über die Pfeiltasten geändert wird.

Im `MouseMove`-Ereignis wird mit Hilfe der Methode `HitTest` bestimmt, über welchem Kalendertag sich der Mauszeiger befindet. Ist der Tag ein Feiertag, wird der Name des Feiertags mit `SetToolTip` angezeigt, ansonsten mit der gleichen Methode gelöscht. In Listing 43 sehen Sie die Implementierung der von `MonthCalendar` abgeleiteten Klasse `HolidayCalendar`, Abbildung 8 zeigt das Steuerelement im Einsatz.

```
Public Class HolidayCalendar
    Inherits MonthCalendar

    ... ' Variablen für Steuerelemente
    ' Referenz der Feiertagsliste
    Dim Holidaylist As HolidayCollection

    Public Sub New()
        MyBase.New()
        InitializeComponent()

        ' Feiertagsliste anlegen
        Holidaylist = New HolidayCollection(Me.SelectionStart.Year)
        ' Feiertage markieren
        Me.BoldedDates = Holidaylist.GetHolidayDates()
    End Sub

    Private Sub InitializeComponent()
        ...
    End Sub

    Protected Overrides Sub OnMouseMove(ByVal e As _
        System.Windows.Forms.MouseEventArgs)
```

---

```

' Basisklassenmethode aufrufen
MyBase.OnMouseMove(e)

' Datum an Mausposition ermitteln
Dim d As DateTime = Me.HitTest(e.X, e.Y).Time

' Bei ungültigem Datum Tooleptext löschen
If d.Year < 1600 Then
    ToolTip1.SetToolTip(Me, "")
    Return
End If

' Feiertag ermitteln
Dim hd As Holiday = Holidaylist.GetHolidayByDate(d)
If hd Is Nothing Then
    ' Wenn es keiner ist, Tooleptext löschen
    ToolTip1.SetToolTip(Me, "")
Else
    ' sonst den Namen des Feiertags als Tooltip anzeigen
    ToolTip1.SetToolTip(Me, hd.Name)
End If

End Sub

Protected Overrides Sub OnDateChanged(ByVal drevent As _
    System.Windows.Forms.DateRangeEventArgs)

    ' Wenn das Jahr wechselt, Timer starten
    If drevent.Start.Year <> Holidaylist.Year Then
        Timer1.Enabled = True
    End If

End Sub

Private Sub Timer1_Tick(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles Timer1.Tick

    ' Timer wieder stoppen
    Timer1.Enabled = False

    ' Neue Liste anlegen
    Holidaylist = New HolidayCollection(Me.SelectionStart.Year)

    ' Feiertage markieren
    Me.BoldedDates = Holidaylist.GetHolidayDates()

End Sub
End Class

```

---

Listing 43: Erweiterung des MonthCalendar-Controls durch Ableitung (Forts.)



Abbildung 8: Feiertage werden fett dargestellt, über ein ToolTip-Fenster wird der Name angezeigt

## 42 Gregorianisches Datum in Julianische Tageszählung

Die Julianische Tageszählung ist vom Aufbau sehr einfach gehalten. Es werden die Tage und Tagesbruchteile gezählt, die nach dem 1. Januar 4713 v. Chr. 12:00 Uhr vergangen sind. Es gibt also keine Monate und Jahre, sondern nur einen Tageswert, der Tag für Tag um den Wert 1 wächst.

Unser bürgerliches Datum, das so genannte Gregorianische Datum, wurde durch Papst Gregor XIII 1582 eingeführt. Vorher galt das Julianische Datum, welches von Gaius Julius Caesar eingeführt wurde und nichts mit der Julianischen Tageszählung zu tun hat. Um den Kalender, der damals sehr aus dem Takt geraten war, wieder mit der Realität der Beobachtung in Übereinstimmung zu bringen, folgte auf dem 4. Oktober 1582 direkt der 15. Oktober 1582. Dieser Gregorianische Kalender galt aber zunächst nicht überall. So führte Großbritannien diesen Kalender erst 1752 ein, die Türkei erst 1927. Um in der Wissenschaft, hauptsächlich der Astronomie, von solchen Abhängigkeiten nicht betroffen zu sein, hat sich schon früh die Julianische Tageszählung durchgesetzt. Sie wurde von Joseph Scalinger eingeführt, der diese Tageszählung nach seinem Vater Julius Scallinger benannte.

Möchte man größere Zeiträume betrachten, ist diese Tageszählung die alleinige Methode. Betriebssysteme moderner Computer haben noch immer ihr Problem mit langen Zeiträumen. So können je nach Darstellung nur Zeiten ab 1600, ab 1972 oder sogar ab 1 dargestellt werden, aber nicht einheitlich. Hier hilft die Julianische Tageszählung.

Um von einem gültigen bürgerlichen (Gregorianischen) Datum auf die Julianische Tageszählung (JD) zu kommen, ist in Listing 44 die Funktion `Greg2JD` zur Umrechnung eines Datumswertes aufgeführt.

```
Public Function Greg2JD(ByVal InDate As DateTime) As Double
    Dim mYear As Integer
    Dim mMonth As Integer
    Dim mDay As Integer
    Dim mDayDec As Double
    Dim mHour As Integer
    Dim mHourDec As Double
    Dim mMinute As Integer
    Dim mMinuteDec As Double
```

Listing 44: Umrechnung des Gregorianischen Datums in die Julianische Tageszählung

---

```

Dim mSecond As Integer
Dim mSecondDec As Double
Dim JD As Double
Dim JulReform As DateTime = New DateTime(1582, 10, 15)
Dim A As Integer
Dim B As Integer
Dim C As Integer
Dim D As Integer

mYear = InDate.Year
mMonth = InDate.Month
mDay = InDate.Day
mHour = InDate.Hour
mMinute = InDate.Minute
mSecond = InDate.Second

' Umrechnung in Bruchteile der jeweiligen Einheit
mHourDec = Convert.ToDouble(mHour) / 24.0
mMinuteDec = Convert.ToDouble(mMinute) / Convert.ToDouble(24 * 60)
mSecondDec = Convert.ToDouble(mSecond) / Convert.ToDouble(24 * _
    60 * 60)

' Zusammenfassung der Bruchteile in den Tagesbruchteil
mDayDec = mDay + mHourDec + mMinuteDec + mSecondDec

' Januar und Februar werden auf das vorhergehende Jahr gerechnet
If ((mMonth = 1) Or (mMonth = 2)) Then
    mYear -= 1
    mMonth += 12
End If

A = Int(mYear / 100)

' Datum vor dem 15.10.1582
If InDate < JulReform Then
    B = 0
Else
    B = 2 - A + Int(A / 4)
End If

' Hilfsvariable
A = CDb1(mYear + 4716)
C = Int(365.25 * CDb1(mYear + 4716))
D = Int(30.60001 * CDb1((mMonth + 1)))

' Berechnung des Julianischen Datums
JD = B + C + D + mDayDec - 1524.5

Return JD
End Function

```

---

Listing 44: Umrechnung des Gregorianischen Datums in die Julianische Tageszählung (Forts.)

Aus dem der Funktion übergebenen Wert vom Typ `DateTime` werden die einzelnen Bestandteile einer Datumsangabe ermittelt und Stunden, Minuten, Sekunden und Sekundenbruchteile als dezimale Nachkommastellen zur Tageszahl addiert. Da der Februar eine Besonderheit hat, werden Januar und Februar zum Vorjahr gezählt. Dabei wird eine Monatszahl akzeptiert, die es in der üblichen Kalenderzählung nicht gibt. So wird aus dem 15.01.2004 der 15.13.2003.

Um festzustellen, ob das Jahr des zu berechnenden Datums ein Schaltjahr ist (teilbar durch 4), aber eine Ausnahme darstellt (teilbar durch 100), wird die Hilfsvariable `A` berechnet. Der letzte Fall einer solchen Ausnahme war das Jahr 2000. Vor der Kalenderreform galt eine andere Regelung für Schaltjahre. Dieser Umstand wird durch die Variable `B` abgebildet. Anschließend wird das Julianische Datum (diese Bezeichnung ist auch üblich, aber nicht der Julianische Kalender) berechnet und dem aufrufenden Programm zurückgegeben.

Durch den Übergabeparameter vom Typ `DateTime` kann diese Funktion alle Datumswerte ab dem 01.01.0001 umrechnen. Will man Daten aus dem Zeitraum vor unserer Zeitrechnung in das Julianische Datum umrechnen, muss die Funktion umgebaut werden. Man kann zum Beispiel die einzelnen Datumsbestandteile der Funktion übergeben. Dieser Weg wurde hier nicht gewählt, da der Datentyp `DateTime` häufig Anwendung findet und die Zeit vor dem Jahr 1 eher selten auftritt.

## 43 Julianische Tageszählung in Gregorianisches Datum

Der entgegengesetzte Weg zu Listing 45 findet sich in Listing 45. Dieser Funktion wird das Julianische Datum (nicht zu verwechseln mit dem Julianischen Kalender – siehe Rezept 2.45 Gregorianisches Datum in Julianische Tageszählung) übergeben und es wird das Datum im Format `DateTime` zurückgeliefert. Dadurch ist diese Funktion auf Datumsangaben ab dem 01.01.0001 eingeschränkt.

```
Public Function JD2Greg(ByVal InJD As Double) As DateTime
    Dim mYear As Integer
    Dim mMonth As Integer
    Dim mDay As Integer
    Dim mDayDec As Double
    Dim mHour As Integer
    Dim mHourDec As Double
    Dim mMinute As Integer
    Dim mMinuteDec As Double
    Dim mSecond As Integer
    Dim mSecondDec As Double
    Dim ReturnDate As DateTime
    Dim JD As Double
    Dim A As Integer
    Dim Alpha As Integer
    Dim B As Integer
    Dim C As Integer
    Dim D As Integer
    Dim E As Integer
    Dim Z As Integer
    Dim F As Double
```

```
' Kleiner Null ist für JD nicht definiert
If InJD < 0 Then
    Dim ex As System.Exception = _
```

Listing 45: Umrechnung Julianische Tageszählung in Gregorianisches Datum

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

```

    New System.Exception("JD < 0 nicht berechenbar")
    Throw ex
End If

' JD beginnt 12:00 Greenwich
JD = InJD + 0.5
Z = Fix(JD)
F = JD - Z

' Datum vor dem 15.10.1582
If Z < 2299161 Then
    A = Z
Else
    Alpha = Fix((CDB1(Z) - 1867216.25) / 36524.25)
    A = Z + 1 + Alpha - Fix(Alpha / 4)
End If

' Hilfsvariable
B = A + 1524
C = Fix((CDB1(B) - 122.1) / 365.25)
D = Fix(365.25 * CDB1(C))
E = Fix(CDB1(B - D) / 30.60001)

' mit Dec wird der Dezimalteil der jeweiligen Größe bezeichnet
mDayDec = B - D - Fix(30.60001 * CDB1(E)) + F
mDay = Fix(mDayDec)
mHourDec = (mDayDec - mDay) * 24.0
mHour = Fix(mHourDec)
mMinuteDec = (mHourDec - mHour) * 60.0
mMinute = Fix(mMinuteDec)
mSecondDec = (mMinuteDec - mMinute) * 60.0
mSecond = Fix(mSecondDec)

' Resultiert aus dem Umbruch im Februar
If E < 14 Then
    mMonth = E - 1
Else
    mMonth = E - 13
End If

If mMonth > 2 Then
    mYear = C - 4716
Else
    mYear = C - 4715
End If

ReturnDate = _
    New DateTime(mYear, mMonth, mDay, mHour, mMinute, mSecond)

Return ReturnDate
End Function

```

---

In der Funktion wird zuerst geprüft, ob ein negatives Julianisches Datum übergeben wurde. Dies ist nicht definiert. Da das Julianische Datum mittags den Tageswechsel hat, wird ein halber Tag addiert und anschließend Tag und Bruchteil aufgesplittet. Um den Besonderheiten der Kalenderreform Rechnung zu tragen, wird der Zeitraum vor dem 15.10.1582 für die Hilfsvariable A gesondert betrachtet. Nach der Berechnung weiterer Hilfsvariablen werden die Tagesanteile berechnet. Sollte die Hilfsvariable  $E$  14 oder 15 sein, handelt es sich um die Monate Januar oder Februar. Hierbei muss vom Monat der Wert 13 abgezogen und zusätzlich das Jahr korrigiert werden.

In Abbildung 9 ist ein Beispiel für die Umrechnung eines gregorianischen Datums in ein Julianisches Datum zu sehen. In diesem Testprogramm wählt man mit den Schaltflächen in der Mitte des Formulars die gewünschte Umrechnungsmethode. Entsprechend werden die Eingabefelder und die Schaltfläche zum Start der Umrechnung freigeschaltet.

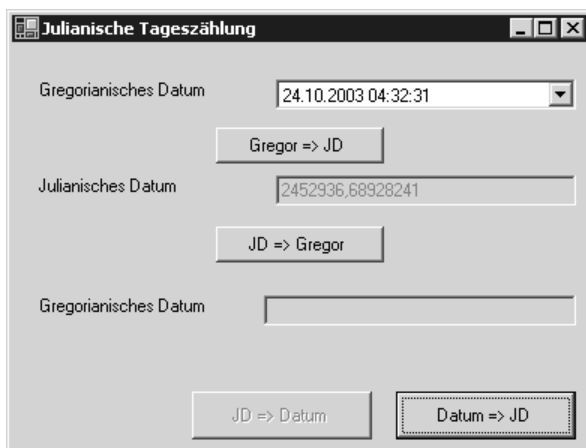


Abbildung 9: Gregorianisches Datum in Julianischer Tageszählung

## 44 Datum und Uhrzeit im ISO 8601-Format ausgeben und einlesen

Für die Ausgabe der Zeit als Ortszeit ohne UTC-Offset stellt die Struktur `DateTime` bereits ein passendes Standardmuster (Formatzeichen `s`) zur Verfügung:

```
Debug.WriteLine(DateTime.Now.ToString("s"))
```

gibt die aktuelle Zeit im ISO-Format aus

```
2003-10-22T12:57:09
```

Benötigen Sie eine eindeutige Zeitangabe mit Angabe der Zeitverschiebung zu UTC, dann müssen Sie das Formatmuster selbst zusammensetzen. Für die Zeitverschiebung wird das Formatmuster `z` benötigt:

```
Debug.WriteLine(DateTime.Now.ToString("yyyy-MM-ddTHH:mm:sszzzz"))
```

ergibt:

---

```
2003-10-22T13:02:34+02:00
```

---

Sommer- und Winterzeiteinstellungen werden berücksichtigt. Die folgende Schleife gibt die Zeitinformation im ISO-Format für einige Tage um den Tag der Umschaltung in die Winterzeit aus:

---

```
For i As Integer = 23 To 27
    Dim d As New DateTime(2003, 10, i, 12, 30, 0)
    Debug.WriteLine(d.ToString("yyyy-MM-ddTHH:mm:sszzzz"))
Next
```

---

und erzeugt folgende Ausgabe:

---

```
2003-10-23T12:30:00+02:00
2003-10-24T12:30:00+02:00
2003-10-25T12:30:00+02:00
2003-10-26T12:30:00+01:00
2003-10-27T12:30:00+01:00
```

---

Die Ausgabe zeigt jeweils den Zeitpunkt 12:30 Uhr, wobei bis zum 25.10. die Lokalzeit der UTC-Zeit um zwei Stunden vorseilt.

Auch Sekundenbruchteile können ausgegeben werden. Hierzu wird das Formatmuster `f` eingesetzt:

---

```
DateTime.Now.ToString("yyyy-MM-ddTHH:mm:ss.fffzzzz")
```

---

Dieser Code erzeugt einen String in dieser Form:

---

```
2003-10-22T13:11:27.103+02:00
```

---

Um eine Zeitangabe im ISO 8601-Format in eine `DateTime`-Struktur umzuwandeln, genügt ein Aufruf von `DateTime.Parse`. Die Methode kennt das ISO-Format und kann es interpretieren:

---

```
Dim t As String = DateTime.Now.ToString("yyyy-MM-ddTHH:mm:sszzzz")
Dim d As DateTime = DateTime.Parse(t)
Debug.WriteLine(t)
Debug.WriteLine(d)
```

---

Der Code formatiert die aktuelle Zeit im ISO-Format und wandelt sie anschließend wieder in eine `DateTime`-Struktur um. Folgende Ausgabe wird generiert:

---

```
2003-10-22T13:32:55+02:00
22.10.2003 13:32:55
```

---



Bei der Interpretation des ISO-Formates geht `DateTime.Parse` wie folgt vor:

- ▶ Fehlt der Offset zu UTC, dann wird die Zeit 1:1 als Lokalzeit interpretiert.
- ▶ Ist ein Offset zu UTC angegeben, dann wird dieser berücksichtigt, um die Standardzeit zu ermitteln. Diese Zeit wird dann auf die Zeitzone des Betriebssystems umgerechnet.

Die nach Aufruf von `DateTime.Parse` erhaltene Zeit ist also immer die Lokalzeit des Rechners. Wenn Sie die Zeitangabe in Standardzeit benötigen, müssen Sie anschließend die Konvertierungsmethode `ToUniversalTime` aufrufen.

---

Basics

---

**Datum/  
Zeit**

---

Anwen-  
dungen

---

Zeich-  
nen

---

Bild-  
bearbei-  
tung

---

Win-  
dows  
Forms

---

Controls

---

Proper-  
tyGrid

---

Datei-  
system

---

Netz-  
werk

---

System

---

Daten-  
banken

---

XML

---

Wissen-  
schaft

---

Ver-  
schiede-  
nes



# Anwendungen

In diesem Kapitel werden typische anwendungsspezifische Aufgabenstellungen behandelt. Hauptsächlich geht es hier um den Umgang mit Konfigurationsdaten. Beschrieben wird aber auch, wie man mehrfache Starts eines Anwendungsprogramms verhindern kann.

## 45 Anwendungskonfigurationsdatei erstellen und bearbeiten

Für eine Anwendung lässt sich eine XML-Konfigurationsdatei erstellen, in der Informationen abgelegt werden können, die für die Anwendung wichtig sind. Die Datei wird beim Start der Applikation geladen. Eine Reihe von Klassen und Methoden gibt Ihnen die Möglichkeit, auf diese Informationen zuzugreifen. Einige davon werden in den nachfolgenden Rezepten beschrieben.

Die Konfigurationsdatei einer Anwendung hat einen festgelegten Namen. Er setzt sich zusammen aus dem Namen der Anwendung und der Erweiterung `.config`. Für eine Applikation namens `App01` heißt die Konfigurationsdatei folglich `App01.exe.config`. Sie liegt im gleichen Verzeichnis wie die `.exe`-Datei der Anwendung.



Abbildung 10: Hinzufügen der Anwendungskonfigurationsdatei

Wenn Sie in Ihrer Anwendung nicht bereits durch andere Aktionen eine Konfigurationsdatei erzeugt haben, gehen Sie folgendermaßen vor:

Wählen Sie **PROJEKT/NEUES ELEMENT HINZUFÜGEN**. Im sich öffnenden Dialog wählen Sie den Eintrag **ANWENDUNGSKONFIGURATIONSDATEI** (Abbildung 10). Visual Studio erzeugt dann die Datei `App.config`, die Sie in der Projektmappe wieder finden. Anschließend starten Sie Ihr Programm

und beenden es wieder. Beim Start der Anwendung kopiert die Entwicklungsumgebung die Datei *App.config* in das Bin-Verzeichnis unter dem richtigen Namen (Endung *.config*). Sie finden diese Datei im Projektmappen-Explorer, wenn Sie die Ansicht auf ALLE DATEIEN ANZEIGEN setzen (siehe Abbildung 10).

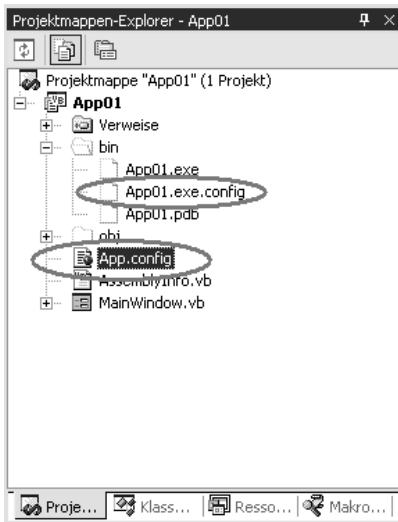


Abbildung 11: *App.config* wird automatisch nach *App01.exe.config* kopiert

Da bei jedem Start der Anwendung die Datei *App.config* kopiert wird, wird die eigentliche Konfigurationsdatei auch jedes Mal überschrieben. Folglich sollten Sie nur die Datei *App.config* im Projekt-Verzeichnis ändern, wenn Sie nicht Gefahr laufen wollen, dass Ihre Änderungen beim nächsten Start verloren gehen.

Die Einträge in der Konfigurationsdatei müssen bestimmten Konventionen gehorchen. Reihenfolge und Art der Knoten sind von entscheidender Bedeutung. Ungeschickte Änderungen können dazu führen, dass die Framework-Methoden nicht mehr mit der Datei arbeiten können. Lesen Sie die nachfolgenden Rezepte, um zu erfahren, wie die Einträge aussehen müssen.

## 46 Globale Werte aus der *.config*-Datei lesen

Einfache, bezogen auf die Konfiguration globale Daten können Sie im Element `<appSettings>` speichern. Dabei handelt es sich um Key/Value-Paare, bei denen sowohl der Schlüssel als auch der Wert aus einfachem Text besteht. Listing 46 zeigt ein Beispiel für eine Konfigurationsdatei, in der die Werte für *Company* und *Address* abgelegt sind.

---

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add key="Company" value="Addison-Wesley"/>
    <add key="Address" value="München"/>
  </appSettings>
</configuration>
```

---

Listing 46: Einfache globale Werte in der Sektion `<appSettings>`

Abfragen können Sie einen Wert über die statische Eigenschaft (Indexer) `ConfigurationSettings.AppSettings.Item`. Als Parameter wird der Schlüssel angegeben und als Ergebnis der gesuchte Text zurückgegeben. Wird der Schlüssel nicht gefunden, ist der Rückgabewert `Nothing`.

Die folgende Sequenz ruft die Texte für die Schlüssel `Company` und `Address` ab und zeigt die Texte auf zwei Label-Controls an (Abbildung 12).

```
Imports System.Configuration
...
LBLCompany.Text = ConfigurationSettings.AppSettings.Item("Company")
LBLAddress.Text = ConfigurationSettings.AppSettings.Item("Address")
```



Abbildung 12: Aus der Konfigurationsdatei gelesene Texte

## 47 Zusätzliche Konfigurationsdatei-Sektionen mit Attributwerten lesen

Wenn Sie Ihre Daten nicht gemeinsam mit anderen in der Section `<appSettings>` ablegen wollen, können Sie auch weitere Knoten hinzufügen. Diese müssen allerdings bekannt gemacht werden. Listing 47 zeigt ein Beispiel für eine Konfigurationsdatei mit einem zusätzlichen Knoten `CompanyInfo`.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <section name="CompanyInfo"
      type="System.Configuration.SingleTagSectionHandler" />
  </configSections>
  <CompanyInfo Name="Addison-Wesley" Address="München" />
</configuration>
```

Listing 47: Zusätzlicher Knoten in der Konfigurationsdatei mit Attributen

Innerhalb des `<configuration>`-Knotens muss als erster untergeordneter Knoten der `<configSections>`-Knoten angelegt werden. In diesem müssen alle zusätzlichen Sektionen mit `Name` und `Typ` deklariert werden. Der Typ `System.Configuration.SingleTagSectionHandler` bedeutet hierbei, dass die entsprechende Sektion keine Kindknoten besitzt, sondern die Werte über Attribute festlegt (im Beispiel die Attribute `Name` und `Address`). Der Zugriff auf einen solchen Knoten erfolgt über die statische Methode `ConfigurationSettings.GetConfig`:

```
Imports System.Configuration
...
Dim settings As Hashtable
```

---

```
' Hashtable abrufen
settings = DirectCast(ConfigurationSettings.GetConfig( _
    "CompanyInfo"), Hashtable)

' Werte aus Hashtable abrufen
LBLCompany.Text = CStr(settings("Name"))
LBLAddress.Text = CStr(settings("Address"))
```

---

Die Methode gibt eine Hashtable zurück, die alle Key/Value-Paare der angegebenen Sektion enthält.

## 48 Zusätzliche Konfigurationsdatei-Sektionen mit Key/Value-Einträgen lesen

Möchten Sie die Daten aus eigenen Sektionen lesen, die denselben Aufbau haben wie die <appSettings>-Sektion, dann müssen Sie im <configSections>-Knoten den Typ dieser Sektionen mit `System.Configuration.DictionarySectionHandler` angeben. Listing 48 zeigt den Aufbau der Konfigurationsdatei.

---

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <section name="CompanyInfo"
      type="System.Configuration.DictionarySectionHandler"/>
  </configSections>
  <CompanyInfo>
    <add key="Company" value="Addison-Wesley"/>
    <add key="Address" value="München"/>
  </CompanyInfo>
</configuration>
```

---

Listing 48: Zusätzliche Sektion *CompanyInfo* mit Key/Value-Liste

Der Zugriff im Programmcode ist identisch mit dem im vorherigen Beispiel. Wieder wird die Methode `GetConfig` aufgerufen und gibt eine Hashtable mit den Key/Value-Paaren der Sektion zurück:

---

```
Imports System.Configuration
...
Dim settings As Hashtable

' Hashtable abrufen
settings = DirectCast(ConfigurationSettings.GetConfig( _
    "CompanyInfo"), Hashtable)

' Werte aus Hashtable abrufen
LBLCompany.Text = CStr(settings("Company"))
LBLAddress.Text = CStr(settings("Address"))
```

---

## 49 Knoten beliebiger Struktur aus einer Konfigurationsdatei lesen

Sobald die Konfigurationsdaten komplexer werden, reichen die einfachen Key/Value-Paare nicht mehr für die Speicherung aus. Listen oder Baumstrukturen lassen sich so nicht sinnvoll in der Konfigurationsdatei ablegen. Da es sich bei der Konfigurationsdatei aber um eine wohlgeformte XML-Datei handelt, können Sie auch eigene Knoten hinzufügen, deren Struktur Sie selbst festlegen können. Voraussetzung ist natürlich, dass die XML-Syntax eingehalten wird.

Sektionen, deren Aufbau nicht vom Framework unterstützt wird, müssen mit einem eigenen Handler bearbeitet werden. Dieser muss im Deklarationsteil (`<configSections>`) der Konfigurationsdatei angegeben werden. In Listing 49 wird als Typ der Section `House` der voll qualifizierte Name der Klasse `HouseSectionHandler` angegeben. Zusätzlich muss, mit Komma abgetrennt, der Name der Assembly angegeben werden.

Der zusätzliche Knoten `<House>` beinhaltet das Attribut `Address` sowie beliebig viele Kindknoten vom Typ `<Floor>`. Diese wiederum besitzen das Attribut `Name` sowie beliebig viele Unterknoten vom Typ `<Room>`. Näheres zum Aufbau von XML-Dateien und deren Bearbeitung finden Sie in Kapitel 13 ((XML)).

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>

  <configSections>
    <!-- Deklaration der Sektion "House" -->
    <section name="House"
      type="App04.HouseSectionHandler,App04"/>
  </configSections>

  <!-- Sektion "House" mit beliebiger XML-Struktur -->
  <House Address="München, Berliner Straße">
    <Floor Name="First Floor">
      <Room Name="Entrance Hall"/>
      <Room Name="Livingroom"/>
      <Room Name="Kitchen"/>
      <Room Name="Diningroom"/>
    </Floor>
    <Floor Name="Second Floor">
      <Room Name="Sleepingroom"/>
      <Room Name="Child1"/>
      <Room Name="Child2"/>
    </Floor>
    <Floor Name="Third Floor">
      <Room Name="Office1"/>
      <Room Name="Office2"/>
    </Floor>
  </House>

</configuration>
```

Listing 49: Deklaration und Definition zusätzlicher beliebig strukturierter Sektionen

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Proper-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

Die in der Deklaration angegebene Klasse `App04.HouseSectionHandler` muss natürlich auch existieren. In Listing 50 sehen Sie eine mögliche Implementierung. Damit die Klasse vom Framework benutzt werden kann, muss sie die Schnittstelle `IConfigurationSectionHandler` und somit die Methode `Create` implementieren. `Create` bekommt als dritten Parameter die Referenz eines `XmlNode`-Objektes übergeben, das die XML-Struktur des Knotens repräsentiert.

Im Beispiel wird die XML-Struktur auf `TreeNode`-Objekte abgebildet, um diese später in einer `TreeView` anzeigen zu können. Die Analyse der Knoten erfolgt mit Hilfe der Klassen `XmlNode` und `XmlNodeList`, die in Kapitel 13 ((XML)) im Detail behandelt werden.

---

```
Imports System.Configuration
Imports System.Xml

Public Class HouseSectionHandler
    Implements IConfigurationSectionHandler

    Public Function Create(ByVal parent As Object, _
        ByVal configContext As Object, _
        ByVal section As System.Xml.XmlNode) As Object _
        Implements IConfigurationSectionHandler.Create

        ' Sicherheitsabfrage
        Debug.Assert(Not section Is Nothing)

        ' Neue TreeNode-Instanz anlegen
        Dim tn1 As New TreeNode("House")

        ' Alle <Floor>-Knoten durchlaufen
        For Each fnd As XmlNode In section.SelectNodes("Floor")

            ' Für jeden Knoten ein TreeNode-Objekt anlegen
            Dim tnf As New TreeNode(fnd.Attributes("Name").Value)

            ' Alle untergeordneten <Room>-Knoten durchlaufen
            For Each rnd As XmlNode In fnd.SelectNodes("Room")

                ' Für jeden Knoten ein TreeNode-Objekt anlegen
                Dim tnr As New TreeNode(rnd.Attributes("Name").Value)

                ' und an die Knotenliste des Floor-Knotens anhängen
                tnf.Nodes.Add(tnr)
            Next

            ' Floor-Knoten an die Knotenliste des House-Knotens anhängen
            tn1.Nodes.Add(tnf)

        Next

        ' Oberstes TreeNode-Objekt zurückgeben
        Return tn1
    End Function
End Class
```

---

Listing 50: `HouseSectionHandler` erzeugt aus den Informationen des `<House>`-Knotens ein verzweigtes `TreeNode`-Objekt



---

End Function

End Class

---

*Listing 50: HouseSectionHandler erzeugt aus den Informationen des <House>-Knotens ein verzweigtes TreeNode-Objekt (Forts.)*

Zugegriffen wird auf die Daten wieder ähnlich wie in den vorangegangenen Rezepten mit `ConfigurationSettings.GetConfig` unter Angabe des Sektionsnamens:

---

```
' Sektion "House" abrufen, generiert ein verzweigtes
' TreeNode-Objekt
Dim o As Object = ConfigurationSettings.GetConfig("House")
```

---

Dieser Aufruf führt dazu, dass eine Instanz der Klasse `HouseSectionHandler` erzeugt und über das Interface  `IConfigurationSectionHandler` die Methode `Create` aufgerufen wird. Der in Listing 50 abgebildete Code wird ausgeführt und gibt die Referenz eines `TreeNode`-Objektes zurück. Dieses Objekt wird nach einem Typecast der `Nodes`-Auflistung der `TreeView` hinzugefügt. Das Ergebnis sehen Sie in Abbildung 13.

---

```
Dim tn As TreeNode = DirectCast(o, TreeNode)

' An vorhandenes TreeView-Control anhängen
TreeView1.Nodes.Add(tn)
```

---



Abbildung 13: Anzeige der Informationen aus der Sektion <House> der Konfigurationsdatei

Mit dieser Vorgehensweise können Sie beliebig komplexe Daten in die Konfigurationsdatei einer Anwendung aufnehmen und einlesen. Sie können sogar mit Hilfe von `<sectionGroup>`-Deklarationen die Daten noch weiter strukturieren. Weitergehende Informationen hierzu finden Sie in der MSDN-Hilfe zu `<configSections>`.

## 50 Konfigurationsdatei zur Laufzeit ändern

Sie wollen zur Laufzeit Daten in der .config-Datei ändern? Unser Rezept hierzu:

Besser nicht!

Microsoft stellt im Framework eine Reihe von Techniken zur Verfügung, um aus der Konfigurationsdatei Daten strukturiert zu lesen. In den vorangegangenen Rezepten wurde das ausführlich erläutert. Zum Schreiben in die .config-Datei hat Microsoft aber nichts dergleichen vorgesehen, und das wahrscheinlich aus gutem Grund.

Die .config-Datei einer Applikation wird einmalig beim Start gelesen. Änderungen nach Programmstart können erst beim nächsten Start berücksichtigt werden.

Aber das größte Problem ist, dass die Datei im gleichen Verzeichnis liegen muss wie die .exe-Datei der Applikation und Sie sich nicht darauf verlassen können, dass Ihr Programm Schreibrechte für dieses Verzeichnis besitzt.

In anderen Büchern und sogar in einigen Beispielen von Microsoft finden Sie die Anleitung, auf die .config-Datei mit XML-Klassen wie z.B. XmlDocument zuzugreifen und die Datei so zur Laufzeit zu ändern. Wenn das wirklich sein muss, schlagen Sie in Kapitel 13 ((XML)) nach, wie Sie mit XML-Dateien arbeiten und umgehen. Aber wie bereits gesagt: Besser nicht.

Lesen Sie in den nachfolgenden Rezepten, wie Sie Konfigurationsdateien als XML-Datei in anderen Verzeichnissen ablegen können, ohne die Einschränkungen der .config-Datei berücksichtigen zu müssen.

## 51 Eine eigene Konfigurationsdatei für Ihre Anwendung

Konfigurationsdateien lassen sich derzeit am besten in Form von XML-Dateien gestalten. Zum einen bietet XML gute Strukturierungsmöglichkeiten, zum anderen das Framework hervorragende Unterstützung für den Umgang mit XML (siehe Kapitel 13 ((XML))). Da es sich bei XML-Dateien um Textdateien handelt, kann man sie notfalls auch mit einem einfachen Texteditor bearbeiten.

Zunächst muss die Frage geklärt werden, wo eine solche Datei gespeichert werden soll. Das Verzeichnis, in dem die .exe-Datei der Anwendung liegt, wird nur selten zur Verfügung stehen, da, besonders in Firmennetzen, der jeweilige Anwender nicht als Administrator angemeldet ist und daher meist keine Schreibrechte für dieses Verzeichnis hat. Windows sieht für solche Zwecke andere Verzeichnisse vor. Je nach Aufgabenstellung können Sie eines der in Tabelle 11 aufgeführten für die Speicherung Ihrer Konfigurationsdaten benutzen.

| SpecialFolder-Konstante | Beispielverzeichnis                                                                      |
|-------------------------|------------------------------------------------------------------------------------------|
| ApplicationData         | <i>D:\Dokumente und Einstellungen\Poweruser\Anwendungsdaten</i>                          |
| Personal                | <i>D:\Dokumente und Einstellungen\Peter\Eigene Dateien</i>                               |
| LocalApplicationData    | <i>D:\Dokumente und Einstellungen\Administrator\Lokale Einstellungen\Anwendungsdaten</i> |
| CommonApplicationData   | <i>D:\Dokumente und Einstellungen\All Users\Anwendungsdaten</i>                          |
| CommonProgramFiles      | <i>D:\Programme\Gemeinsame Dateien</i>                                                   |

Tabelle 11: Mögliche Verzeichnisse für die Ablage von Konfigurationsdateien

Eine simple, aber effektive Möglichkeit zum Lesen und Speichern von Anwendungs- und Benutzerdaten in XML-Dateien ist der Einsatz der Klasse `XmlSerializer` (vergleiche Rezept 13.19 ((Serialisierung mit Hilfe der Klasse `XmlSerializer`))). Hierzu definieren Sie einfach eine Klasse, die alle benötigten Daten in öffentlichen Member-Variablen speichern kann. Beim Start der Anwendung erzeugen Sie mit Hilfe der Methode `XmlSerializer.Deserialize` eine Instanz dieser Klasse mit den gespeicherten Daten, beim Schließen der Anwendung speichern Sie die Informationen durch Aufruf der Methode `XmlSerializer.Serialize` ab.

Ein kleines Beispiel soll die prinzipielle Vorgehensweise demonstrieren. Folgende Daten sollen in einer Konfigurationsdatei abgelegt werden:

```
' Konfigurationsdaten
Public NextMeeting As DateTime = DateTime.Now.AddDays(4)
Public Company As String
Public MaxNumberOfFiles As Integer = 4
Public Documents As New ArrayList
```

Diese Datenfelder werden als öffentliche Member-Variablen der Klasse `ConfigData` (Listing 50) definiert. `ConfigData` enthält neben den Daten zwei öffentliche Methoden:

- die Instanz-Methode `Save` zum Speichern der Daten sowie
- die statische Methode `Load` zum Lesen der Daten.

In der Methode `Save` wird zunächst überprüft, ob das Verzeichnis, in dem die Datei gespeichert werden soll, existiert. Wenn nicht, wird es angelegt. Anschließend wird eine Instanz der Klasse `XmlSerializer` angelegt und die Datei mit Hilfe einer `StreamWriter`-Instanz zum Schreiben geöffnet. Mit `Serialize` werden die Daten als XML-Struktur in der Datei gespeichert. Zuletzt wird der Stream wieder geschlossen. Das Schließen wurde in den `Finalize`-Block ausgelagert, damit die Datei auch geschlossen wird, wenn bei der Serialisierung Fehler aufgetreten sind. Aufgetretene Fehler werden mit einer zusätzlichen Meldung weitergeleitet, wobei die Referenz der ursprünglichen Exception als `InnerException` weitergegeben wird. Ein Beispiel für die erzeugte XML-Datei zeigt Listing 51

```
<?xml version="1.0" encoding="utf-8"?>
<ConfigData xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <NextMeeting>2003-10-21T12:41:06.5577520+02:00</NextMeeting>
  <Company>Addison-Wesley</Company>
  <MaxNumberOfFiles>4</MaxNumberOfFiles>
  <Documents>
    <anyType xsi:type="xsd:String">D:\Dokumente und ...
  </anyType>
    <anyType xsi:type="xsd:String">D:\Dokumente und ...
  </anyType>
    <anyType xsi:type="xsd:String">D:\Dokumente und ...
  </anyType>
    <anyType xsi:type="xsd:String">D:\Dokumente und ...
  </anyType>
  </Documents>
</ConfigData>
```

Listing 51: Ausschnitt aus der erzeugten Konfigurationsdatei

Das fast spiegelbildliche Gegenstück zu `Save` ist die Methode `Load`. Im Unterschied zu `Save` muss `Load` als statische (Shared) Methode definiert werden, da ja zum Zeitpunkt des Ladens noch keine Instanz der Klasse `ConfigData` existiert. Zunächst wird überprüft, ob die Konfigurationsdatei existiert. Ist das nicht der Fall, dann wird eine neue Instanz der Klasse `ConfigData` angelegt und deren Referenz zurückgegeben.

Existiert die Datei, wird sie mit Hilfe von `StreamReader` und `XmlSerializer.Deserialize` gelesen und eine neue Instanz von `ConfigData` angelegt und mit den gespeicherten Daten initialisiert. Im Fehlerfall wird auch hier wieder die Exception weitergeleitet.

---

```
Imports System.IO
Imports System.Xml.Serialization

Public Class ConfigData

    ' Konfigurationsdaten
    Public NextMeeting As DateTime = DateTime.Now.AddDays(4)
    Public Company As String
    Public MaxNumberOfFiles As Integer = 4
    Public Documents As New ArrayList

    ' Speichern mit XmlSerializer
    Public Sub Save(ByVal path As String)

        Dim xs As XmlSerializer
        Dim sw As StreamWriter

        Try
            ' Verzeichnis anlegen, falls es nicht existiert
            Dim di As DirectoryInfo = Directory.GetParent(path)
            If Not di.Exists Then di.Create()

            ' XmlSerializer-Instanz für diese Klasse
            xs = New XmlSerializer(GetType(ConfigData))

            ' Datei mit StreamWriter öffnen
            sw = New StreamWriter(path)

            ' Serialisieren
            xs.Serialize(sw, Me)

        Catch ex As Exception
            ' Bei Fehlern Exception als InnerException weiterreichen
            Throw New Exception( _
                "Speichern der Konfigurationsdaten nicht möglich", ex)

        Finally
            ' Datei schließen
            If Not sw Is Nothing Then sw.Close()
        End Try
    End Sub
End Class
```

---

---

End Sub

*' Laden mit XmlSerializer*

Public Shared Function Load(ByVal path As String) As ConfigData

Dim xs As XmlSerializer

Dim sr As StreamReader

Try

*' Wenn die Datei noch nicht existiert, leere Instanz*
*' zurückgeben*

If Not File.Exists(path) Then Return New ConfigData

*' XmlSerializer-Instanz für diese Klasse*

xs = New XmlSerializer(GetType(ConfigData))

*' Datei mit StreamReader öffnen*

sr = New StreamReader(path)

*' Deserialisieren*

Dim data As ConfigData

data = DirectCast(xs.Deserialize(sr), ConfigData)

*' Kein Fehler, Rückgabe der deserialisierten Daten*

Return data

Catch ex As Exception

*' Bei Fehlern Exception als InnerException weiterreichen*
Throw New Exception( \_  
" Laden der Konfigurationsdaten nicht möglich", ex)

Finally

*' Datei schließen*

If Not sr Is Nothing Then sr.Close()

End Try

End Function

End Class

---

Listing 52: Klasse ConfigData zum Lesen und Speichern der Konfigurationsdaten (Forts.)

Ein kleines Anwendungsprogramm lädt die Konfigurationsdaten im Load-Ereignis des Hauptfensters (Listing 53). Als Ablagepfad wird hier ein Unterverzeichnis von ApplicationData genutzt. Der zusammengesetzte Pfad ergibt sich für einen als Administrator angemeldeten Benutzer etwa zu:

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Proper-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

---

```

"D:\Dokumente und Einstellungen\Administrator\Anwendungsdaten\App05\config.xml"
' Referenz der Konfigurationsdaten
Public Config As ConfigData

' Pfad der Konfigurationsdatei
Public ConfigPath As String

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

    ' Pfad für Anwendungsdaten
    ConfigPath = Environment.GetFolderPath( _
        Environment.SpecialFolder.ApplicationData)

    ' mit Produktname-Unterverzeichnis und Dateiname ergänzen
    ConfigPath &= "\" & Application.ProductName & "\config.xml"

    ' Konfiguration laden
    Config = ConfigData.Load(ConfigPath)

End Sub

```

---

#### Listing 53: Laden der Konfigurationsdaten

Nach Aufruf von `ConfigData.Load` verweist die Member-Variable `Config` auf die Instanz der Klasse `ConfigData`. Über diese kann jederzeit auf die Konfigurationsdaten zugegriffen werden. Listing 54 zeigt beispielsweise die Implementierung eines Konfigurations-Dialogs, wie Sie ihn in Abbildung 14 sehen können.

---

```

Public Class OptionsDialog
    Inherits System.Windows.Forms.Form

    ' ... Vom Windows Form Designer generierter Code

    Public Shared Function CreateAndShow(ByVal config As ConfigData) _
        As DialogResult

        ' Dialog-Instanz anlegen
        Dim dialog As New OptionsDialog

        Try
            ' Steuerelemente initialisieren
            dialog.TBCompany.Text = config.Company
            dialog.LBFiles.Items.AddRange(config.Documents.ToArray())
            dialog.NUDMaxFiles.Value = config.MaxNumberOfFiles
            dialog.DTPNextMeeting.Value = config.NextMeeting

        Catch ex As Exception

```

---

Listing 54: Auf die Konfigurationsdaten kann jederzeit zugegriffen werden, wie hier in einer Dialog-Implementierung demonstriert wird

```

        Debug.WriteLine(ex.Message)
    End Try

    ' Dialog anzeigen
    Dim dr As DialogResult = dialog.ShowDialog()

    ' Bei OK, Daten übernehmen
    If dr = System.Windows.Forms.DialogResult.OK Then
        config.Company = dialog.TBCompany.Text
        config.MaxNumberOfFiles = CInt(dialog.NUDMaxFiles.Value)
        config.NextMeeting = dialog.DTPNextMeeting.Value
    End If

End Function

End Class

```

*Listing 54: Auf die Konfigurationsdaten kann jederzeit zugegriffen werden, wie hier in einer Dialog-Implementierung demonstriert wird (Forts.)*

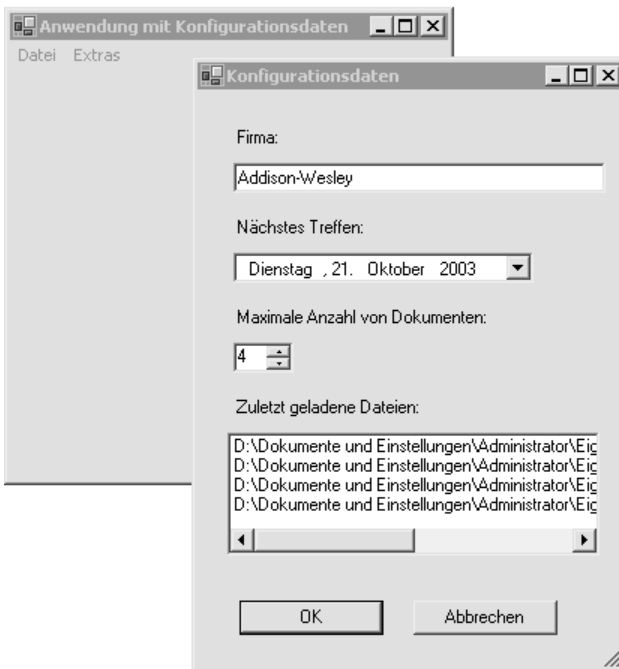


Abbildung 14: Der Konfigurations-Dialog greift auf die aus der Konfigurationsdatei gelesenen Daten zu

Gespeichert werden die Konfigurationsdaten im `Closed`-Ereignis des Hauptfensters (Listing 55). Sie können aber jederzeit, wenn Sie einen Zwischenstand festhalten wollen, mittels `Save` die Konfigurationsdaten abspeichern.

Basics

Datum/  
ZeitAnwen-  
denZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Propor-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

---

```
Private Sub MainWindow_Closed(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles MyBase.Closed

    ' Konfiguration speichern
    Config.Save(ConfigPath)

End Sub
```

---

*Listing 55: Speichern der Konfigurationsdaten beim Schließen der Anwendung*

Über Attribute lässt sich der Aufbau der XML-Datei steuern. Beachten Sie aber bitte, dass der `XmlSerializer` nicht mit allen Datentypen umgehen kann. Details hierzu finden Sie in Rezept [13.19](#) ((Serialisierung mit Hilfe der Klasse `XmlSerializer`)). Eine Alternative zur Klasse `XmlSerializer` ist die Klasse `SoapFormatter` (siehe Rezept [13.21](#) ((Serialisierung mit Hilfe der Klasse `SoapFormatter`))), die sich zwar besser für die Serialisierung komplexer Daten eignet, aber XML-Dateien erzeugt, die von Hand nicht mehr sinnvoll angepasst werden können.

Bei Verwendung des `XmlSerializers` können Sie auch so vorgehen, dass Sie erst eine Schema-Datei anlegen, die Aufbau und Umfang der Daten festlegt, und anschließend mit `xsd.exe` hieraus die Klassendatei generieren (vergleiche Rezept [13.18](#) ((Klassenhierarchie aus XML-Schema erzeugen))). Welche Variante Sie wählen, hängt davon ab, ob Sie auf die Struktur der XML-Datei Einfluss nehmen wollen oder nicht.

## 52 Konfigurationsmanager und -daten trennen

Um etwas mehr Überblick über die Klassenstruktur zu erhalten, können Sie die eigentlichen Konfigurationsdaten getrennt vom Serialisierungsmechanismus in einer eigenen Klasse definieren. Die Serialisierung dieser Daten übernimmt ein Konfigurationsmanager. Dieser verwaltet zusätzlich einige oft benötigte Informationen, wie Datum und Zeit des letzten Programmstarts und der letzten Speicherung und die Anzahl der Starts. Zusätzlich entnimmt er den Pfad der Konfigurationsdatei den Informationen aus der `.config`-Datei der Anwendung.

Der Ablageort der Konfigurationsdatei wird in zwei Teile aufgeteilt. Im ersten Teil (Schlüssel `DirectoryConfigFile`) wird die Enumerations-Konstante für den Basispfad angegeben, im zweiten Teil (Schlüssel `LocalPathConfigFile`) der lokale Pfad der Datei. Ein Beispiel für eine Konfiguration in der `.config`-Datei zeigt Listing 56.

---

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add key="DirectoryConfigFile" value="ApplicationData"/>
    <add key="LocalPathConfigFile"
      value="VBCodeBook\App06\Config.xml"/>
  </appSettings>
</configuration>
```

---

*Listing 56: In der .config-Datei wird festgelegt, wo die Konfigurationsdatei liegt*

Der Konfigurationsmanager (Listing 57) liest in der statischen Methode `Load` die beiden Werte ein, ermittelt den Basispfad und setzt beide Teile zusammen. Das Ergebnis wird für spätere `Save`-Aufrufe in `configPath` gespeichert.



In der öffentlichen Member-Variablen `Data` wird die Referenz der getrennt gehaltenen Konfigurationsdaten gespeichert. Leider muss der `XmlSerializer` den Datentyp kennen. Daher wird die Referenz nicht als allgemeine `Object`-Referenz definiert, sondern der tatsächliche Klassenname (`ConfigData`) angegeben. Achten Sie an dieser Stelle auch auf die Besonderheiten, die sich im Zusammenhang mit dem `XmlSerializer` und abgeleiteten Klassen ergeben (Rezept 13.19 ((Serialisierung mit Hilfe der Klasse `XmlSerializer`))).

Die private Member-Variablen `DataLoader` bzw. die zugehörige öffentliche schreibgeschützte Eigenschaft `DataLoadedFromFile` geben an, ob die Konfigurationsdaten durch Deserialisierung der Konfigurationsdatei oder durch Instanzierung mit `New` angelegt worden sind. Hiermit kann das übergeordnete Programm feststellen, ob die Konfigurationsdatei bereits vorlag oder erst mit dem nächsten Aufruf von `Save` angelegt wird. Im ersten Fall wird der Zähler `NumberOfStarts` inkrementiert.

---

```
Imports System.Configuration
Imports System.IO
Imports System.Xml.Serialization

Public Class ConfigManager

    ' Standard-Informationen
    Public LastStarted As DateTime = DateTime.Now
    Public LastSaved As DateTime = DateTime.Now
    Public NumberOfStarts As Integer = 0

    Private DataLoaded As Boolean = False
    Public ReadOnly Property DataLoadedFromFile() As Boolean
        Get
            Return DataLoaded
        End Get
    End Property

    ' Verweis auf die eigentlichen Daten
    Public Data As ConfigData

    ' Merker für die Startzeit
    Private thisTimeStarted As DateTime = DateTime.Now

    ' Pfad der XML-Datei
    Private Shared configPath As String

    ' Öffentlicher Zugriff auf Pfad
    Public ReadOnly Property Path() As String
        Get
            Return configPath
        End Get
    End Property

    ' Konfigurationsdatei laden
    Public Shared Function Load() As ConfigManager
```

---

Listing 57: Der eigenständige Konfigurationsmanager benötigt nur einen Verweis auf die eigentlichen Daten

---

Basics

---

Datum/  
Zeit

---

Anwen-  
dungen

---

Zeich-  
nen

---

Bild-  
bearbei-  
tung

---

Win-  
dows  
Forms

---

Controls

---

Propor-  
tityGrid

---

Datei-  
system

---

Netz-  
werk

---

System

---

Daten-  
banken

---

XML

---

Wissen-  
schaft

---

Ver-  
schiede-  
nes

---

---

```

Dim xs As XmlSerializer
Dim sr As StreamReader
Dim cm As ConfigManager

Try

    ' Pfad zusammensetzen
    Dim dcf As String = ConfigurationSettings.AppSettings.Item( _
        "DirectoryConfigFile")

    configPath = Environment.GetFolderPath(DirectCast( _
        [Enum].Parse(GetType(Environment.SpecialFolder), dcf), _
        Environment.SpecialFolder))

    configPath &= "\" & ConfigurationSettings.AppSettings.Item( _
        "LocalPathConfigFile")

    ' Wenn die Datei noch nicht existiert, leere Instanz
    ' zurückgeben
    If Not File.Exists(configPath) Then
        cm = New ConfigManager
        cm.Data = New ConfigData
        Return cm
    End If

    ' XmlSerializer-Instanz für diese Klasse
    xs = New XmlSerializer(GetType(ConfigManager))

    ' Datei mit StreamReader öffnen
    sr = New StreamReader(configPath)

    ' Deserialisieren
    cm = DirectCast(xs.Deserialize(sr), ConfigManager)

    ' Daten als Geladen vermerken
    cm.DataLoaded = True

    ' Anzahl Starts inkrementieren
    cm.NumberOfStarts += 1

    ' Kein Fehler, Rückgabe der deserialisierten Daten
    Return cm

Catch ex As Exception
    ' Bei Fehlern Exception als InnerException weiterreichen
    Throw New Exception( _
        "Laden der Konfigurationsdaten nicht möglich", ex)

Finally

```

---

Listing 57: Der eigenständige Konfigurationsmanager benötigt nur einen Verweis auf die eigentlichen Daten (Forts.)

```

        ' Datei schließen
        If Not sr Is Nothing Then sr.Close()

    End Try

End Function

' Speichern mit XmlSerializer
Public Sub Save()

    Dim xs As XmlSerializer
    Dim sw As StreamWriter

    Try
        ' Verzeichnis anlegen, falls es nicht existiert
        Dim di As DirectoryInfo = Directory.GetParent(configPath)
        If Not di.Exists Then di.Create()

        ' Zeitangaben setzen
        LastSaved = DateTime.Now
        LastStarted = thisTimeStarted

        ' XmlSerializer-Instanz für diese Klasse
        xs = New XmlSerializer(GetType(ConfigManager))

        ' Datei mit StreamWriter öffnen
        sw = New StreamWriter(configPath)

        ' Serialisieren
        xs.Serialize(sw, Me)

    Catch ex As Exception
        ' Bei Fehlern Exception als InnerException weiterreichen
        Throw New Exception( _
            "Speichern der Konfigurationsdaten nicht möglich", ex)
    Finally
        ' Datei schließen
        If Not sw Is Nothing Then sw.Close()
    End Try

End Sub

End Class

```

*Listing 57: Der eigenständige Konfigurationsmanager benötigt nur einen Verweis auf die eigentlichen Daten (Forts.)*

Beim Speichern der Konfigurationsdaten werden die zusätzlichen Zeit-Informationen (LastSaved, LastStarted) gesetzt und wie im vorangegangenen Beispiel die gesamte Datenstruktur in der XML-Datei gespeichert. Für den in Listing 58 gezeigten Aufbau der Klasse ConfigData ergibt sich ein Aufbau der XML-Datei, wie er in Listing 59 verkürzt wiedergegeben wird.

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Proper-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schie-  
des

---

```
Public Class ConfigData

    Public User As String = "Donald Duck"
    Public Hometown As String
    Public Neighbours() As String
    Public ID As Integer

End Class
```

---

*Listing 58: Die Klasse für die Konfigurationsdaten wurde auf das Wesentliche reduziert*

---

```
<?xml version="1.0" encoding="utf-8"?>
<ConfigManager xmlns:xsd="http://www.w3.org/2001/XMLSchema" ...
  <LastStarted>2003-10-17T18:29:47.3804176+02:00</LastStarted>
  <LastSaved>2003-10-17T18:29:54.1201088+02:00</LastSaved>
  <NumberOfStarts>4</NumberOfStarts>
  <Data>
    <User>Donald Duck</User>
    <Hometown>Ducksburg</Hometown>
    <Neighbours>
      <String>Mickey Mouse</String>
      <String>Goofy</String>
      <String>Daisy</String>
    </Neighbours>
    <ID>123</ID>
  </Data>
</ConfigManager>
```

---

*Listing 59: Beispiel für die erzeugte Konfigurationsdatei*

Auch die Struktur der erzeugten XML-Datei erscheint etwas übersichtlicher. Alle Daten der Klasse ConfigData befinden sich im Knoten <Data>.

Die Benutzung des Konfigurationsmanagers im Hauptprogramm gestaltet sich wiederum recht einfach. Sie benötigen lediglich eine Referenzvariable für die Instanz von ConfigManager und, wenn gewünscht, eine für die Instanz von ConfigData:

---

```
Public CM As ConfigManager = ConfigManager.Load()
Public CData As ConfigData = CM.Data
```

---

Beim Beenden der Anwendung rufen Sie wiederum die Save-Methode auf, um die Konfigurationsdatei zu speichern:

---

```
Private Sub MainWindow_Closed ...
    CM.Save()
End Sub
```

---

## 53 Verhindern, dass mehrere Instanzen einer Anwendung gleichzeitig gestartet werden können

Während die mehrfache Instanziierung einer Klasse durch Einsatz des Singleton-Patterns verhindert werden kann, muss man auf der Ebene der Applikationen etwas tiefer in die Trickkiste greifen. Da jeder Prozess seinen eigenen Adressraum hat, helfen statische Variablen, wie sie zur Realisierung des Singleton-Patterns verwendet werden, hier nicht weiter.

Um herauszufinden, ob bereits eine Instanz einer Anwendung läuft, gibt es verschiedene Möglichkeiten. Eine oft beschriebene, aber relativ unzuverlässige ist die Überprüfung der Fenstertitel der laufenden Prozesse. Unzuverlässig ist sie deshalb, weil sich der Titel während der Laufzeit ändern kann (z.B. zur Anzeige eines gerade in Bearbeitung befindlichen Dokumentes). Auch der Vergleich der Prozessnamen ist nicht eindeutig, da verschiedene Programme in unterschiedlichen Verzeichnissen den gleichen Namen haben können. Nicht einmal der Vergleich der Pfade ist sicher, denn die .exe-Datei könnte in ein anderes Verzeichnis kopiert und von dort gestartet werden.

Ein anderer Ansatz besteht darin, in einer Datei eine Information abzulegen, die von anderen Instanzen abgefragt werden kann. Hier ergibt sich allerdings die Gefahr, dass der Dateieintrag nicht mehr zurückgesetzt wird, wenn das Programm vorzeitig abstürzt (soll vorkommen ;-)). Dann ließe sich das Programm anschließend gar nicht mehr starten, was weder im Sinne des Anwenders noch des Entwicklers wäre.

Ohne Hilfsmittel des Betriebssystems wird man daher nicht weiterkommen. Es bietet sich der Einsatz eines *Mutex* (Mutual Exclusion)-Objektes an, das für die Synchronisation von Threads und Prozessen bereitgestellt wird. Mutexe sind prozessübergreifend gültig. Ein Prozess kann einen Mutex anlegen und belegen. Andere Prozesse können überprüfen, ob der Mutex existiert, und entsprechend reagieren.

Das Framework stellt für den Umgang mit Mutexen die Klasse `System.Threading.Mutex` zur Verfügung. Bei der Instanziierung eines Mutex-Objektes kann dem Konstruktor die Referenz einer booleschen Variablen übergeben werden. Der Konstruktor setzt die Variable auf `True`, wenn der Mutex neu angelegt wurde, und auf `False`, wenn er schon existiert.

Um einen Mutex identifizieren zu können, muss er einen eindeutigen Namen haben. Damit von vornherein Namenskollisionen mit anderen Programmen ausgeschlossen werden, benutzen Sie für den Mutex-Namen am besten eine Guid. Diese können Sie in der Entwicklungsumgebung über das Menü EXTRAS/GUID ERSTELLEN generieren.

Achtung

Falls Sie das Beispiel kopieren, benutzen Sie bitte nicht die dort angegebene Guid, sondern generieren Sie eine neue. Ansonsten gibt es vielleicht zukünftig mehrere Programme, die alle mit dem gleichen Mutex arbeiten.

Stellt sich als nächste Frage, wann denn der Mutex angelegt bzw. abgefragt wird. Die sicherste Lösung ist, den erforderlichen Code so zu platzieren, dass er vor jedem anderen Code ausgeführt wird: also am Anfang von `Sub Main`. Listing 60 zeigt eine mögliche Lösung. Nach Anlegen des Mutex-Objektes wird das Hauptfenster nur dann angezeigt, wenn der Mutex neu angelegt wurde.

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Propor-  
talityGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

```

Public Shared Sub Main()

    Dim createdNew As Boolean

    ' ID für Mutex - BITTE NICHT KOPIEREN, SONDERN NEUE ERZEUGEN
    Dim mutexGuid As String = "{A0330AB0-8961-415a-A13E-86A585925C4C}"

    ' Mutex abfragen und ggf. belegen
    Dim mtx As New System.Threading.Mutex(False, mutexGuid, _
        createdNew)

    ' War der Mutex bereits angelegt?
    If createdNew Then

        ' Nein, das ist die einzige Instanz auf dieser Maschine
        ' Hauptfenster anzeigen
        Application.Run(New MainWindow)

    Else

        ' Ja, es läuft bereits eine Instanz
        ' Aktueller Prozess:
        Dim curProcess As Process = Process.GetCurrentProcess

        ' Alle Prozesse mit gleichem Namen abrufen
        Dim processes() As Process = _
            Process.GetProcessesByName(curProcess.ProcessName)

        ' Liste durchlaufen (sollte in der Regel zwei Einträge haben)
        For Each p As Process In processes

            ' Richtige Applikation?
            If p.MainModule.FileName = curProcess.MainModule.FileName Then

                ' Für den ersten fremden Prozess
                If p.Id <> curProcess.Id Then
                    ' Hauptfenster dieses Prozesses anzeigen
                    API.SetForegroundWindow(p.MainWindowHandle)

                    ' Wenn es minimiert ist, wiederherstellen
                    If API.IsIconic(p.MainWindowHandle) Then
                        API.ShowWindow(p.MainWindowHandle, _
                            API.ShowWindowConstants.SW_RESTORE)
                    Else
                        ' Ansonsten unverändert anzeigen
                        API.ShowWindow(p.MainWindowHandle, _
                            API.ShowWindowConstants.SW_SHOW)
                    End If

                    ' Bearbeitung beenden
                    Return
                End If
            End If
        Next
    End If

```

---

```

        End If
    Next
End If

```

```
End Sub
```

Listing 60: Mehrfachen Start eines Programms verhindern (Forts.)

Existiert der Mutex bereits, weil er zuvor von einer anderen Instanz angelegt worden ist, dann wird die weitere Bearbeitung abgebrochen. An dieser Stelle müssen Sie entscheiden, wie weiter zu verfahren ist. Eine gängig Lösung, wie sie auch im Listing implementiert ist, ist, die bereits laufende Instanz zu aktivieren und deren Hauptfenster in den Vordergrund zu holen. So wird ein Anwender, der versehentlich die Anwendung ein zweites Mal starten wollte, direkt zur laufenden Instanz geführt. Viele Windows-Programme wie z.B. Microsoft Outlook verhalten sich so.

Bevor das Hauptfenster der laufenden Instanz angezeigt werden kann, sind aber noch eine Reihe von Schritten auszuführen. Zunächst muss der richtige Prozess gefunden werden. Für verschiedene Vergleiche wird als Erstes der aktuelle Prozess bestimmt. `Process.GetCurrentProcess` gibt ein `Process`-Objekt für diesen zurück. Durch den Aufruf von `GetProcessesByName` lassen sich alle Prozesse mit demselben Namen abrufen. Wie oben bereits erwähnt, können auch völlig fremde Prozesse zufällig denselben Namen haben. Daher wird in einer Schleife über alle gefundenen Prozesse auch verglichen, ob beide Prozesse auf derselben .exe-Datei basieren. Zwar ergibt sich hier die gleiche Unsicherheit wie oben, da der andere Prozess von einer Kopie der Datei gestartet worden sein kann, jedoch würde das hier nur dazu führen, dass die andere Anwendung nicht aktiviert wird. Die mehrfache Instanzierung wurde ja bereits über den Mutex festgestellt.

In der Prozessliste befindet sich natürlich auch der aktuelle Prozess, der hier nicht betrachtet werden soll. Daher werden die IDs verglichen (`p.Id <> curProcess.Id`). Nur wenn diese unterschiedlich sind, dann handelt es sich um das `Process`-Objekt der ersten Instanz der Anwendung, die aktiviert werden soll.

Nachdem nun der Prozess gefunden worden ist, muss dessen Hauptfenster aktiviert und in den Vordergrund geholt werden. Das geschieht mit Hilfe der API-Funktion `SetForegroundWindow`. Als nächste Hürde erweist sich die Möglichkeit, dass das Fenster minimiert ist. Dann wird es leider nicht automatisch angezeigt, sondern muss über einen Aufruf von `ShowWindow` mit dem Parameter `SW_RESTORE` wiederhergestellt werden. Da dieser Aufruf ein zuvor maximiertes Fenster aber unter Umständen wieder verkleinern würde, darf er nur erfolgen, wenn das Fenster wirklich minimiert war. Das wird durch den Aufruf der API-Funktion `IsIconic` überprüft.

Der zuvor gestartete Prozess ist nun gefunden und sein Hauptfenster aktiviert worden. Mehr gibt es für den aktuellen Prozess nicht mehr zu tun, so dass er jetzt beendet werden kann, indem mit `Return` die `Sub Main` verlassen wird.

Hinweis

Der beschriebene Lösungsweg eignet sich nur, um den mehrfachen Start auf einer lokalen Maschine zu verhindern. Um in einem Netzwerk die Anzahl der laufenden Anwendungs-Instanzen zu überwachen, können Sie nicht auf Mutexe zurückgreifen, da diese nur lokal zur Verfügung stehen. Stattdessen bietet es sich an, eine Server-Applikation zu installieren, bei der sich jeder Prozess an- und abmelden muss. Nur über eine zentrale Anwendung, die die laufenden Instanzen im Netzwerk nachhält, kann z.B. die Einhaltung der lizenzierten Anzahl verifiziert werden.

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

## 54 Zentrales Exception-Handling

Es gibt Situationen, in denen man eventuell auftretende Exceptions nicht durch Try/Catch-Blöcke abfangen kann oder will. Dazu gehören z.B. automatisch beim Databinding ausgelöste Ausnahmen, die auftreten können, ohne dass sie im eigenen Code abgefangen werden könnten. Auch gibt es oft organisatorische oder andere Gründe, eine zentrale Ausnahmebehandlung vorzusehen.

Den Aufruf von `Application.Run` in einem Try-Block zu platzieren ist wenig sinnvoll. Zwar würde jede nicht behandelte Ausnahme zu einem Aufruf des zugeordneten Catch-Blockes führen, die Ausführung von `Application.Run` und somit der gesamten Fensterkonstruktion wäre aber damit beendet. Diese Variante eignet sich höchstens für die Ausgabe einer Fazit-Meldung wie »Dieser Absturz wurde Ihnen präsentiert von ...«.

Für jeden Thread können Sie jedoch eine zentrale Ausnahmebehandlung vorsehen, indem Sie einen Handler an das Ereignis `Application.ThreadException` anhängen. Eine nicht behandelte Ausnahme führt dann zum Aufruf dieses Handlers. Die Anbindung des Handlers muss allerdings erfolgen, bevor `Application.Run` aufgerufen wird. Listing 61 zeigt die übliche Vorgehensweise, bei der in `Sub Main` erst der Handler registriert und anschließend das Hauptfenster angezeigt wird.

---

```
Public Shared Sub Main()

    ' Instanz des Hauptfensters anlegen
    Dim mw As New MainWindow

    ' Zentraler Error-Handler ist Member-Funktion des Hauptfensters
    AddHandler Application.ThreadException, _
        AddressOf mw.CentralExceptionHandler

    ' Hauptfenster anzeigen
    Application.Run(mw)

End Sub
```

---

Listing 61: Binden eines Event-Handlers zum zentralen Exception-Handling

---

```
Private Sub CentralExceptionHandler(ByVal sender As Object, _
    ByVal e As System.Threading.ThreadExceptionEventArgs)

    ' Meldung zusammensetzen und ausgeben
    Dim sw As New System.io.StringWriter
    sw.Write("Hier könnte jetzt die zentrale Auswertung für ")
    sw.WriteLine("die Exception")
    sw.Write("<< ")
    sw.Write(e.Exception.Message)
    sw.WriteLine(">>")
    sw.WriteLine("vorgenommen werden")
    sw.Close()
    MessageBox.Show(sw.ToString(), "Zentrale Fehlerbehandlung")

End Sub
```

---

Listing 62: Beispielimplementierung eines zentralen Error-Handlers



Wie Sie den Handler gestalten, bleibt Ihnen überlassen. Die in Listing 62 gezeigte Ausführung dient nur zur Demonstration (siehe auch Abbildung 15). Natürlich müssen die Fehler, die hier gemeldet werden, sinnvoll bearbeitet werden. Je nach Art des Fehlers kann es durchaus sinnvoll sein, das Programm zu beenden. Der Vorteil des zentralen Handlers liegt jedoch darin, dass trotz nicht abgefangener Ausnahmen das Programm fortgesetzt werden kann, sofern der Fehler für Ihren Code nicht kritisch oder behebbar ist.

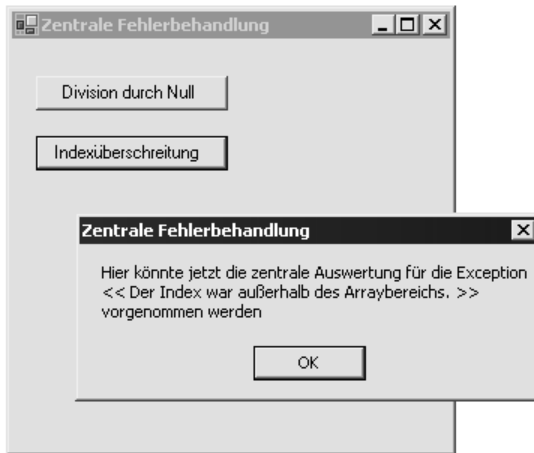


Abbildung 15: Nicht abgefangene Exceptions werden im zentralen Handler bearbeitet



# GDI+ Zeichnen

GDI+ ist so vielseitig, dass wir nicht einmal annähernd die zahllosen Möglichkeiten der Grafikausgabe beschreiben können. Daher müssen wir uns auf einige wenige Rezepte beschränken. In dieser Kategorie wird der Umgang mit Schriftzügen, Transparenz und 3D-Effekten beschrieben.

## 55 Outline-Schrift erzeugen

Schriftzüge mit großen Fonts wirken oft angenehmer, wenn sie nicht vollständig schwarz oder farbig gezeichnet werden, sondern wenn nur die Ränder schwarz nachgezeichnet werden. Die Füllung der Zeichen kann dann in Weiß oder mit einer hellen Farbe erfolgen (siehe Abbildung 16).

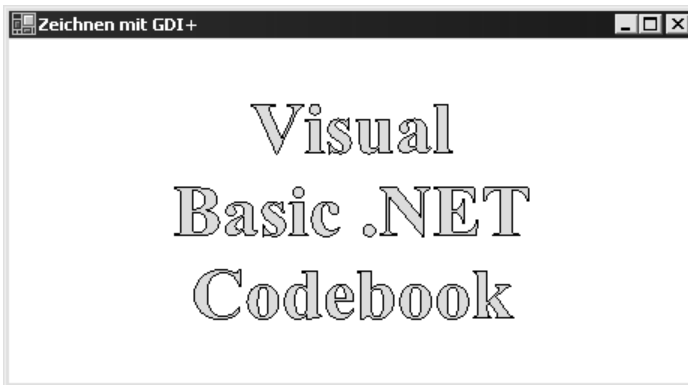


Abbildung 16: Schriftzug mit Outline-Schrift

Eine derartige Outline-Schrift steht nicht direkt zur Verfügung, sondern muss z.B. durch Pfad-Operationen erzeugt werden. Hierzu wird zunächst ein neues `GraphicsPath`-Objekt angelegt und der auszugebende Schriftzug hinzugefügt (Listing 63). Die Positionierung des Textes erfolgt in einem vorgegebenen Rechteck, so dass der Text automatisch umgebrochen werden kann. Das ebenfalls übergebene `StringFormat`-Objekt dient zur Zentrierung des Textes innerhalb des rechteckigen Bereiches.

Anschließend wird der erzeugte Pfad an der gewünschten Position (hier in der Bildmitte) zweimal gezeichnet. Mit der ersten Zeichenoperation wird die Füllung gezeichnet (Methode `FillPath`), mit der zweiten der Umriss (Methode `DrawPath`). Beachten Sie die Reihenfolge der beiden Zeichenoperationen, da bei umgekehrter Vorgehensweise Teile des Umrisses beim Zeichnen der Füllung übermalt werden können.

```
Private Sub MainWindow_Paint(ByVal sender As Object, _  
    ByVal e As System.Windows.Forms.PaintEventArgs) _  
    Handles MyBase.Paint
```

Listing 63: Erzeugen einer Outline-Schrift

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

```

' String-Format-Objekt für zentrierte Ausgabe
Dim sf As New StringFormat
sf.Alignment = StringAlignment.Center
sf.LineAlignment = StringAlignment.Center

' Pfad mit Text anlegen
Dim path As New GraphicsPath
path.AddString(Displaytext, Font.FontFamily, Font.Style, _
    Font.Size, New Rectangle(0, 0, 300, 300), sf)

' Positionierung berechnen
Dim bounds As RectangleF = path.GetBounds()
Dim x As Integer = CInt((Me.ClientSize.Width - bounds.Width) / 2 -
    bounds.Left)
Dim y As Integer = CInt((Me.ClientSize.Height - bounds.Height) / 2 -
    bounds.Top)

' Zentrieren
e.Graphics.TranslateTransform(x, y)

' Erst die Füllung der Schrift zeichnen
e.Graphics.FillPath(Brushes.Wheat, path)

' Dann die Umrahmung
e.Graphics.DrawPath(Pens.Black, path)

' Aufräumen
sf.Dispose()

```

End Sub

---

*Listing 63: Erzeugen einer Outline-Schrift (Forts.)*

## 56 Text im Kreis ausgeben und rotieren lassen

Um Text an eine beliebige Kurvenform anzupassen, gibt es verschiedene Möglichkeiten. Sie können beispielsweise den Text in Zeichen zerlegen und jedes Zeichen nach vorangegangener Koordinatentransformation einzeln ausgeben. Wir wollen Ihnen hier einen anderen recht einfachen Weg zeigen, der sich z. B. für die Ausgabe eines Textes in Kurven- oder Kreisform gut eignet.

Das Ergebnis des Beispielprogramms sehen Sie zum Teil in Abbildung 17. Zum Teil deshalb, weil Sie in der Abbildung nicht erkennen können, dass der Text kontinuierlich gedreht wird (es sei denn, Sie drehen das Buch J). Der Text wird kreisförmig angeordnet und mit einem Graustufen-Verlauf gefüllt.

Die grundlegende Idee ist, den Text in einen Pfad zu wandeln und dann die einzelnen Punkte des Pfades zu verschieben. Listing 64 zeigt das Anlegen des Pfades im Load-Ereignis des Hauptfensters. Der als konstanter String festgelegte Text wird mit den Font-Einstellungen des Fensters in einen Pfad gewandelt. Ein Pfad besteht aus Punkten, die durch Geraden oder Kurven miteinander verbunden sein können. Diese Punkte werden als Array gespeichert (*PathPoints*). Ein zweites Array (*PathTypes*) speichert den Typ der Verbindungen zwischen den Punkten.



Abbildung 17: Und es dreht sich doch! Probieren Sie es aus

Um einen Pfad an eine Kurvenform anzupassen, müssen die einzelnen Punkte transformiert werden. Die Eigenschaft `PathPoints` gibt nicht die Referenz des gespeicherten Arrays zurück, sondern legt eine Kopie an, die verändert werden darf. Die Referenz der Kopie wird in der Variablen `points` gespeichert (Listing 65). Für weitere Berechnungen werden die Höhe und die Breite des Pfades benötigt, die aus der von `GetBounds` zurückgegebenen `RectangleF`-Struktur berechnet werden.

Im konkreten Anwendungsfall muss zunächst die lineare X-Koordinate auf einen Kreis abgebildet werden (Abbildung 18). Der Radius des Kreises muss so gewählt werden, dass der Umfang des Kreises der Breite des Pfades entspricht:

---


$$R = \text{Textbreite} / 2 * \Pi$$


---

Das Koordinatensystem für die Kreistransformation bezieht sich auf das umschließende Quadrat des Kreises. Links oben ist die Koordinate (0,0), der Mittelpunkt liegt dann auf (R, R).

Damit bei der zyklischen Ausgabe das Flackern, das durch das Löschen des Hintergrundes verursacht wird, vermieden wird, wird nicht direkt auf das Fenster, sondern zunächst in ein Bitmap gezeichnet. Falls dieses noch nicht existiert, wird es jetzt angelegt. Es hat die Größe des umschließenden Quadrates.

In einer Schleife werden alle Punkte nacheinander transformiert. Zunächst wird die lineare X-Koordinate auf den Winkel  $\alpha$  abgebildet.  $\alpha$  ist der Winkel zwischen einer senkrechten Geraden, die durch den Kreismittelpunkt geht, und dem Punkt auf dem Kreis, auf den der Punkt des Pfades abgebildet werden soll. Da die Breite des Pfades auf den Umfang des Kreises abgebildet werden soll, ergibt sich für den Winkel:

---


$$\alpha = \text{Pfadpunkt.X} * 2 * \varnothing / \text{Textbreite} + \text{Startwinkel}$$


---

Bei dem Startwinkel handelt es sich um einen Vorgabewert, der zeitgesteuert verändert wird (Member-Variable `StartAngle`). Der neue Punkt soll unter diesem Winkel mit dem gleichen Abstand über dem Kreisbogen platziert werden, den er im Ausgangszustand über der Basislinie des Pfades hat (`Pfadpunkt.Y`). Mit Hilfe des Winkels und des Abstandes werden jetzt die Koordinaten des Punktes neu festgelegt:

---

```
Pfadpunkt.X = MittelpunktX + (R - Pfadpunkt.Y) * sin( $\alpha$ )
Pfadpunkt.Y = MittelpunktY + (R - Pfadpunkt.Y) * cos( $\alpha$ )
```

---

Nachdem alle Punkte transformiert worden sind, wird ein neues `GraphicsPath`-Objekt angelegt. Hierbei werden dem Konstruktor das neue `points`-Array sowie eine unveränderte Kopie des `PathTypes`-Arrays übergeben. Dieser Trick funktioniert deshalb, weil sich ja nur die Lage der Punkte, nicht jedoch die Verbindungstypen geändert haben.

In der Größe des umschließenden Rechtecks wird dann ein neues Farbverlaufs-Objekt (`LinearGradientBrush`) generiert, das für die Zeichenausgabe verwendet wird. Mit `FillPath` wird der Pfad farbig gemäß des vorgegebenen Verlaufs ausgefüllt auf das `Bitmap`-Objekt gezeichnet. Die grafische Aufarbeitung des Textes ist hiermit abgeschlossen.

Nun muss das erzeugte Image noch auf das Fenster kopiert werden. Mittels `TranslateTransform` wird die Ausgabe auf dem Fenster zentriert. `DrawImage` zeichnet das Bild und überschreibt dabei automatisch frühere Ausgaben in diesem Bereich.

---

```
Imports System.Drawing.Drawing2D
...

' Anzuzeigender Text
Protected DrawingText As String = "Sie lesen das VB.NET Codebook * "
```

*' Kontinuierlich angepasster Startwinkel*

```
Protected StartAngle As Single = 0
```

*' Bitmap-Referenz für Zeichenbereich*

```
Protected Bmp As Bitmap
```

*' Referenz des Original-Pfades (ungebogener Text)*

```
Protected Path As New GraphicsPath
```

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

*' Neu zeichnen, wenn die Fenstergröße verändert wird*

```
SetStyle(ControlStyles.ResizeRedraw, True)
```

*' Pfad mit Text anlegen*

```
Path.AddString(DrawingText, Font.FontFamily, Font.Style, _
    Font.Size, New Point(0, 0), New StringFormat)
```

```
End Sub
```

---

Listing 64: Wichtige Member-Variablen und Initialisierung des `Path`-Objektes

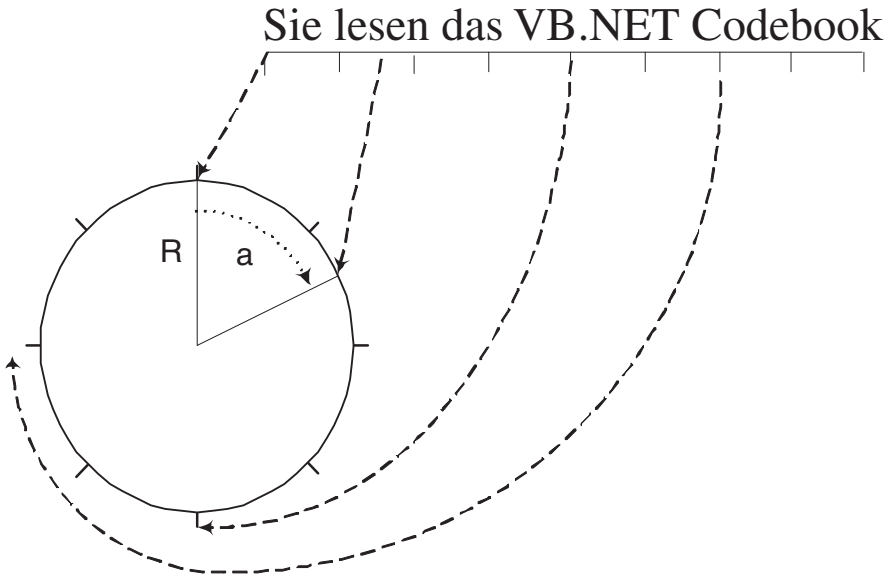


Abbildung 18: Abbildung des linearen Textes auf einen Kreis

```

Private Sub DrawText()

    ' Umschließendes Rechteck des Original-Pfads
    Dim bounds As RectangleF = Path.GetBounds()

    ' Points-Array kopieren
    Dim points() As PointF = Path.PathPoints

    ' Breite und Höhe des Originalpfads
    Dim B As Double = bounds.Width + bounds.X
    Dim H As Double = bounds.Height + bounds.Y

    ' Radius des Kreises berechnen
    Dim R As Double = B / 2 / Math.PI + H

    ' Mittelpunkt
    Dim Mx As Double = R
    Dim My As Double = R

    ' Bitmap-Objekt ggf. neu anlegen
    If Bmp Is Nothing Then Bmp = New Bitmap(CInt(R * 2), CInt(R * 2))

    ' Graphics-Objekt für Zeichnung in Buffer
    Dim g As Graphics = Graphics.FromImage(Bmp)

    ' Vollständig löschen
    g.Clear(Color.White)

```

---

```

' Punkte transformieren
For i As Integer = 0 To points.GetUpperBound(0)
    ' Winkel bestimmt sich aus StartAngle und der X-Koordinate
    ' des aktuellen Punktes
    Dim alpha As Double = points(i).X * 2 * Math.PI / B + StartAngle

    ' Der Punkt wird auf Basis des Winkels transformiert
    points(i).X = CSng(Mx + (R - points(i).Y) * Math.Sin(alpha))
    points(i).Y = CSng(My - (R - points(i).Y) * Math.Cos(alpha))

```

Next

```

' Umschließendes Rechteck des Kreises berechnen
Dim rect As New Rectangle(CInt(Mx - R), CInt(My - R), _
    CInt(2 * (R)), CInt(2 * (R)))

' Verlauf für dieses Rechteck definieren
Dim br As Brush = New LinearGradientBrush(rect, Color.LightGray, _
    Color.Black, LinearGradientMode.BackwardDiagonal)

' Neuen Pfad anlegen, der aus den transformierten Punkten besteht
Dim path2 As GraphicsPath = _
    New GraphicsPath(points, Path.PathTypes)

' Pfad in Bitmap ausgeben
g.FillPath(br, path2)

' Graphics-Objekt für Fenster holen
Dim g2 As Graphics = Me.CreateGraphics()

' Koordinatensystem so verschieben, dass der Kreis in der Mitte
' des Fensters ausgegeben wird
g2.TranslateTransform((Me.ClientSize.Width - Bmp.Width) / 2.0F, _
    (Me.ClientSize.Height - Bmp.Height) / 2.0F)

' Bitmap zeichnen
g2.DrawImage(Bmp, 0, 0)

' Aufräumen
path2.Dispose()
g2.Dispose()
br.Dispose()
g.Dispose()

```

End Sub

---

Listing 65: Transformieren und Ausgeben des Textes in Kreisform (Forts.)

Der Aufruf der in Listing 65 gedruckten Methode `DrawText` erfolgt zyklisch im Timer-Event eines Windows Timer-Controls (Listing 66). Hier wird die geschützte Member-Variable `StartAngle` kontinuierlich um einen kleinen Winkelbetrag verkleinert, so dass sich der Text gegen den Uhrzeigersinn dreht. Da die Grafikausgabe in kurzen Zeitintervallen wiederholt und so das Bild ständig aufgefrischt wird, kann auf die Implementierung des `OnPaint`-Ereignisses verzichtet werden.



```

Private Sub Timer1_Tick(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles Timer1.Tick

    ' Startwinkel dekrementieren für Drehung gegen den Uhrzeigersinn
    StartAngle -= CSng(Math.PI / 200.0)

    ' Text zeichnen
    DrawText()

End Sub

```

Listing 66: Kontinuierliche Anpassung des Startwinkels und Zeichnen des Textes

Sicher gibt es viele andere Lösungen, um den beschriebenen Effekt zu erreichen. Ein Grund für die gewählte Vorgehensweise war der Wunsch, den Farbverlauf, mit dem der Text gezeichnet wird, im Bezug auf das Fenster vorzugeben und festzuhalten. Was das Buch im Gegensatz zum lauffähigen Programm auf der CD nicht zeigen kann, ist, dass sich der Schriftzug quasi unter dem Verlauf dreht. Der helle Bereich des Schriftzugs ist immer rechts oben, unabhängig vom jeweiligen Drehwinkel.

## 57 Schriftzug mit Hintergrundbild füllen

Ein hübscher Effekt ergibt sich, wenn Sie einen Schriftzug statt mit einer festen Farbe oder einem Verlauf mit einem Hintergrundbild füllen, wie z.B. in Abbildung 19 dargestellt. Um eine möglichst große Wirkung zu erzielen, sollten Sie eine fette Schrift wählen, bei der größere zusammenhängende Partien des Bildes erkennbar sind. Schmale dünne Schriften sind hierfür eher ungeeignet. Im Beispiel wurde die Schriftart Impact, Stil Fett, Grad 100 gewählt.



Abbildung 19: Schriftzug, gefüllt mit Ausschnitten eines Fotos

Zur Lösung der Aufgabenstellung gibt es zwei prinzipielle Möglichkeiten. In der ersten (Listing 67) wird ein `GraphicsPath`-Objekt angelegt und der Text mit der eingestellten Schriftart hinzugefügt. Hierbei wird für den Text ein Rechteck vorgegeben, so dass er automatisch umgebrochen wird. Um den Text sowohl horizontal als auch vertikal in diesem Rechteck zu zentrieren, wird ein `StringFormat`-Objekt mit den passenden Eigenschaften an `AddString` übergeben.

Anschließend wird der erzeugte Pfad als Clipping-Bereich für nachfolgende Grafikausgaben eingestellt und mit `TranslateClip` auf dem Fenster zentriert. Das Ausfüllen des Schriftzugs erfolgt dann durch Zeichnen des Bildes. Nur die im Clipping-Bereich liegenden Bildteile sind sichtbar.

---

```
Imports System.Drawing.Drawing2D
...

Public Displaytext As String = "VB.NET Codebook"

Private Sub MainWindow_Paint(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.PaintEventArgs) _
    Handles MyBase.Paint

    DrawStringVersion1(e.Graphics)

End Sub

Public Sub DrawStringVersion1(ByVal g As Graphics)
    ' Ein neues Path-Objekt als Clip-Bereich
    Dim path As New GraphicsPath

    ' String-Format-Objekt für zentrierte Ausgabe
    Dim sf As New StringFormat
    sf.Alignment = StringAlignment.Center
    sf.LineAlignment = StringAlignment.Center

    ' Text in Path aufnehmen und umbrechen
    path.AddString(Displaytext, Font.FontFamily, Font.Style, _
        Font.Size, New Rectangle(0, 0, 600, 400), sf)

    ' Zentrierung berechnen
    Dim bounds As RectangleF = path.GetBounds()
    Dim x As Integer = CInt((Me.ClientSize.Width - bounds.Width) / 2 _
        - bounds.Left)
    Dim y As Integer = CInt((Me.ClientSize.Height - bounds.Height) / 2 _
        - bounds.Top)

    ' Pfad als Clipping-Bereich verwenden
    g.SetClip(path)

    ' Bereich zentrieren
    g.TranslateClip(x, y)

    ' Bitmap laden
    Dim bmp As New Bitmap(Application.StartupPath & "..\n011-01.jpg")
```

---

Listing 67: Text durch Clipping mit Grafik füllen

```

' Im Clipping-Bereich zeichnen
g.DrawImage(bmp, 0, 0)

' Aufräumen
bmp.Dispose()
path.Dispose()
sf.Dispose()

```

End Sub

*Listing 67: Text durch Clipping mit Grafik füllen (Forts.)*

Die andere Variante (siehe Listing 68) besteht darin, ein `TextureBrush`-Objekt zum Zeichnen des Textes zu verwenden. (Zum Testen muss der Aufruf im `Paint`-Ereignis abgeändert werden.) Ein solches `Brush`-Objekt zeichnet Füllungen nicht mit einer festen Farbe, sondern setzt stattdessen die jeweiligen Pixel des zugeordneten Bildes ein. Definiert wird ein `TextureBrush`-Objekt im einfachsten Fall, indem bei der Instanzierung dem Konstruktor die Referenz eines `Bitmap`-Objektes mitgegeben wird.

Anschließend kann der Text ganz gewöhnlich mit `DrawString` gezeichnet werden. Dem Aufruf werden die Referenz des `Brush`-Objektes, das Ausgabe-Rechteck für den Umbruch sowie die `StringFormat`-Informationen mitgegeben. Das Resultat ist ähnlich der Variante 1 (siehe Abbildung 20).

```

Public Sub DrawStringVersion2(ByVal g As Graphics)

' Bitmap laden
Dim bmp As New Bitmap(Application.StartupPath & "..\o060-01.jpg")

' Mit diesem Bild ein Brush-Objekt generieren
Dim tbr As New TextureBrush(bmp)

' String-Format-Objekt für zentrierte Ausgabe
Dim sf As New StringFormat
sf.Alignment = StringAlignment.Center
sf.LineAlignment = StringAlignment.Center

' Gewöhnliche Textausgabe mit erzeugtem TextureBrush
g.DrawString("Texture Brush", Me.Font, tbr, New RectangleF( _
    0, 0, ClientSize.Width, ClientSize.Height), sf)

' Aufräumen
sf.Dispose()
bmp.Dispose()
tbr.Dispose()

End Sub

```

*Listing 68: Schriftzug mit TextureBrush zeichnen*

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung  
Win-  
dows  
Forms

Controls

Propor-  
tYGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

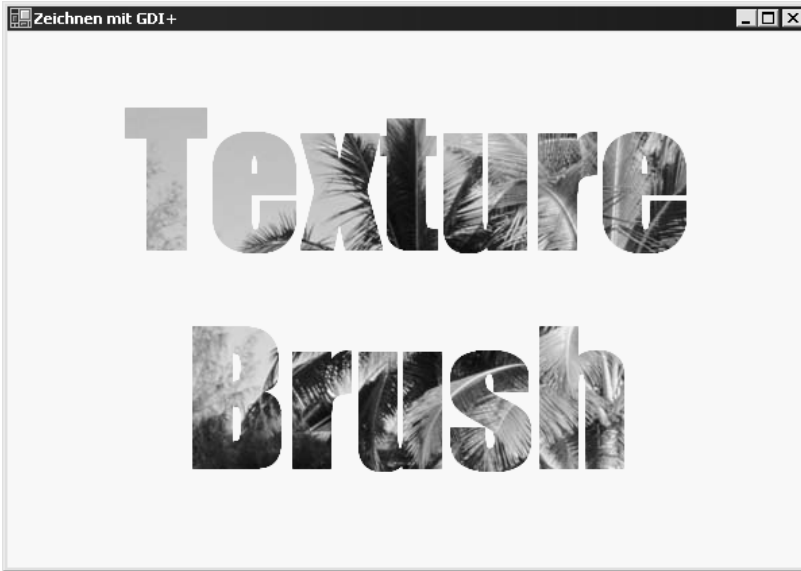


Abbildung 20: Nicht mit Clipping, sondern mit TextureBrush erzeugter Schriftzug

## 58 Transparente Schriftzüge über ein Bild zeichnen

Zur Beschriftung von Bildern, die beispielsweise als Einleitungsbild gezeigt werden sollen, kann man sich Transparenz-Effekte zu Nutze machen. Dunkle Partien z.B. lassen sich gut durch halbtransparente helle Farben beschriften (siehe Abbildung 21). Auf hellen Partien kann man dunklere halbtransparente Farben verwenden (siehe Abbildung 22).



Abbildung 21: Weißer halbtransparenter Schriftzug auf dunklem Hintergrund

Die Vorgehensweise ist recht simpel (Listing 69). Das Hintergrundbild lädt man am besten im Load-Ereignis des Fensters, so dass ggf. noch andere Anpassungen vorgenommen werden können. Im Paint-Ereignis wird dann ein `Brush`-Objekt mit der gewünschten Farbe angelegt. Je nach Hintergrundbild muss die passende Farbe durch Versuche ermittelt werden. Der Alpha-Wert für die Transparenz sollte nicht zu groß sein, damit die Struktur des Bildes nicht verloren geht.

Mit dem eingestellten `Brush`-Objekt wird dann der Text an einer beliebigen Position mit `DrawString` gezeichnet.

---

```
Imports System.Drawing.Drawing2D

...
Public Displaytext As String = "Urlaub 2003"
Public Picture As Bitmap

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

    ' Hintergrundbild laden und Fenstergröße anpassen
    Picture = New Bitmap(Application.StartupPath & "\\..\\o060-01.jpg")
    Me.BackgroundImage = Picture
    Me.ClientSize = Picture.Size

End Sub

Private Sub MainWindow_Paint(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.PaintEventArgs) _
    Handles MyBase.Paint

    ' Brush-Objekt mit halbtransparenter Farbe
    Dim br As New SolidBrush(Color.FromArgb(150, Color.White))

    ' String-Format-Objekt für zentrierte Ausgabe
    Dim sf As New StringFormat
    sf.Alignment = StringAlignment.Center
    sf.LineAlignment = StringAlignment.Center

    ' Gewöhnliche Textausgabe mit erzeugtem TextureBrush
    e.Graphics.DrawString(Displaytext, Me.Font, br, New RectangleF( _
        0, 100, ClientSize.Width, ClientSize.Height - 100), sf)

    ' Aufräumen
    sf.Dispose()
    br.Dispose()

End Sub
```

---

Listing 69: Bild mit halbtransparenter Schrift übermalen

Auf den Schwarz/Weiß-Abbildungen kommen die Effekte nicht so gut zur Geltung. Experimentieren Sie mit dem Beispielprogramm auf der CD und variieren Sie die Farb- und Transparenz-Einstellungen.

---

Basics

---

Datum/  
Zeit

---

Anwen-  
dungen

---

Zeich-  
nen

---

Bild-  
bearbei-  
tung

---

Win-  
dows  
Forms

---

Controls

---

Proper-  
tyGrid

---

Datei-  
system

---

Netz-  
werk

---

System

---

Daten-  
banken

---

XML

---

Wissen-  
schaft

---

Ver-  
schiede-  
nes

---



Abbildung 22: Schwarzer halbtransparenter Schriftzug auf hellem Hintergrund

Eine weitere Möglichkeit für einen sehr ähnlichen Effekt besteht darin, die Schrift nicht mit einem transparenten `Brush`-Objekt zu zeichnen, sondern mit einem `TextureBrush`. Dazu legt man eine Kopie des Hintergrundbildes an und hellt diese auf (z.B. durch `Matrix`-Operationen wie in Rezept 5.19 ((Weitere Bildmanipulationen mit Hilfe der `ColorMatrix`)) beschrieben). Dieses aufgehellte Hintergrundbild dient dann als Vorlage für das `TextureBrush`-Objekt. Wenn Hintergrundbild und das Image des `TextureBrush`-Objektes exakt übereinander liegen, werden überall dort, wo das Hintergrundbild übermalt wird, dessen Pixel durch die aufgehellten ersetzt. Für Abdunklungen geht man analog vor.

## 59 Blockschrift für markante Schriftzüge

Bei Buchstaben in Blockschrift wird ein leichter 3D-Effekt simuliert. Auf einer Seite (meist links unten) werden die Ränder der Zeichen dunkel nachgezeichnet (siehe Abbildung 23). Der dunkle Bereich gibt dem Schriftzug eine virtuelle Tiefe.

Die Realisierung einer solchen Blockschrift ist nicht weiter schwierig. Allein die Reihenfolge der Zeichenoperationen gilt es zu beachten. Aus dem Schriftzug wird ein neues `GraphicsPath`-Objekt generiert (Listing 70). Dann wird zunächst der dunkle Block-Bereich gezeichnet, indem der Pfad mehrfach mit schwarzer Füllung versetzt gezeichnet wird.

Anschließend wird an der Ausgangsposition der Schriftzug in der gewünschten Farbe (hier Weiß) gefüllt gezeichnet und zum Schluss die Umrahmung für den `Outline`-Effekt. Je nach Schriftart und -größe sind unterschiedliche Block-Dicken sinnvoll. Sie sollten experimentell die beste Konstellation für Ihren Zweck ermitteln.

```

Private Sub MainWindow_Paint(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.PaintEventArgs) _
    Handles MyBase.Paint

    ' String-Format-Objekt für zentrierte Ausgabe
    Dim sf As New StringFormat
    sf.Alignment = StringAlignment.Center
    sf.LineAlignment = StringAlignment.Center

    ' Pfad mit Text anlegen
    Dim path As New GraphicsPath
    path.AddString(Displaytext, Font.FontFamily, Font.Style, _
        Font.Size, New Point(0, 0), sf)

    ' Positionierung berechnen
    Dim bounds As RectangleF = path.GetBounds()
    Dim x As Integer = CInt((Me.ClientSize.Width - bounds.Width) / 2 _
        - bounds.Left)
    Dim y As Integer = CInt((Me.ClientSize.Height - bounds.Height) / _
        2 - bounds.Top)

    Dim g As Graphics = e.Graphics

    ' Zentrieren
    g.TranslateTransform(x, y)

    ' Erst den Block zeichnen
    Dim steps As Integer = 5
    For i As Integer = 1 To steps
        ' um 1 Pixel nach links unten verschieben
        g.TranslateTransform(-1, 1)

        ' Pfad komplett schwarz ausfüllen
        g.FillPath(Brushes.Black, path)
    Next

    ' Zurück zum Ausgangspunkt
    g.TranslateTransform(steps, -steps)

    ' Erst die Füllung der Schrift zeichnen
    g.FillPath(Brushes.White, path)

    ' Dann die Umrahmung
    g.DrawPath(Pens.Black, path)

    ' Aufräumen
    path.Dispose()
    sf.Dispose()

End Sub

```

Listing 70: Erzeugung von Blockschrift durch mehrfaches versetztes Zeichnen des Textes

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes



Abbildung 23: Schriftzug in Blockschrift

## 60 Text mit versetztem Schatten zeichnen

Nach einem ähnlichen Muster wie im vorangegangenen Rezept lässt sich auch ein Schatten zeichnen, der vom Schriftzug etwas abgesetzt ist (siehe Abbildung 23). Bei strukturierten Hintergründen lässt sich die Wirkung noch verbessern, wenn der Schatten halbtransparent ist und die Struktur des Hintergrundes nicht völlig überdeckt. Für den Betrachter sieht es dann aus, als würde der Schriftzug über dem Hintergrund schweben.

Gezeichnet wird der Schriftzug wieder mit Hilfe eines `GraphicsPath`-Objektes. Vor dem Zeichnen der eigentlichen Outline-Schrift wird der Schatten gezeichnet. Mit welchem Versatz das geschehen muss, hängt von der Schriftgröße und -art ab und sollte experimentell ermittelt werden. Im Beispiel wurde ein Versatz von 20 Pixel eingestellt.

Auch die Farbe des Schattens können Sie variieren. Sie sollte relativ dunkel gewählt werden, mit niedrigem Alpha-Wert für eine möglichst große Transparenz. Ein Schatten muss nicht unbedingt schwarz oder grau sein (ja, hier im Buch leider doch!). Auch ein leichter Farbstich kann sehr elegant wirken.



Abbildung 24: Ein einfacher versetzter halbtransparenter Schatten



```

Private Sub MainWindow_Paint(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.PaintEventArgs) _
    Handles MyBase.Paint

    ' String-Format-Objekt für zentrierte Ausgabe
    Dim sf As New StringFormat
    sf.Alignment = StringAlignment.Center
    sf.LineAlignment = StringAlignment.Center

    ' Pfad mit Text anlegen
    Dim path As New GraphicsPath
    path.AddString(Displaytext, Font.FontFamily, Font.Style, _
        Font.Size, New Rectangle(0, 0, ClientSize.Width, _
            ClientSize.Height), sf)

    ' Positionierung berechnen
    Dim bounds As RectangleF = path.GetBounds()
    Dim x As Integer = CInt((Me.ClientSize.Width - bounds.Width) / 2 _
        - bounds.Left)
    Dim y As Integer = CInt((Me.ClientSize.Height - bounds.Height) / _
        2 - bounds.Top)

    ' Zielposition für Textausgabe
    e.Graphics.TranslateTransform(x, y)

    ' Versatz für Schatten
    Dim dx As Integer = -20
    Dim dy As Integer = 20

    ' Verschiebung für Schatten
    e.Graphics.TranslateTransform(dx, dy)

    ' Halbtransparente Farbe für Schatten verwenden
    Dim br As New SolidBrush(Color.FromArgb(50, 40, 40, 40))

    ' Schattenschriftzug zeichnen
    e.Graphics.FillPath(br, path)

    ' Zurück zum Ausgangspunkt
    e.Graphics.TranslateTransform(-dx, -dy)

    ' Erst die Füllung der Schrift zeichnen
    e.Graphics.FillPath(Brushes.White, path)

    ' Dann die Umrahmung
    e.Graphics.DrawPath(Pens.Black, path)

    ' Aufräumen
    br.Dispose()
    sf.Dispose()

```

Listing 71: Outlineschrift mit halbtransparenten Schatten erzeugen

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Proper-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

---

```
path.Dispose()
```

```
End Sub
```

---

*Listing 71: Outlineschrift mit halbtransparenten Schatten erzeugen (Forts.)*

## 61 Schriftzug perspektivisch verzerren

Die Klasse `GraphicsPath` bietet für einfache perspektivische Verzerrungen die Methode `Warp` an. Mit ihr lassen sich die Punkte eines Pfades gemäß eines als Punkte-Array vorgegebenen Trapezes transformieren. So lassen sich schnell perspektivische Effekte wie in Abbildung 24 zu sehen realisieren. Durch geschickte Farbgebungen, z.B. wie in der Abbildung mit halbtransparenten Verläufen, lässt sich der Effekt noch verstärken.

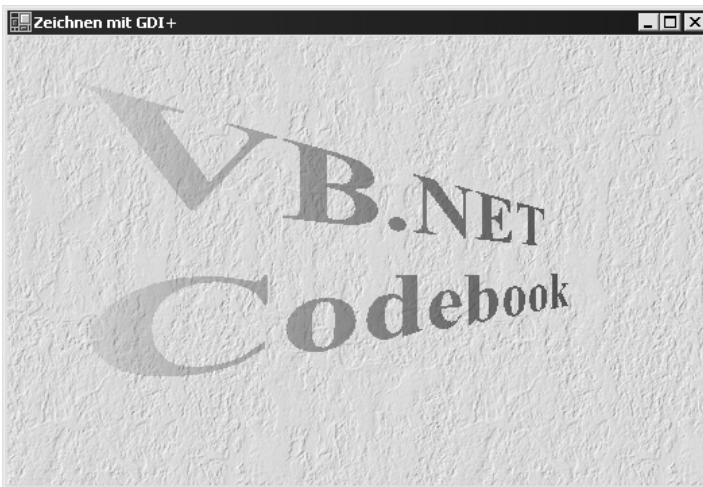


*Abbildung 25: Mit Warp verzerrter Schriftzug*

Der Schriftzug wird wieder mit der benötigten Größe und Formatierung in einem `GraphicsPath`-Objekt angelegt. Mit `warpPoints` wird ein Array mit vier Punkten definiert, das das Trapez beschreibt. Dieses Trapez gibt in Verbindung mit einem Rechteck die durchzuführende Transformation vor. Dabei werden die Punkte des Rechtecks auf die des Trapezes abgebildet. Die Methode `Warp` nimmt Trapez und Rechteck als Parameter entgegen und transformiert alle Punkte des Pfades.

Zum Zeichnen des Pfades wird ein Verlauf (`LinearGradientBrush`) definiert, der mit halbtransparenten Farben arbeitet. So kann der Hintergrund leicht durchscheinen. Die helle Startfarbe wird für die visuell nahen Bereiche benutzt, während die dunkle Endfarbe des Verlaufs den Schriftzug in der Ferne verschwinden lässt.

---

```
Imports System.Drawing.Drawing2D
...

Public Displaytext As String = "VB.NET Codebook"
```

---

*Listing 72: Mit einer Warp-Transformation einen Schriftzug verzerren*

```

Private Sub MainWindow_Paint(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.PaintEventArgs) _
    Handles MyBase.Paint

    ' String-Format-Objekt für zentrierte Ausgabe
    Dim sf As New StringFormat
    sf.Alignment = StringAlignment.Near
    sf.LineAlignment = StringAlignment.Center

    ' Pfad mit Text anlegen
    Dim path As New GraphicsPath
    path.AddString(Displaytext, Font.FontFamily, Font.Style, _
        Font.Size, New Rectangle(0, 0, ClientSize.Width, _
            ClientSize.Height), sf)

    ' Verzerrungspunkte
    Dim warpPoints(3) As PointF
    warpPoints(0).X = 0.1F * ClientSize.Width
    warpPoints(0).Y = 0.1F * ClientSize.Height
    warpPoints(1).X = 0.8F * ClientSize.Width
    warpPoints(1).Y = 0.4F * ClientSize.Height
    warpPoints(2).X = 0.1F * ClientSize.Width
    warpPoints(2).Y = 0.8F * ClientSize.Height
    warpPoints(3).X = 0.8F * ClientSize.Width
    warpPoints(3).Y = 0.6F * ClientSize.Height

    ' Transformation auf Pfad anwenden
    path.Warp(warpPoints, path.GetBounds())

    ' Verlauf definieren
    Dim rect As New Rectangle(0, 0, ClientSize.Width, _
        ClientSize.Height)
    Dim startColor As Color = Color.FromArgb(50, Color.White)
    Dim endColor As Color = Color.FromArgb(200, Color.Black)
    Dim br As Brush = New LinearGradientBrush(rect, startColor, _
        endColor, LinearGradientMode.Horizontal)

    ' Füllung der Schrift zeichnen
    e.Graphics.FillPath(br, path)

    ' Aufräumen
    sf.Dispose()
    br.Dispose()
    path.Dispose()

```

End Sub

Listing 72: Mit einer Warp-Transformation einen Schriftzug verzerren (Forts.)

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

## 62 Font-Metrics zur exakten Positionierung von Schriftzügen ermitteln

Für viele Effekte ist es wichtig, Schriftzüge exakt positionieren zu können. Hierzu sollten Sie ein paar Methoden kennen, mit deren Hilfe Sie die wesentlichen Maße berechnen können. In vielen Büchern und auch der MSDN-Hilfe finden Sie entsprechende Zeichnungen, die die Maße erläutern. Wir programmieren uns die Zeichnung einfach selbst J.

Abbildung 26 zeigt das Ergebnis. Über die Schaltfläche können Sie beliebige Schriftarten auswählen und den Verlauf der Linien für Ascent, Descent und Height begutachten. Mithilfe des Beispielprogramms können Sie nachvollziehen, wie die Maße für die Positionierung der Linien berechnet werden. Wir wollen aber nicht zu tief in die Typografie einsteigen und beschränken uns daher auf die Erklärung der drei genannten Größen.

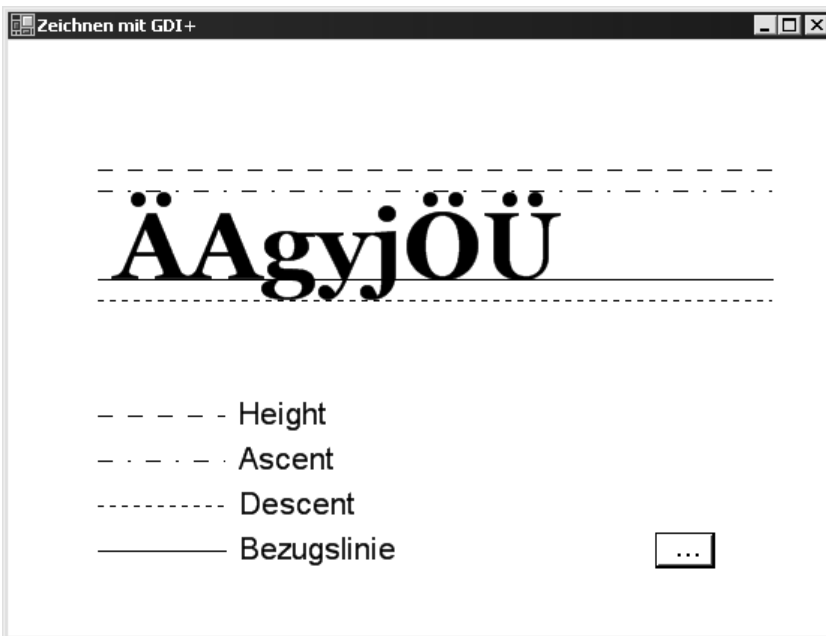


Abbildung 26: Wichtige Maße für die Darstellung von Schriften

Die Werte für Ascent und Descent lassen sich mithilfe der Methoden `GetCellAscent` bzw. `GetCellDescent` der Klasse `FontFamily` abrufen, allerdings nicht in Pixel, sondern in *Entwurfseinheiten*. Diese Entwurfseinheiten sind eine typografische Maßeinheit, die für uns nicht weiter von Belang ist. Die Bezugsgröße (`LineSpacing`), die via `GetLineSpacing` abgefragt werden kann, ist ebenfalls in dieser Einheit definiert. Relevant ist lediglich das Verhältnis von Ascent bzw. Descent zu `LineSpacing`. Dieses Verhältnis, multipliziert mit der Höhe der Schrift in Pixel, die sich durch den Aufruf von `GetHeight` erfragen lässt, ergibt die korrekten Werte für Ascent und Descent in Pixel, also den benötigten Ausgabeeinheiten.

Um etwas Rand für die Zeichnung freizuhalten, wird das Koordinatensystem mit `TranslateTransform` etwas nach rechts unten verschoben. Der Schriftzug wird gezeichnet, wobei die linke obere Ecke auf den Ursprung des Koordinatensystems fällt.

Mit der anschließenden Kombination aus den Transformationen `TranslateTransform` und `ScaleTransform` wird das Koordinatensystem auf die Basislinie des Schriftzugs verschoben und die Richtung der Y-Achse umgekehrt. Die X-Achse ist somit die Bezugslinie des Schriftzugs und die positive Y-Achse zeigt nach oben.

Mit unterschiedlichen Strichmustern werden nun waagerechte Linien in Höhe der Werte `Ascent`, `Height`, `Descent` sowie der Basislinie gezeichnet. Beobachten Sie, wie sich diese Linien verschieben, wenn Sie andere Schriftarten, -stile und -größen einstellen.

Der Rest des Codes dient zum Zeichnen der Legende. Hier wird jeweils die entsprechende Linie gezeichnet und der Text mithilfe eines Rechtecks vertikal zentriert.

```
Private Sub MainWindow_Paint(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.PaintEventArgs) _
    Handles MyBase.Paint

    ' Font-Metrics berechnen

    ' Ascent
    Dim fontAscent As Single = Font.GetHeight(e.Graphics) * _
        Font.FontFamily.GetCellAscent(Font.Style) / _
        Font.FontFamily.GetLineSpacing(Font.Style)

    ' Descent
    Dim fontDescent As Single = Font.GetHeight(e.Graphics) * _
        Font.FontFamily.GetCellDescent(Font.Style) / _
        Font.FontFamily.GetLineSpacing(Font.Style)

    ' Höhe (Height)
    Dim h As Single = Font.GetHeight(e.Graphics)

    ' Position für Schriftzug vorgeben
    e.Graphics.TranslateTransform(60, 100)

    ' Schriftzug zeichnen
    e.Graphics.DrawString(Displaytext, Font, Brushes.Black, 0, 0)

    ' Neues Koordinatensystem auf Basislinie setzen
    e.Graphics.TranslateTransform(0, h - fontDescent)
    e.Graphics.ScaleTransform(1, -1)

    ' Bezugslinie zeichnen
    e.Graphics.DrawLine(Pens.Black, 0, 0, 450, 0)

    Dim y As Single
    Dim p As New Pen(Color.Black)

    ' Linie für Height zeichnen
    y = h
    p.DashStyle = DashStyle.Custom
    p.DashPattern = New Single() {10, 10, 10, 10}
    e.Graphics.DrawLine(p, 0, y, 450, y)
```

Listing 73: Programm zur Demonstration der Fontmetrics

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

```

' Linie für Ascent zeichnen
y = fontAscent
p.DashPattern = New Single() {10, 10, 2, 10}
e.Graphics.DrawLine(p, 0, y, 450, y)

' Linie für Descent zeichnen
y = -fontDescent
p.DashPattern = New Single() {4, 4, 4, 4}
e.Graphics.DrawLine(p, 0, y, 450, y)

' Transformation für Legende
e.Graphics.ResetTransform()
e.Graphics.TranslateTransform(60, 250)

Dim sf As New StringFormat

sf.Alignment = StringAlignment.Near
sf.LineAlignment = StringAlignment.Center

' Font für Legende
Dim ft As New Font("Arial", 16)
Dim rf As New RectangleF(90, -20, 200, 40)

' Legende für Height
p.DashPattern = New Single() {10, 10, 10, 10}
e.Graphics.DrawLine(p, 0, 0, 85, 0)
e.Graphics.DrawString("Height", ft, Brushes.Black, rf, sf)

e.Graphics.TranslateTransform(0, 30)

' Legende für Ascent
p.DashPattern = New Single() {10, 10, 2, 10}
e.Graphics.DrawLine(p, 0, 0, 85, 0)
e.Graphics.DrawString("Ascent", ft, Brushes.Black, rf, sf)

e.Graphics.TranslateTransform(0, 30)

' Legende für Descent
p.DashPattern = New Single() {4, 4, 4, 4}
e.Graphics.DrawLine(p, 0, 0, 85, 0)
e.Graphics.DrawString("Descent", ft, Brushes.Black, rf, sf)
e.Graphics.TranslateTransform(0, 30)

' Legende für Bezugslinie
p.DashStyle = DashStyle.Solid
e.Graphics.DrawLine(p, 0, 0, 85, 0)
e.Graphics.DrawString("Bezugslinie", ft, Brushes.Black, rf, sf)

' Aufräumen
sf.Dispose()

```

---

```
p.Dispose()  
ft.Dispose()
```

End Sub

Listing 73: Programm zur Demonstration der Fontmetrics (Forts.)

## 63 Schatten durch Matrix-Operationen erzeugen

Mithilfe einiger simpler Matrix-Operationen lassen sich effektiv Schatten in beliebige Richtungen zeichnen (siehe Abbildung 27). Um an die vorherigen Beispiele anzuknüpfen, zeigen wir in diesem Rezept, wie Sie Schatten für Schriftzüge erstellen können. Das Verfahren beschränkt sich aber nicht auf Schriften, sondern lässt sich für nahezu alle Zeichenausgaben verwenden.

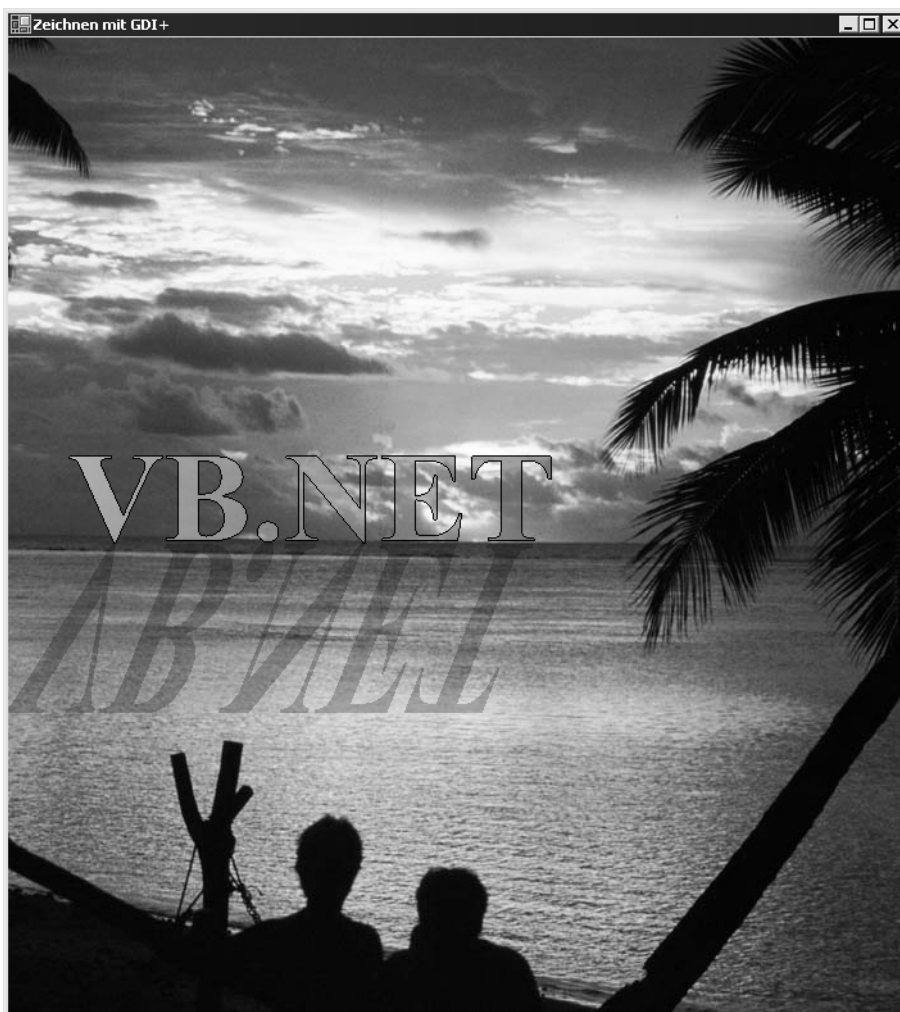


Abbildung 27: Mit Matrix-Operationen ist das Zeichnen von Schatten ganz einfach

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

Wenn, wie in der Abbildung, ein Schatten direkt an einen Schriftzug anschließen soll, ist es wichtig, dass eine exakte Positionierung vorgenommen wird. Im Beispielcode wird der Schriftzug in einem `GraphicsPath`-Objekt abgelegt. Damit er unten genau bündig im umschließenden Rechteck des Pfades abschließt, wird die Differenz aus den Fontmetrics-Werten `Ascend` und `Descend` (vgl. Rezept 4.8) zur Verschiebung in Y-Richtung eingesetzt.

Vor dem Zeichnen wird das Koordinatensystem so eingestellt, dass die Ausgabe des Pfades ohne weitere Verschiebung erfolgen kann. Bevor der Haupt-Schriftzug gezeichnet wird, muss erst der Schatten gezeichnet werden. Bei der Konstellation wie in Abbildung 27 spielt die Reihenfolge zwar keine Rolle, da sich Schriftzug und Schatten nicht überschneiden, wenn Sie jedoch den Schatten virtuell nach hinten (also auf der Zeichenfläche nach oben) fallen lassen wollen, dann ist die richtige Reihenfolge entscheidend.

Für die Transformation des Schattens wird ein neues Matrix-Objekt angelegt. Der Standard-Konstruktor erzeugt automatisch eine Identitätsmatrix (1 auf der Hauptdiagonalen, Rest 0). Um den Schriftzug vergrößert nach unten zu zeichnen, wird die Y-Achse mit dem Faktor -2 multipliziert. Die X-Achse bleibt unverändert. Diese Skalierung erfolgt durch:

---

```
mtx.Scale(1, -2)
```

---

Anschließend wird eine leichte Scherung nach links eingestellt. Eine Scherung in Y-Richtung erfolgt nicht. Diese Operation geschieht durch:

---

```
mtx.Shear(0.8, 0)
```

---

Die so vorbereitete Matrix kann nun für die Zeichenausgabe verwendet werden. `MultiplyTransform` ruft eine Matrizen-Multiplikation auf und verändert so die Transformationsmatrix des `Graphics`-Objektes. Alle nachfolgenden Zeichenoperationen erfolgen mit dem neuen Koordinatensystem.

Der Schatten kann jetzt wie gewöhnlich durch Auswahl eines `Brush`-Objektes (hier ein dunkler halbtransparenter `SolidBrush`) und Aufruf von `FillPath` gezeichnet werden. Anschließend wird ein Verlauf für die Outline-Schrift des Haupt-Schriftzuges erstellt.

Bevor dieser gezeichnet werden kann, muss die Transformation wieder rückgängig gemacht werden. Im Beispiel erfolgt das durch die Multiplikation mit der invertierten Matrix. Danach steht das Koordinatensystem wieder genauso wie vor der Transformation und der Text kann mit `FillPath` und `DrawPath` ausgegeben werden.

---

```
Imports System.Drawing.Drawing2D
...

Private Sub MainWindow_Paint(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.PaintEventArgs) _
    Handles MyBase.Paint

    Dim sf As New StringFormat

    ' Font-Metrics berechnen
```

---



```

Dim fontAscent As Single = Font.GetHeight(e.Graphics) * _
    CSng(Font.FontFamily.GetCellAscent(Font.Style) / _
        Font.FontFamily.GetLineSpacing(Font.Style))

Dim fontDescent As Single = Font.GetHeight(e.Graphics) * _
    CSng(Font.FontFamily.GetCellDescent(Font.Style) / _
        Font.FontFamily.GetLineSpacing(Font.Style))

' Position für Schriftzug vorgeben
e.Graphics.TranslateTransform(30, 393)

' GraphicPath mit Schriftzug anlegen
Dim path As New GraphicsPath
path.AddString(Displaytext, Font.FontFamily, Font.Style, _
    Font.Size, New PointF(0, fontDescent - fontAscent), sf)

' Matrix für Transformation des Schattens
Dim mtx As New Matrix

' Y-Skalierung nach unten (holt Schatten nach vorne)
mtx.Scale(1, -2)

' Scherung nach links
mtx.Shear(0.8, 0)

' Matrix anwenden
e.Graphics.MultiplyTransform(mtx)

' Schatten transparent zeichnen
Dim br As Brush = New SolidBrush(Color.FromArgb(60, Color.Black))
e.Graphics.FillPath(br, path)

' Verlauf für Normalschrift definieren
Dim startColor As Color = Color.FromArgb(150, Color.White)
Dim endColor As Color = Color.FromArgb(150, Color.Black)
Dim lgb As New LinearGradientBrush(New Point(0, 0), _
    New Point(600, 0), startColor, endColor)

' Transformationen rückgängig machen
mtx.Invert()
e.Graphics.MultiplyTransform(mtx)

' Outlineschrift mit halbtransparenter Füllung zeichnen
e.Graphics.FillPath(lgb, path)
e.Graphics.DrawPath(Pens.Black, path)

' Aufräumen
sf.Dispose()
br.Dispose()
lgb.Dispose()
path.Dispose()

```

Listing 74: Schriftzug mit Schatten über ein Bild zeichnen (Forts.)

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Proper-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

---

```
mtx.Dispose()
```

```
End Sub
```

---

*Listing 74: Schriftzug mit Schatten über ein Bild zeichnen (Forts.)*

Wie Sie einen Schatten gestalten, also in welche Richtung, mit welcher Neigung, welcher Farbe, welcher Transparenz usw., hängt sehr stark von den Umständen ab, die durch das Bild vorgegeben werden und ist ein weiteres Feld für Experimente. Bei Schriftzügen müssen Sie auch besonders auf Unterlängen achten. Im Beispiel wurde der Schatten an der Basislinie der Schrift angesetzt. Wenn Zeichen mit Unterlängen vorhanden sind, dann muss der Schatten entsprechend tiefer angesetzt werden.

## 64 Rechtecke mit abgerundeten Ecken zeichnen

Rechtecke zu zeichnen ist ja kein Problem. Aber für Rechtecke mit abgerundeten Ecken hat GDI+ keine passende Methode parat. Diese lässt sich aber mit Hilfe von Pfaden schnell zusammensetzen.

Die Klasse `RoundedRectangle` (Listing 75) kapselt die benötigte Funktionalität. Der Konstruktor nimmt eine `RectangleF`-Struktur für die Größe des Rechtecks und den Radius für die Abrundungen entgegen und speichert sie in öffentlichen Member-Variablen. Zwei öffentliche Methoden, `Draw` und `Fill`, können aufgerufen werden, um entweder die Umrisse oder die Füllung der Figur zu zeichnen. Beide rufen die Methode `GetPath` auf, die einen Pfad zurückgibt, der zum Zeichnen der Figur verwendet wird.

`GetPath` setzt die Linien segmentweise zusammen. Abwechselnd werden Geradenstücke und 90°-Bögen gezeichnet. Die acht Segmente werden abschließend durch den Aufruf von `CloseFigure` zu einer geschlossenen Figur verbunden, die als Umriß oder ausgefüllt gezeichnet werden kann.

---

```
Imports System.Drawing.Drawing2D

Public Class RoundedRectangle

    ' Umschließendes Rechteck und Radius für Abrundungen
    Public Rect As RectangleF
    Public Radius As Single

    ' Konstruktor zur schnellen Initialisierung
    Public Sub New(ByVal rect As RectangleF, ByVal radius As Single)
        Me.Rect = rect
        Me.Radius = radius
    End Sub

    ' Umrisse des Rechtecks zeichnen
    Public Sub Draw(ByVal g As Graphics, ByVal pen As Pen)

        ' Pfad ermitteln
        Dim path As GraphicsPath = GetPath()
```

---

*Listing 75: Klasse `RoundedRectangle` zum Zeichnen von Rechtecken mit abgerundeten Ecken*

```

' Zeichnen
g.DrawPath(pen, path)

' Aufräumen
path.Dispose()

End Sub

' Füllung zeichnen
Public Sub Fill(ByVal g As Graphics, ByVal brush As Brush)

' Pfad ermitteln
Dim path As GraphicsPath = GetPath()

' Füllung zeichnen
g.FillPath(brush, path)

' Aufräumen
path.Dispose()

End Sub

' Pfad für Rechtecke berechnen
Public Function GetPath() As GraphicsPath

' Hilfsvariablen
Dim path As New GraphicsPath
Dim x As Single = Rect.X
Dim y As Single = Rect.Y
Dim w As Single = Rect.Width
Dim h As Single = Rect.Height
Dim r As Single = Radius
Dim d As Single = 2 * Radius

' 4 Linien + 4 Bögen
path.AddLine(x + r, y, x + w - d, y)
path.AddArc(x + w - d, y, d, d, 270, 90)
path.AddLine(x + w, y + r, x + w, y + h - d)
path.AddArc(x + w - d, y + h - d, d, d, 0, 90)
path.AddLine(x + w - d, y + h, x + r, y + h)
path.AddArc(x, y + h - d, d, d, 90, 90)
path.AddLine(x, y + h - d, x, y + r)
path.AddArc(x, y, d, d, 180, 90)

' Abschließen
path.CloseFigure()

Return path
End Function

End Class

```

Listing 75: Klasse RoundedRectangle zum Zeichnen von Rechtecken mit abgerundeten Ecken (Forts.)

Basics

Datum/  
ZeitAnwen-  
dungen**Zeich-  
nen**Bild-  
bearbei-  
tung  
Win-  
dows  
Forms

Controls

Proper-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

Ein Codefragment im Paint-Ereignis eines Fensters wie das folgende:

---

```
Dim lgb As LinearGradientBrush

Dim r As New RectangleF(50, 50, 200, 100)
Dim rr As New RoundedRectangle(r, 20)
lgb = New LinearGradientBrush(r, Color.White, Color.Black, 40)
rr.Fill(e.Graphics, lgb)
rr.Draw(e.Graphics, New Pen(Color.Black, 0))
lgb.Dispose()
```

---

nutzt die Klasse `RoundedRectangle` und zeichnet ein Rechteck mit abgerundeten Ecken und einer Verlaufs-Füllung (vergleiche Abbildung 28).

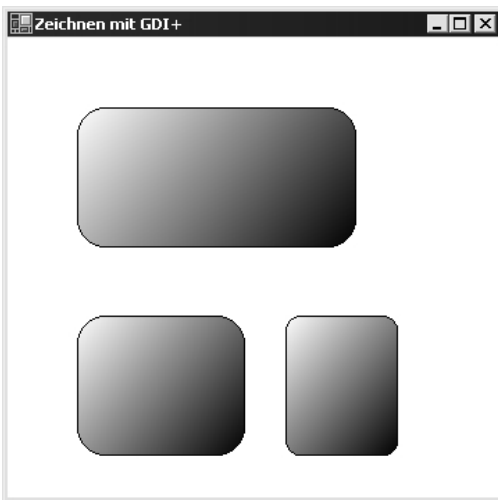


Abbildung 28: *Rounded Rectangles selbst erzeugt*

## 65 3D-Schriften erzeugen

Hervorgehobene (embossed) und vertiefte (chiseled) Schriften lassen sich durch mehrfaches versetztes Zeichnen mit unterschiedlichen Farben simulieren. Die in Abbildung 29 dargestellten Schriftzüge wurden durch je drei Zeichenoperationen generiert.

Der Trick besteht lediglich darin, mit einem leichten Versatz (im Beispiel wurde ein Versatz von 2 Pixel benutzt) den gleichen Text mit Schatten-, Aufhell- und Normalfarbe zu zeichnen. Welche Farben verwendet werden müssen, ist vom Hintergrund abhängig und muss ggf. experimentell ermittelt werden.

Für die Anhebung wird mit einer hellen Farbe (hier Weiß) begonnen. Diese Farbe muss heller sein als der Hintergrund und stellt die beleuchtete Kante dar. Danach wird die abgeschattete Seite mit einem dunklen Grauton gezeichnet (um den doppelten Versatz nach rechts unten verschoben). Diese Farbe muss dunkler sein als der Hintergrund. Abschließend wird der Text mit Schwarz nachgezeichnet.



Abbildung 29: Hervorgehobene und vertiefte Schrift

Um einen Vertiefungseffekt zu erzielen, werden die beiden ersten Zeichenoperationen vertauscht. Dadurch befinden sich die abgeschattete Seite links oben und der beleuchtete Rand rechts unten. Mit der dritten Zeichenoperation wird die Schrift eingefärbt. Hier empfiehlt sich ein etwas hellerer Farbton als der des Hintergrundes. Experimentieren Sie selbst durch Änderung der Farben und der anderen Parameter.

```
Private Sub MainWindow_Paint(ByVal sender As Object, ...
```

```
    Dim fnt As New Font("Times", 60)
    Dim text As String = "3D - Schrift"
    Dim offset As Integer = 2
    Dim g As Graphics = e.Graphics

    g.TranslateTransform(50, 50)

    ' Anhebung zeichnen
    g.DrawString(text, fnt, Brushes.White, 0, 0)

    ' Absenkung zeichnen
    g.TranslateTransform(offset * 2, offset * 2)
    g.DrawString(text, fnt, Brushes.DarkGray, 0, 0)

    ' Normalbereich zeichnen
    g.TranslateTransform(-offset, -offset)
    g.DrawString(text, fnt, Brushes.Black, 0, 0)

    g.TranslateTransform(-offset, 90)

    ' Absenkung zeichnen
    g.DrawString(text, fnt, Brushes.DarkGray, 0, 0)
    g.TranslateTransform(offset * 2, offset * 2)
```

Listing 76: 3D-Effekte durch versetztes Zeichnen

Basics

Datum/  
Zeit

Anwen-  
dungen

**Zeich-  
nen**

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Propor-  
tityGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

```
' Anhebung zeichnen  
g.DrawString(text, fnt, Brushes.White, 0, 0)  
  
' Normalbereich zeichnen  
g.TranslateTransform(-offset, -offset)  
g.DrawString(text, fnt, Brushes.WhiteSmoke, 0, 0)
```

End Sub

*Listing 76: 3D-Effekte durch versetztes Zeichnen (Forts.)*

## 66 3D- und Beleuchtungseffekte auf strukturierten Hintergrundbildern

Nach einem ähnlichen Verfahren wie im vorangegangenen Rezept lassen sich 3D-Effekte auch auf Hintergrundbildern realisieren. Dabei besteht keine Beschränkung auf die Erzeugung von 3D-Schriftzügen, wie Sie sie in Abbildung 30 und Abbildung 31 sehen können, sondern Sie können mit dem vorgestellten Verfahren nahezu beliebige Formen hervorheben oder vertiefen. Beachten Sie in den Abbildungen auch den zusätzlichen Helligkeitsverlauf des Hintergrundbildes von links oben (heller) nach rechts unten (dunkler). Auch dieser wird per Code auf einem ansonsten gleichmäßig hellen Hintergrundbild erzeugt. Der Helligkeitsverlauf verstärkt die 3D-Darstellung, da er zusätzlich andeutet, dass das Bild von einer oben links befindlichen Lichtquelle beleuchtet wird.



Abbildung 30: Relief-Schrift mit Helligkeitsverlauf auf strukturiertem Hintergrund



Abbildung 31: Vertieftes Relief auf strukturiertem Untergrund

Beginnen wir mit dem Helligkeitsverlauf. Dieser wird simuliert, indem das gesamte Fenster durch einen Aufruf von `FillRectangle` mit einem `LinearGradientBrush`-Objekt übermalt wird. Anfangs- und Endfarbe dieses Verlaufs müssen auf den Hintergrund abgestimmt werden. Der Alpha-Wert sollte nicht zu hoch gewählt werden, da sonst der Kontrast des Hintergrundbildes stark verkleinert wird. Transparenzwerte von ca. 100 haben sich als praktikabel herausgestellt.

Die verschiedenen Zeichenoperationen arbeiten mit einem Pfad, der mehrmals gezeichnet werden muss. Der Text wird daher einem `GraphicsPath`-Objekt hinzugefügt. Anschließend werden zwei `Brush`-Objekte angelegt, eines zum Aufhellen und eines zum Abdunkeln des Hintergrundes. Auch hier gilt wieder, dass Farbe, Helligkeit und Transparenz experimentell für verschiedene Hintergründe ermittelt werden müssen. Die Transparenz ist hier nicht ganz so kritisch, da die übermalten Flächen relativ schmal sind und eine stärkere Verdeckung der Hintergrundstruktur nicht so stark ins Gewicht fällt.

Abhängig davon, ob der Pfad erhöht oder vertieft dargestellt werden soll, werden den Referenzvariablen `br1` und `br2` die beiden erzeugten `Brush`-Objekte zugewiesen. Das erspart spätere Abfragen, wenn noch mehrere Objekte gleichermaßen gezeichnet werden müssen.

Da der Hauptteil des Schriftzuges, also der mittlere Bereich, unverändert mit der Hintergrundstruktur dargestellt werden muss, kann nicht, wie im vorherigen Rezept, der Pfad einfach mehrmals vollständig auf das Fenster gezeichnet werden. Anderenfalls würde der gesamte Schriftzug in einem kontrastarmen Grau erscheinen.

Stattdessen werden nur die Kanten gezeichnet. Das wird erreicht, indem der Hauptteil durch Clipping vom Zeichenbereich ausgenommen wird. Hier wird der `Clipping`-Bereich angelegt, indem der Methode `SetClip` der Pfad und der Modus `Exclude` als Parameter übergeben werden. Nachfolgende Zeichenoperationen wirken sich daher nur auf den äußeren Bereich aus.

Nun wird das Koordinatensystem nach links oben verschoben und der Pfad mit dem ersten Brush-Objekt gezeichnet (hell oder dunkel, je nach Einstellung). Sichtbar wird nur der über den Clipping-Bereich herausragende Teil (bei Anhebung der helle Rand). Anschließend wird das Koordinatensystem um den doppelten Versatz nach rechts unten verschoben und die Zeichenoperation mit dem anderen Brush-Objekt wiederholt.

---

Public Up As Boolean

```
Private Sub MainWindow_Paint(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.PaintEventArgs) _
    Handles MyBase.Paint

    Dim g As Graphics = e.Graphics

    ' Start- und Endfarbe für den Helligkeitsverlauf
    Dim c1 As Color = Color.FromArgb(100, 232, 252, 255)
    Dim c2 As Color = Color.FromArgb(100, 27, 34, 124)
    ' Brush erzeugen
    Dim lgb As New LinearGradientBrush(Me.ClientRectangle, c1, c2, _
        LinearGradientMode.ForwardDiagonal)

    ' Helligkeitsverlauf durch Übermalen
    g.FillRectangle(lgb, ClientRectangle)

    lgb.Dispose()

    ' String-Format-Objekt für zentrierte Ausgabe
    Dim sf As New StringFormat
    sf.Alignment = StringAlignment.Center
    sf.LineAlignment = StringAlignment.Center

    ' Pfad mit Text anlegen
    Dim path As New GraphicsPath
    path.AddString(Displaytext, Font.FontFamily, Font.Style, _
        Font.Size, New Rectangle(0, 0, ClientSize.Width, _
        ClientSize.Height), sf)

    ' Positionierung berechnen
    Dim bounds As RectangleF = path.GetBounds()
    Dim x As Integer = CInt((Me.ClientSize.Width - bounds.Width) / 2 _
        - bounds.Left)
    Dim y As Integer = CInt((Me.ClientSize.Height - bounds.Height) / _
        2 - bounds.Top)

    ' Zielposition für Textausgabe
    e.Graphics.TranslateTransform(x, y)

    ' Brush-Objekte zum Aufhellen/Abdunkeln
    Dim brushDark As New SolidBrush( _
        Color.FromArgb(140, 120, 120, 120))
    Dim brushLight As New SolidBrush( _
```

---



```

    Color.FromArgb(140, Color.White))

    ' Brush auswählen, abhängig von der Darstellungsart
    Dim br1, br2 As Brush
    If Up Then
        br1 = brushLight
        br2 = brushDark
    Else
        br1 = brushDark
        br2 = brushLight
    End If

    ' Versatz in Pixel
    Dim offset3D As Integer = 2

    g.SmoothingMode = SmoothingMode.HighQuality

    ' Zeichnen nur außerhalb des Pfadbereiches
    g.SetClip(path, CombineMode.Exclude)

    ' Versatz nach links oben
    g.TranslateTransform(-offset3D, -offset3D)

    ' dunkel/hell zeichnen
    g.FillPath(br1, path)

    ' Versatz nach rechts unten
    g.TranslateTransform(2 * offset3D, 2 * offset3D)

    ' hell/dunkel zeichnen
    g.FillPath(br2, path)

    ' Für weitere Zeichenoperationen evtl. Auskommentierung entfernen
    'g.TranslateTransform(-offset3D, -offset3D)
    'g.ResetClip()

    ' Aufräumen
    g.ResetTransform()
    brushDark.Dispose()
    brushLight.Dispose()
    sf.Dispose()
    path.Dispose()

End Sub

```

Listing 77: Generieren von 3D-Effekten auf strukturierten Hintergründen (Forts.)

Im Beispielprogramm lässt sich über die Member-Variable `Up` steuern, ob die Schrift vertieft oder erhaben dargestellt werden soll. Über die Oberfläche kann diese Variable mithilfe zweier Radio-Buttons umgeschaltet werden (siehe Abbildungen).

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Proper-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes



# GDI+ Bildbearbeitung

In diesem Kapitel geht es um den Umgang mit Bildern, hauptsächlich mit Fotos. Dank moderner Farbkameras und Scannern, schneller Prozessoren, preiswertem Arbeits- und Festplattenspeicher und hochauflösenden Farbmonitoren ist der Umgang mit digitalisiertem Fotomaterial heute den meisten PC-Anwendern vertraut. Das Kapitel erklärt einige Techniken in Zusammenhang mit der Darstellung und Veränderung von Bildern.

## 67 Bilder zeichnen

Um Bilder auf einem Fenster darzustellen, können Sie sich vieler Möglichkeiten bedienen. Schon die `Form`-Klasse bietet die Eigenschaft `BackgroundImage`, um dem Fenster ein Hintergrundbild zuzuweisen. Für die Platzierung kleinerer Bilder kann beispielsweise die `PictureBox` eingesetzt werden. Über `SizeMode` lassen sich verschiedene Skalierungsoptionen einstellen.

Oft reichen die Möglichkeiten der Steuerelemente jedoch nicht aus, will man z. B. spezielle Effekte erzielen oder zusätzliche Informationen in das Bild zeichnen. Dann ist es von Vorteil, die Bilder selbst mit den zur Verfügung stehenden GDI+-Klassen zu zeichnen.

Gezeichnet wird mit GDI+ grundsätzlich nur auf `Graphics`-Objekte. Diese Objekte können z. B. Fenstern, Steuerelementen, Bitmaps oder der Druckerausgabe zugeordnet sein. Mit einfachen Methoden erhalten Sie den Zugriff auf die `Graphics`-Objekte.

Die Ausgabe von Bildern erfolgt über die Methode `Graphics.DrawImage`. Diese Methode verfügt über ca. 30 Überladungen und bietet somit eine Vielzahl an Möglichkeiten. Angefangen bei einfachen 1:1 Ausgaben über skalierte Ausgaben bis hin zu verzerrten Darstellungen wird eine breite Palette an Effekten angeboten.

Listing 78 zeigt exemplarisch für ein `Button_Click`-Event, wie Sie in einem beliebigen Ereignis eines Fensters ein Bild auf dieses zeichnen können.

```
Private Sub ButtonZeichnen_Click(ByVal sender As System.Object, _  
    ByVal e As System.EventArgs) Handles ButtonZeichnen.Click  
  
    ' Bitmap aus Datei lesen  
    Dim Pict As New Bitmap("../banksial.jpg")  
  
    ' Graphics-Objekt für dieses Fenster holen  
    Dim g As Graphics = Me.CreateGraphics()  
  
    ' Bild an vorgegebener Position ohne Skalierung zeichnen  
    g.DrawImage(Pict, 150, 10, Pict.Width, Pict.Height)  
  
    ' Ressourcen freigeben  
    g.Dispose()  
    Pict.Dispose()  
  
End Sub
```

Listing 78: Zeichnen einer Bitmap

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Propert-  
yGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

Diese Vorgehensweise hat jedoch einen Nachteil: Wird nach dem Zeichnen des Bildes das Fenster durch ein anderes verdeckt und später wieder nach vorne geholt, dann wird die Zeichnung nicht aufgefrischt und kann ganz oder teilweise zerstört worden sein. Eine persistente Bildanzeige muss daher immer wieder neu gezeichnet werden.

Deswegen ist es sinnvoll, Grafikausgaben, die unabhängig vom Zustand des Fensters korrekt dargestellt werden sollen, entweder im `Paint`-Ereignis des Fensters oder des betreffenden Steuerelements bzw. in der entsprechenden `OnPaint`-Überschreibung zu platzieren. Muss das Fenster neu gezeichnet werden, weil es zuvor verdeckt war, sorgen Windows und das Framework automatisch dafür, dass die `Paint`-Methode aufgerufen wird.

Um die gleiche Funktionalität wie im vorigen Beispiel zu erreichen, benötigt man eine Referenzvariable für das Bitmap als Membervariable der Fensterklasse sowie eine Methode, die aufgerufen wird, wenn das Fenster neu gezeichnet werden soll. Listing 79 zeigt diesen Aufbau unter Verwendung des `Paint`-Events:

---

```
...
Private Pict As Bitmap
...

Private Sub ButtonZeichnen_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles ButtonZeichnen.Click

    ' Bild laden
    Pict = New Bitmap("..\\banksia1.jpg")

    ' Fenster neu zeichnen
    Me.Refresh()

End Sub

Private Sub Example2_Paint(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.PaintEventArgs) _
    Handles MyBase.Paint

    ' Wenn noch kein Bild zugewiesen wurde, nichts tun
    If Pict Is Nothing Then Exit Sub

    ' Bild zeichnen
    e.Graphics.DrawImage(Pict, 150, 10, Pict.Width, Pict.Height)

End Sub
```

---

Listing 79: Zeichnen im `Paint`-Ereignis

Alternativ können Sie auch die `OnPaint`-Methode der `Form`-Klasse überschreiben. Es ist jedoch oft wichtig, dass Sie die `OnPaint`-Methode der Basisklasse zusätzlich aufrufen. Listing 80 zeigt die Alternative

---

```
Protected Overrides Sub OnPaint(ByVal e As System.Windows.Forms.PaintEventArgs)

    ' OnPaint der Basisklasse sollte aufgerufen werden!
    MyBase.OnPaint(e)

    ' Wenn noch kein Bild zugewiesen wurde, nichts tun
    If Pict Is Nothing Then Exit Sub

    ' Bild zeichnen
    e.Graphics.DrawImage(Pict, 150, 10, Pict.Width, Pict.Height)

End Sub
```

---

Listing 80: Zeichnen in der OnPaint-Überschreibung

Sicher haben Sie bereits bemerkt, dass wir beim Aufruf von `DrawImage` die Bildgröße als Parameter übergeben haben. Mit dieser Angabe steuern Sie die Ausgabegröße. So können Sie beispielsweise die Kantenlänge des angezeigten Bildes halbieren:

---

```
' Bild zeichnen
e.Graphics.DrawImage(Pict, 150, 10, Pict.Width \ 2, Pict.Height \ 2)
```

---

Abbildung 32 zeigt das Ergebnis:



Abbildung 32: Zeichnen eines Fotos mit angepasster Kantenlänge

Wenn das Fenster geschlossen wird, sollten die zusätzlich belegten Ressourcen wieder freigegeben werden. Im Beispiel empfiehlt es sich, das geladene Bild mit `Dispose` zu entfernen (s. ).

---

```
Protected Overloads Overrides Sub Dispose(ByVal disposing _
    As Boolean)

    If disposing Then
        If Not (components Is Nothing) Then
```

---

Listing 81: Ressourcen freigeben

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

**Bild-  
bearbei-  
tung**

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

```
        components.Dispose()  
    End If  
End If  
  
MyBase.Dispose(disposing)  
'Ressourcen für Bild freigeben  
Pict.Dispose()  
  
End Sub
```

*Listing 81: Ressourcen freigeben (Forts.)*

Visual Studio fügt den Code für die Überschreibung von `Dispose` beim Anlegen des Fensters bereits ein. Sie müssen ihn lediglich für die Freigabe der Bitmap ergänzen.

## 68 Bildausschnitt zoomen

Mit einfachen Mitteln ist es möglich, einen Bildausschnitt vom Anwender auswählen zu lassen und diesen vergrößert darzustellen. Abbildung 33 zeigt ein derartiges Beispiel.

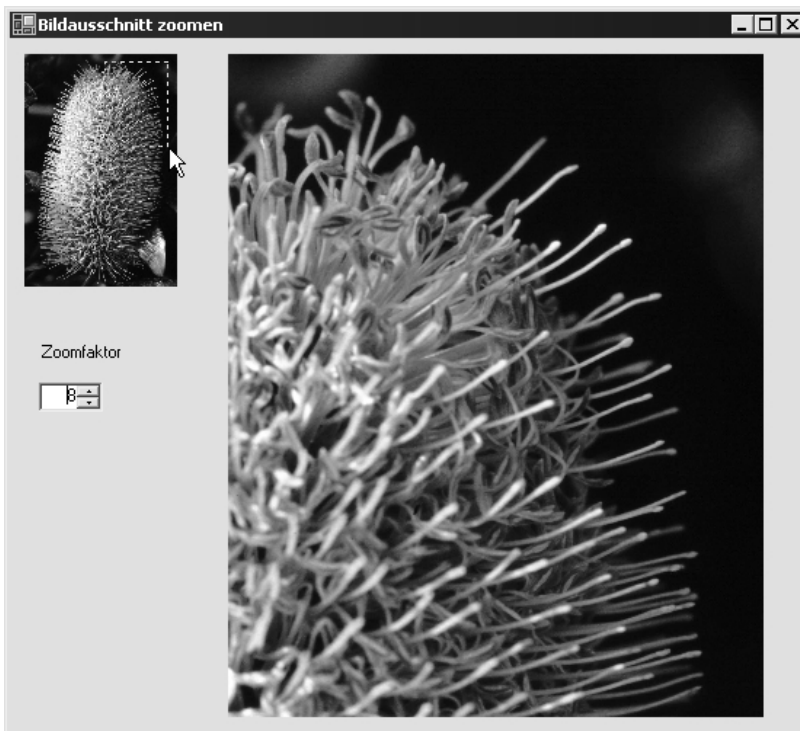


*Abbildung 33: Vergrößern eines Bildausschnitts*

Das Bild wird als Miniatur links oben im Fenster dargestellt. Der Anwender kann auf dieser Miniatur einen Auswahlrahmen aufziehen. Nach Loslassen der Maustaste wird der ausgewählte Bereich mit der eingestellten Vergrößerung gezeichnet. Der Vergrößerungsfaktor kann verändert werden, ohne dass der Bereich neu ausgewählt werden muss.

Folgende Schritte sind für dieses Beispiel notwendig. Zunächst werden einige Membervariablen für die Mausektionen und Zeichenoperationen benötigt:

```
' Offset für die Anzeige der Miniatur
Private offsetX As Integer = 10, offsetY As Integer = 10

' Skalierungsfaktor für Miniatur
Private scaleFactor As Integer = 4

' Das anzuzeigende Bild
Private pict As New Bitmap("../banksial.jpg")

' Gespeicherte Mauspositionen
Private startX, startY As Integer
Private endX, endY As Integer

' Das ausgewählte Rechteck
Private zoomRect As Rectangle

' Offset für die zu zeichnende Vergrößerung
Private offsetZoomX As Integer = 100
Private offsetZoomY As Integer = 10

' Vergrößerungsfaktor
Private zoomFactor As Integer = 5
```

Im MouseDown-Event werden die Startkoordinaten für das Ziehen des Auswahlrahmens gespeichert (Listing 82).

```
Private Sub Example3_MouseDown(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.MouseEventArgs) _
    Handles MyBase.MouseDown

    If e.Button <> MouseButtons.Left Then Exit Sub

    ' Ausgangsposition merken
    ' Minimum ist der Punkt links oben
    startX = Math.Max(e.X, offsetX)
    startY = Math.Max(e.Y, offsetY)
    endX = startX
    endY = startY
End Sub
```

Listing 82: MouseDown-Ereignis speichert die Startkoordinaten

Im MouseMove-Event (Listing 83) wird der alte Auswahlrahmen gelöscht und der neue gezeichnet. Zum Zeichnen des Auswahlrechtecks wird die Methode DrawReversibleFrame verwendet. Sie ist eine der wenigen GDI+-Methoden, die XOR-Drawing unterstützt. Das Zeichnen der Linien geschieht durch Invertieren der Pixel-Farben. Ein zweimaliges Zeichnen stellt somit den Ausgangszustand wieder her. Die Methode zeichnet allerdings nicht auf einen vorgegebenen Ausgabe-kontext, sonder direkt auf den Bildschirm. Daher müssen die Koordinaten umgerechnet werden von lokalen Fensterkoordinaten in Bildschirmkoordinaten.

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

**Bild-  
bearbei-  
tung**  
Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

```

Private Sub Example3_MouseMove(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.MouseEventArgs) _
    Handles MyBase.MouseMove

    If e.Button <> MouseButtons.Left Then Exit Sub

    ' Umrechnung in Screen-Koordinaten
    Dim p As New Point(startX, startY)
    p = PointToScreen(p)
    Dim r As New Rectangle(p.X, p.Y, endX - startX, endY - startY)

    ' altes Auswahlrechteck durch wiederholtes Zeichnen löschen
    ControlPaint.DrawReversibleFrame(r, Color.Black, _
        FrameStyle.Dashed)

    ' neue Koordinaten umrechnen und Bereich einschränken
    ' Maximum ist der Punkt rechts unten
    endX = Math.Min(e.X, offsetX + pict.Width \ scaleFactor)
    endY = Math.Min(e.Y, offsetY + pict.Height \ scaleFactor)
    r = New Rectangle(p.X, p.Y, endX - startX, endY - startY)

    ' neues Auswahlrechteck zeichnen
    ControlPaint.DrawReversibleFrame(r, Color.Black, _
        FrameStyle.Dashed)

End Sub

```

---

*Listing 83: Zeichnen des Auswahlrahmens im MouseMove-Ereignis*

In jedem MouseMove-Ereignis wird zunächst das vorherige Rechteck gezeichnet und die Linien somit wieder entfernt. Anschließend wird das neue gezeichnet. Im MouseUp-Event (Listing 84) werden die Endkoordinaten ermittelt und das Fenster neu gezeichnet.

---

```

Private Sub Example3_MouseUp(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.MouseEventArgs) _
    Handles MyBase.MouseUp

    If e.Button <> MouseButtons.Left Then Exit Sub

    ' Auswahlbereich merken
    zoomRect = New Rectangle(0, 0, endX - startX, endY - startY)

    ' Fenster neu zeichnen (und damit auch den Zoom-Bereich)
    Me.Refresh()

End Sub

```

---

*Listing 84: Abschließende Berechnung im MouseUp-Ereignis*

Der zuletzt gezeichnete Rahmen muss nicht wieder gelöscht werden, da das Fenster ohnehin neu gezeichnet wird. Im Paint-Event (Listing 85) werden die Miniatur und der vergrößerte Ausschnitt gezeichnet.



---

```

Private Sub Example3_Paint(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.PaintEventArgs) _
    Handles MyBase.Paint

    ' Miniatur zeichnen
    e.Graphics.DrawImage(pict, offsetX, offsetY, pict.Width \ _
        scaleFactor, pict.Height \ scaleFactor)

    ' Ausschnitt nur zeichnen, wenn ein gültiges Auswahlrechteck
    ' vorliegt
    If zoomRect.Width <= 0 Or zoomRect.Height <= 0 Then Exit Sub

    ' Zielrechteck für die Ausgabe in Fensterkoordinaten
    Dim destRect As New Rectangle(offsetZoomX, offsetZoomY, _
        zoomRect.Width * zoomFactor, zoomRect.Height * zoomFactor)

    ' Quellrechteck für den Ausschnitt in Bildkoordinaten
    Dim srcRect As New Rectangle( _
        (startX - offsetX) * scaleFactor, _
        (startY - offsetY) * scaleFactor, _
        zoomRect.Width * scaleFactor, _
        zoomRect.Height * scaleFactor)

    ' Ausschnitt zeichnen
    e.Graphics.DrawImage(pict, destRect, srcRect, GraphicsUnit.Pixel)

End Sub

```

---

Listing 85: Bilder zeichnen im Paint-Ereignis

Beide Bilder werden aus der gleichen Quelle gezeichnet. `pict` verweist auf das bei der Definition zugewiesene Bitmap. In beiden Fällen wird `DrawImage` verwendet, um das jeweilige Bild auf das Fenster zu zeichnen. Mit je einem Rechteck für die Quelle (in Bildkoordinaten) und für das Ziel (in Fensterkoordinaten) wird der Bildausschnitt an der gewünschten Position mit der eingestellten Vergrößerung dargestellt.

Im Changed-Event (Listing 86) des `NumericUpDown`-Steuerelements wird der neue Zoom-Faktor eingelesen und das Fenster neu gezeichnet.

---

```

Private Sub NUDZoom_ValueChanged(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles NUDZoom.ValueChanged

    ' Zoomfaktor speichern
    zoomFactor = CInt(NUDZoom.Value)
    ' Und Fenster neu zeichnen
    Me.Refresh()

End Sub

```

---

Listing 86: Änderung der Eingabedaten führt zum Neuzeichnen des Fensters

---

Basics

---

Datum/  
Zeit

---

Anwen-  
dungen

---

Zeich-  
nen

---

**Bild-  
bearbei-  
tung**

---

Win-  
dows  
Forms

---

Controls

---

Propert-  
yGrid

---

Datei-  
system

---

Netz-  
werk

---

System

---

Daten-  
banken

---

XML

---

Wissen-  
schaft

---

Ver-  
schiede-  
nes

---

Die Qualität der Vergrößerung hängt natürlich direkt von der Bildqualität der Bilddatei ab. In der Abbildung 33 wurde eine JPG-Datei mit einer Auflösung von 3158 \* 4824 Pixel verwendet. Dadurch können die Ausschnitte qualitativ hochwertig angezeigt werden. Als Nachteil erkauft man sich dabei allerdings eine relativ lange Bildaufbauzeit. Auch für die Miniatur muss das gesamte Bild skaliert werden. Verwendet man andererseits Bilddateien mit geringerer Auflösung, dann wird zwar der Bildaufbau beschleunigt, aber die Vergrößerung stößt schnell an ihre Grenzen.

## 69 Basisklasse für eine Dia-Show

In den nachfolgenden Rezepten wird gezeigt, wie man Bilder ineinander überblenden kann. Dieses Rezept beschreibt die Basisklasse für eine Dia-Show, die selbständig lauffähig ist und eine Reihe von Fotos zyklisch anzeigen kann. Sie besitzt bereits die grundlegende Funktionalität für die spätere Überblendung, so dass die anderen Beispielklassen von ihr abgeleitet und nur geringfügig ergänzt werden müssen.

Die Bilder werden in diesem Beispiel nicht auf das Fenster, sondern auf eine `PictureBox` gezeichnet. Das hat den Vorteil, dass man das Fenster im Designer leichter arrangieren und die Bilder mit einem Rahmen versehen kann. Der Zeichenmechanismus ist jedoch identisch.



Abbildung 34: Das Basisfenster für die Dia-Show

Nach Betätigung der START-Taste beginnt die ÜBERBLENDZEIT. Sie wird unterteilt in ÜBERBLEND-SCHRITTE. Anschließend wird das aktuelle Bild entsprechend während der STANDZEIT unverändert angezeigt, bevor der Zyklus von neuem beginnt. Abbildung 34 zeigt das Basisfenster.

Für den Ablauf werden die folgenden Member-Variablen definiert:

---

```
' Zähler für Überblendvorgang
Protected Counter As Integer
```

---

---

```
' Bitmaps, die zyklisch gewechselt werden sollen
```

```
' Bilder werden in FormLoad geladen
```

```
Public Photos() As Bitmap
```

```
' Index für anzuzeigendes Bitmap
```

```
Public IndexActivePhoto As Integer = 0
```

```
' Referenz des aktuellen Bildes
```

```
Protected ActivePic As Bitmap
```

```
' Referenz des nächsten Bildes
```

```
Protected NextPic As Bitmap
```

```
' Client-Rect der PictureBox
```

```
Protected PictureBoxRect As Rectangle
```

```
' Anzahl der Überblendschritte
```

```
Protected Steps As Integer = 50
```

```
' Überblendzeit in Millisekunden
```

```
Protected FadingTime As Integer = 2000
```

---

Um das Beispiel überschaubar zu halten, werden die Bilder zu Beginn geladen. Die Referenzen der Bitmap-Objekte werden im Array `Photos` gespeichert. Für das Beispiel wurden relative Pfade gewählt. Es wird davon ausgegangen, dass die Bilddateien im Parent-Verzeichnis des bin-Ordners liegen, also so, wie es in Visual Studio eingerichtet wurde. Leider kommt der Designer selbst nicht mit diesen Pfaden klar, da er ein anderes Ausgangsverzeichnis benutzt. Deswegen muss das Laden der Bilder innerhalb eines Try/Catch-Blocks erfolgen und wird hier im `Form_Load`-Ereignis vorgenommen (Listing 87).

---

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

```
' Die Bilder müssen hier geladen werden, da im Design-Mode
```

```
' der Pfad nicht stimmt
```

```
Try
```

```
    Photos = New Bitmap() { _
        New Bitmap("../Swampskink1.jpg"), _
        New Bitmap("../Swampskink2.jpg"), _
        New Bitmap("../Swampskink3.jpg"), _
        New Bitmap("../Swampskink4.jpg"), _
        New Bitmap("../Prom28.jpg") }
```

```
    ActivePic = Photos(0)
```

```
    NextPic = Photos(1)
```

```
' Größe der PictureBox festlegen
```

```
PictureBoxRect = New Rectangle(0, 0, Photos(0).Width, _
    Photos(0).Height)
```

---

Listing 87: Einrichtung der Bilder

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

**Bild-  
bearbei-  
tung**  
Win-  
dows  
Forms

Controls

Propert-  
yGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

```

PictureBox.ClientSize = New Size(PictureBoxRect.Width, _
    PictureBoxRect.Height)

Catch ex As Exception
    'MessageBox.Show("Fehler beim Einrichten der Bilder: " & ex.Message)
End Try

End Sub

```

---

*Listing 87: Einrichtung der Bilder (Forts.)*

Ebenfalls zur Vereinfachung wurden Bilder mit gleicher Größe gewählt. Diese Größe wird in der Variablen `PictureBoxRect` festgehalten. Durch das Setzen der `ClientSize`-Eigenschaft der `PictureBox` wird deren Größe so angepasst, dass die Bilder ohne Skalierung gezeichnet werden können.

Hinweis

Die `PictureBox` besitzt eine Eigenschaft `SizeMode`, der unter anderem der Wert `AutoSize` zugewiesen werden kann. Hierdurch wird die Größe der `PictureBox` an die Abmessungen des Bildes angepasst. Laut Dokumentation soll das so erfolgen, dass das Bild vollständig auf den Client-Bereich der `PictureBox` passt.

Leider gibt es hier einen kleinen Bug, denn es wurde vergessen, den Rahmen zu berücksichtigen. Hat die `PictureBox` einen Rahmen, dann wird der Zeichenbereich für das Bild zu klein.

Der Ausweg besteht darin, den `SizeMode` auf `Normal` zu stellen und der Eigenschaft `ClientSize` die benötigte Bildgröße zuzuweisen.

Für die Standzeit wird der Timer `DisplayTimer` benutzt, dessen `Interval`-Eigenschaft auf den im Eingabefeld angegebenen Wert in Millisekunden gesetzt wird. Die Impulse für die Überblendung erzeugt der Zeitgeber `FadingTimer`. Er generiert während der eingestellten Überblendzeit so viele Ereignisse, wie Überblendschritte gewünscht werden.

Da sich die Überblendzeit also aus vielen Teilschritten zusammensetzt und diese je nach Geschwindigkeit des Rechners unterschiedlich lang sein können, wird die tatsächliche Ausführungszeit länger sein als eingestellt. Für das Beispiel ist das jedoch unerheblich und soll daher auch nicht weiter betrachtet werden.

Im Click-Ereignis der START/STOP-Taste `BUTTONFADE` werden die Timer entsprechend des aktuellen Zustands gesetzt (Listing 88).

---

```

Private Sub ButtonFade_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles ButtonFade.Click

    ' Standzeit-Timer stoppen
    DisplayTimer.Enabled = False

    ' Timer-Intervall setzen und starten
    FadingTimer.Interval = FadingTime \ Steps

```

---

*Listing 88: Überblendvorgang starten bzw. stoppen*

```

' Timer ein-/ausschalten
FadingTimer.Enabled = Not FadingTimer.Enabled

If FadingTimer.Enabled Then
    ' Tastenbeschriftung setzen
    ButtonFade.Text = "Stop"
    ' Sofort mit nächstem Überblendvorgang beginnen
    Counter = 0
Else
    ' Tastenbeschriftung setzen
    ButtonFade.Text = "Start"
End If

End Sub

```

*Listing 88: Überblendvorgang starten bzw. stoppen (Forts.)*

Nach Ablauf der Standzeit wird der Überblendvorgang eingeleitet (Listing 89).

```

Private Sub DisplayTimer_Tick(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles DisplayTimer.Tick

    ' Standbild-Timer stoppen
    DisplayTimer.Enabled = False

    ' Überblend-Timer starten
    FadingTimer.Enabled = True

End Sub

```

*Listing 89: DisplayTimer-Ereignis*

Für jeden Überblendschritt wird ein `FadingTimer`-Event ausgelöst. Die zentrale Zählervariable `Counter` wird in jedem Schritt inkrementiert. Listing 90 zeigt die Implementierung.

```

Private Sub FadingTimer_Tick(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles FadingTimer.Tick

    ' Wenn der Zähler abgelaufen ist, auf nächstes Bild
    ' überblenden
    If Counter >= Steps Then

        ' Index neu bestimmen
        IndexActivePhoto = (IndexActivePhoto + 1) Mod Photos.Length
        ActivePic = Photos(IndexActivePhoto)

        ' Nächstes Bild bestimmen
        NextPic = Photos((IndexActivePhoto + 1) Mod Photos.Length)
    End If
End Sub

```

*Listing 90: Vorbereitung für die Überblendung*

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nen**Bild-  
bearbei-  
tung**Win-  
dows  
Forms

Controls

Propert-  
yGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

---

```

' Zähler initialisieren
Counter = 0

' PictureBox neu zeichnen
PictureBox.Refresh()

' Überblenden abgeschlossen
FadingTimer.Enabled = False

' Standzeit beginnt
DisplayTimer.Enabled = True

Exit Sub
End If

Counter += 1

' Überblendvorgang
Dim g As Graphics = PictureBox.CreateGraphics()

' Hier erfolgt in den abgeleiteten Klassen der Überblendvorgang
Fade(g)

' Ressourcen freigeben
g.Dispose()

End Sub

```

---

*Listing 90: Vorbereitung für die Überblendung (Forts.)*

Im letzten Überblendschritt wird die Standzeit initialisiert. Dazu wird mittels `Refresh` die `PictureBox` zum Neuzeichnen aufgefordert. So wird sichergestellt, dass das neue Bild unverfälscht dargestellt wird. Der Zähler für den Bildindex (`IndexActivePhoto`) wird inkrementiert bzw. auf Null gesetzt.

Für jeden Überblendschritt wird die Methode `Fade` aufgerufen. Diese wird in den abgeleiteten Klassen überschrieben und implementiert den jeweiligen Algorithmus für die Überblendung. Ihr wird die Referenz des `Graphic`-Objektes übergeben, das zuvor mit `CreateGraphics` angelegt wird und anschließend wieder freigegeben werden muss.

`Fade` ist nicht abstrakt definiert, damit die Klasse auch vom Designer benutzt werden kann. Dieser muss die Basisklasse instanzieren können und kommt daher nicht mit abstrakten Klassen zurecht.

---

```

' Überblendung für aktuellen Zählerstand
Protected Overridable Sub Fade(ByVal g As Graphics)

End Sub

```

---

*Listing 91: Die leere, nicht abstrakte Definition der Methode Fade*

Durch `Refresh` wird das `Paint`-Ereignis ausgelöst. Hier wird das aktive Bild gezeichnet, so dass es immer korrekt sichtbar ist, auch wenn das Fenster zwischenzeitlich verdeckt war (Listing 92).

```

Private Sub PictureBox_Paint(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.PaintEventArgs) _
    Handles PictureBox.Paint

    ' Das aktive Bild zeichnen
    If Not ActivePic Is Nothing Then
        e.Graphics.DrawImage(ActivePic, PictureBox.ClientRectangle)
    End If

End Sub

```

Listing 92: Zeichnen des aktiven Bildes in OnPaint

Beim Schließen des Fensters werden die Bitmaps freigegeben:

```

' Ressourcen für Bilder freigeben
Dim i As Integer
For i = 0 To Photos.GetUpperBound(0)
    Photos(i).Dispose()
Next

```

In den Ereignis-Routinen `NUDSteps_ValueChanged`, `NUDFadingtime_ValueChanged` und `NUDDisplaytime_ValueChanged` werden die Einstellungen der Eingabefelder in die entsprechenden Variablen übertragen.

Die Klasse kann instanziiert und das Fenster angezeigt werden. Der komplette Mechanismus der Dia-Show ist integriert. Alle Bilder werden zyklisch gewechselt.

In den folgenden Tipps werden nun Beispiele für Überblendungen vorgestellt. Sie sollen Ihnen die Mechanismen aufzeigen, die zur Verfügung stehen. Lassen Sie Ihrer Phantasie freien Lauf und programmieren Sie auf Basis der Beispiele neue Varianten.

## 70 Horizontal und vertikal überblenden

Alle Variante verwenden dasselbe Schema: Das bestehende Bild wird in jedem Schritt mehr und mehr vom nachfolgenden Bild überdeckt, bis schließlich nur noch das neue Bild zu sehen ist. Die ersten vier Varianten verwenden hierfür den Clipping-Mechanismus, die letzte Alpha-Blending.

Um ein Bild horizontal oder vertikal einzublenden, benötigt man einen rechteckigen Bereich, in den das neue Bild gezeichnet wird und der schrittweise vergrößert wird. Am Ende des Überblendvorgangs füllt er die gesamte Bildfläche aus.

Wie bereits in Rezept 5.2 beschrieben, könnten Sie mit Hilfe der `DrawImage`-Methode direkt einen Ausschnitt des neuen Bildes in das Zielrechteck zeichnen. Hier soll jedoch eine andere Vorgehensweise angewendet werden: Zeichnen in eine *Clipping-Region*.

Die Clipping-Region ist der Bereich des `Graphics`-Objekts, in den gezeichnet werden kann. Alles, was außerhalb dieses Bereiches gezeichnet wird, wird abgeschnitten (to clip, Abschneiden) und somit nicht gezeichnet. Im Ausgangszustand eines `Graphics`-Objekts füllt der Clipping-Bereich die gesamte Zeichenfläche.

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tung  
Win-  
dows  
Forms

Controls

Proper-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

Sie können den Clipping-Bereich jedoch beliebig gestalten. Das `Graphics`-Objekt stellt hierfür u.a. die Methode `SetClip` und die Eigenschaft `Clip` zur Verfügung. Im einfachsten Fall kann der Bereich aus einem Rechteck bestehen, genau das, was für dieses Beispiel benötigt wird.

Um diese Variante etwas flexibler zu gestalten, soll der Anwender die Möglichkeit haben, einzustellen, in welcher Richtung die Überblendung erfolgen soll. Die möglichen Richtungen sind zunächst einmal:

- ▶ von links nach rechts
- ▶ von rechts nach links
- ▶ von oben nach unten
- ▶ von unten nach oben

Abbildung 35 zeigt die letzte Variante, in der das neue Bild beginnend von unten über das bestehende gezeichnet wird.

Leider können die Bilder im Buch nur schwarz-weiß wiedergegeben werden und stellen ja auch nur einen Ausschnitt des zeitlichen Verlaufs dar. Die Effekte sind daher nicht so deutlich zu erkennen. Aber probieren Sie das Programm mit den mitgelieferten Fotos aus. Sie werden überrascht sein, wie gut die mit einfachen Mitteln erreichten Überblend-Effekte zur Geltung kommen.



Abbildung 35: Vertikale Überblendung von unten nach oben

Da Position und Größe des Rechteckes je nach vorgegebener Richtung unterschiedlich ausfallen, werden für jede Richtung Parameter definiert. Hierzu wird für jede Variante eine Instanz der in Listing 93 definierten Klasse `FadingDirection` angelegt. Kern dieser Klasse sind die Member-Variablen `StartposX`, `StartposY` sowie `IncrX` und `IncrY`. Sie werden als Multiplikatoren eingesetzt und erlauben so die Durchführung der Berechnung ohne Verzweigungen.



---

Public Class FadingDirection

' Text, der in der Auswahlliste angezeigt werden soll

Public Title As String

' Startposition für den Überblendvorgang

' 0 = links bzw. oben

' 1 = rechts bzw. unten

Public StartposX, StartposY As Integer

' Richtung für den Überblendvorgang

' 1 = links-&gt;rechts bzw. oben-&gt;unten

' 0 = kein Überblenden in der betreffenden Richtung

' -1 = rechts-&gt;links bzw. unten-&gt;oben

Public IncrX, IncrY As Integer

' Konstruktor zum Setzen der Membervariablen

Public Sub New(ByVal title As String, \_  
ByVal startposX As Integer, ByVal startposY As Integer, \_  
ByVal incrX As Integer, ByVal incrY As Integer)

Me.Title = title

Me.StartposX = startposX

Me.StartposY = startposY

Me.IncrX = incrX

Me.IncrY = incrY

End Sub

' ToString gibt den Titel zurück

Public Overrides Function tostring() As String

Return Title

End Function

End Class

---

Listing 93: Die Klasse FadingDirection gibt die Richtung für den Überblendvorgang vor

Im Konstruktor der Fensterklasse werden die benötigten Instanzen angelegt und der ComboBox CBODirection zugefügt. Den anzuzeigenden Text ermittelt die ComboBox mit Hilfe der ToString-Überschreibung der Klasse FadingDirection.

' Combobox mit den acht möglichen Überblendvarianten füllen

CBODirection.Items.Add( \_  
New FadingDirection("Links -> Rechts", 0, 0, 1, 0))

CBODirection.Items.Add( \_  
New FadingDirection("Rechs -> Links", 1, 0, -1, 0))

CBODirection.Items.Add( \_  
New FadingDirection("Oben -> Unten", 0, 0, 0, 1))

CBODirection.Items.Add( \_  
New FadingDirection("Unten -> Oben", 0, 1, 0, -1))

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Proper-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schie-  
denes

Wählt der Anwender einen Eintrag aus der `ComboBox` aus, dann wird die Referenz des angehängten `FadingDirection`-Objektes in der Variablen `Direction` gespeichert:

---

```
Private Sub CB0Direction_SelectedIndexChanged(ByVal sender As _
    System.Object, ByVal e As System.EventArgs) _
    Handles CB0Direction.SelectedIndexChanged

    ' Auswahl speichern
    Direction = DirectCast(CB0Direction.SelectedItem, FadingDirection)

End Sub
```

---

Der eigentliche Überblendvorgang wird in der Methode `Fade` implementiert. Listing 94 zeigt den Aufbau der Funktion. Hier wird an Hand des Zählerstands (Variable `Counter` der Basisklasse) die Größe und Position des Rechtecks berechnet. Betrachten wir für die Berechnung zunächst den in Abbildung 35 gezeigten Fall der vertikalen Überblendung:

Zur Erinnerung: `PictureBoxRect` hat die Dimension des Zeichenbereichs.

Die Breite des Rechtecks bleibt konstant:

---

```
w = PictureBoxRect.Width
```

---

Die Höhe ändert sich proportional zum Zählerstand:

---

```
h = Counter * PictureBoxRect.Height / Steps
```

---

Die x-Koordinate des Rechtecks bleibt konstant:

---

```
x = 0
```

---

Nun muss unterschieden werden, ob das Rechteck von oben nach unten oder von unten nach oben wachsen soll.

Für den Fall, dass von oben nach unten überblendet wird, bleibt die y-Koordinate konstant:

---

```
y = 0
```

---

Wird von unten nach oben überblendet, dann verschiebt sich y um die Höhe des Rechtecks. Der Bezugspunkt ist die untere Kante, also `PictureBoxRect.Height`:

---

```
y = PictureBoxRect.Height - h
```

---

Die beiden Fälle lassen sich nun zusammenfassen:

---

```
y = Direction.StartposY * (PictureBoxRect.Height - h)
```

---

Im ersten Fall ist StartposY 0 und y somit ebenfalls 0, im zweiten Fall ist StartposY 1 und das Ergebnis PictureBox.Rect.Height-h.

```
Protected Overrides Sub Fade(ByVal g As Graphics)

    ' Hilfsvariablen für Berechnungen
    ' 1: Veränderung in der betreffenden Richtung
    ' 0: Keine Veränderung
    Dim absIncrX As Integer = Math.Abs(Direction.IncrX)
    Dim absIncrY As Integer = Math.Abs(Direction.IncrY)

    ' Breite und Höhe des Clipping-Rechtecks
    Dim w As Integer = CInt(counter * _
        (PictureBox.Width / steps)) * absIncrX + _
        (1 - absIncrX) * PictureBox.Width
    Dim h As Integer = CInt(counter * _
        (PictureBox.Height / steps)) * absIncrY + _
        (1 - absIncrY) * PictureBox.Height

    ' Linke obere Ecke des Clipping-Rechtecks
    Dim x As Integer = Direction.StartposX * _
        (PictureBox.Width - w) * absIncrX
    Dim y As Integer = Direction.StartposY * _
        (PictureBox.Height - h) * absIncrY

    ' Das ermittelte Rechteck
    Dim destRect As New Rectangle(x, y, w, h)

    ' Clipping-Bereich setzen
    g.SetClip(destRect)

    ' Bild zeichnen
    g.DrawImage(NextPic, PictureBox)

End Sub
```

*Listing 94: Überschreibung der Methode Fade für die Realisierung der horizontalen oder vertikalen Überblendung*

Um bei der Berechnung nicht zwischen vertikal und horizontal unterscheiden zu müssen, kann man den Absolutwert von Direction.IncrX bzw. Direction.IncrY als Multiplikator heranziehen. Er ist 0, wenn in der betreffenden Richtung keine Änderung erfolgt, und 1, wenn das Rechteck in dieser Richtung linear wachsen soll. Somit ergeben sich folgende Formeln:

```
w = Counter * PictureBox.Width / Steps * |IncrX| + (1 - |IncrX|) * PictureBox.Width
h = Counter * PictureBox.Height / Steps * |IncrY| + (1 - |IncrY|) * PictureBox.Height
x = StartposX * (PictureBox.Width - w) * |IncrX|
y = StartposY * (PictureBox.Height - h) * |IncrY|
```

Mit diesen vier Formeln werden die Parameter für das Clipping-Rechteck für alle Varianten ohne Verzweigungen bestimmt und anschließend ein neues Rechteck (destRect) generiert.

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tung  
Win-  
dows  
Forms

Controls

Propor-  
tityGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

Dieses Rechteck kann nun direkt der Methode `SetClip` des `Graphics`-Objektes übergeben werden. Anschließend wird das neue Bild genauso gezeichnet, als hätte es die volle Größe. Tatsächlich wird aber alles außerhalb des berechneten Rechtecks abgeschnitten.

Die obige Berechnung erlaubt ohne Änderung der `Fade`-Methode noch eine Gestaltungsvariante:

Es ist möglich, das Rechteck gleichzeitig sowohl horizontal als auch vertikal zu verändern. Dadurch ergeben sich folgende zusätzliche Überblendrichtungen:

- von oben links nach unten rechts
- von oben rechts nach unten links
- von unten links nach oben rechts
- von unten rechts nach oben links

Abbildung 36 zeigt die letztgenannte Variante. Im Konstruktor des Fensters werden die zusätzlichen `FadingDirection`-Objekte erzeugt und an die `ComboBox` angehängt:

```
CBODirection.Items.Add( _
    New FadingDirection("O.L. -> U.R.", 0, 0, 1, 1))
CBODirection.Items.Add( _
    New FadingDirection("O.R. -> U.L.", 1, 0, -1, 1))
CBODirection.Items.Add( _
    New FadingDirection("U.L. -> O.R.", 0, 1, 1, -1))
CBODirection.Items.Add( _
    New FadingDirection("U.R. -> O.L.", 1, 1, -1, -1))
```

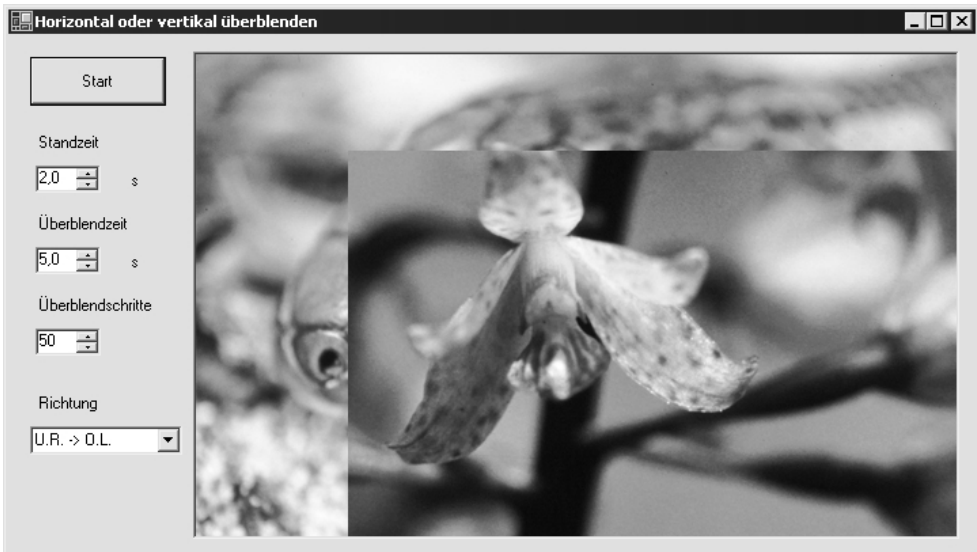


Abbildung 36: Überblendvariante »unten rechts nach oben links«

Als weitere Variation können Sie das neue Bild wie eine Gummifläche aufziehen, also das Bild auf den Clipping-Bereich skalieren. Der Effekt lässt sich sehen. Allerdings müssen Sie berücksichtigen, dass die Skalierung sehr rechenintensiv ist und somit recht lange dauern kann. Schließlich

müssen alle Pixel neu berechnet werden, während beim Clipping überflüssige Pixel einfach ignoriert werden können.

## 71 Diagonal überblenden

Um das neue Bild diagonal einblenden zu können, benötigt man einen dreieckigen Clipping-Bereich. GDI+ bietet eine einfache Möglichkeit, nicht rechteckige Clipping-Bereiche zu definieren: durch ein `GraphicsPath`-Objekt.

Ein `GraphicsPath`-Objekt, kurz Pfad genannt, kann verschiedene geometrische Figuren aufnehmen. Dazu gehören u.a. Geraden, Rechtecke, Ellipsen, Ellipsenausschnitte und Polygone und Strings. Der Pfad kann für verschiedene Aktionen genutzt werden. Es können die Umrisse gezeichnet werden, er kann farbig gefüllt gezeichnet werden und man kann ihn als Clipping-Bereich nutzen.

Für die Aufgabenstellung wird ein Pfad benötigt, der aus drei Linien besteht, die ein Dreieck bilden. Vier Varianten müssen berücksichtigt werden:

- ▶ von oben links nach unten rechts
- ▶ von oben rechts nach unten links
- ▶ von unten links nach oben rechts
- ▶ von unten rechts nach oben links

Das Dreieck beginnt in der ausgewählten Startecke und wird in Richtung der Zielecke vergrößert. Damit im letzten Schritt des Überblendvorgangs das alte Bild vollständig überdeckt wird, muss die Kantenlänge des Dreiecks zum Schluss doppelt so groß sein wie die betreffende Kantenlänge des Zeichenbereichs. In Abbildung 37 werden die Koordinaten und Variablen erläutert, die im Programm verwendet werden.

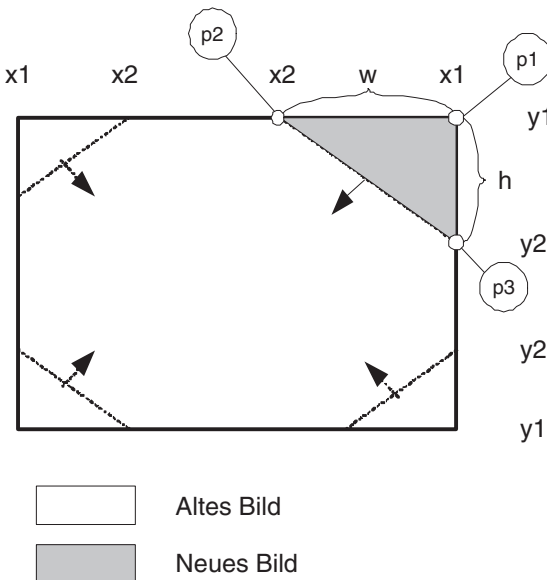


Abbildung 37: Koordinaten für die Berechnung des Dreiecks

Die einzelnen Größen berechnen sich wie folgt:

---

```
w = Counter * PictureBoxRect.Width / Steps * 2
h = Counter * PictureBoxRect.Height / Steps * 2
x1 = StartposX * PictureBoxRect.Width
x2 = StartposX * PictureBoxRect.Width + IncrX * w
y1 = StartposY * PictureBoxRect.Height
y2 = StartposY * PictureBoxRect.Height + IncrY * h
p1 = (x1, y1)
p2 = (x2, y1)
p3 = (x1, y2)
```

---

Mit `path.AddLine(...)` werden die drei Linien des Dreiecks dem Pfad hinzugefügt. Der Methodenaufruf `SetClip(path)` übernimmt diesen Pfad als Clipping-Bereich. Listing 95 zeigt die vollständige Implementierung. Das Ergebnis sehen Sie in Abbildung 38.

---

```
Protected Overrides Sub Fade(ByVal g As _
    System.Drawing.Graphics)

    ' Breite des Dreiecks bestimmen
    Dim w As Integer = CInt(counter * _
        (PictureBoxRect.Width / Steps)) * 2

    ' Höhe des Dreiecks bestimmen
    Dim h As Integer = CInt(counter * _
        (PictureBoxRect.Height / Steps)) * 2

    ' X-Koordinate des Eckpunktes (links oder rechts)
    Dim x1 As Integer = Direction.StartposX * PictureBoxRect.Width

    ' X-Koordinate der inneren Ecke
    Dim x2 As Integer = Direction.StartposX * _
        PictureBoxRect.Width + Direction.IncrX * w

    ' Y-Koordinate des Eckpunktes (oben oder unten)
    Dim y1 As Integer = Direction.StartposY * PictureBoxRect.Height

    ' Y-Koordinate der inneren Ecke
    Dim y2 As Integer = Direction.StartposY * _
        PictureBoxRect.Height + Direction.IncrY * h

    ' Zur Übersicht die drei Punkte
    Dim p1 As New Point(x1, y1)
    Dim p2 As New Point(x2, y1)
    Dim p3 As New Point(x1, y2)

    ' Einen Pfad mit diesem Dreieck anlegen
    Dim path As New GraphicsPath()
    path.AddLine(p1, p2)
    path.AddLine(p2, p3)
```

---

Listing 95: Die Fade-Methode für den diagonalen Übergang

```
path.AddLine(p3, p1)

' Das Dreieck ist die Clipping-Region
g.SetClip(path)

' Neues Bild zeichnen
g.DrawImage(NextPic, PictureBox)

' Entsorgung
path.Dispose()
```

End Sub

Listing 95: Die Fade-Methode für den diagonalen Übergang (Forts.)



Abbildung 38: Diagonal überblenden von oben rechts nach unten links

## 72 Elliptische Überblendung

Analog zum vorherigen Beispiel wird auch hier ein Pfad als Clipping-Bereich benutzt. Lediglich zwei Varianten erscheinen sinnvoll:

- ▶ von innen nach außen
- ▶ von außen nach innen

Die Richtung wird daher in der booleschen Variablen `FromInside` festgehalten und kann über zwei `RadioButtons` vorgegeben werden. Je nach Richtung wird entweder das Äußere oder das Innere der Ellipse als Clipping-Bereich definiert.

Die Ellipse selbst hat als Mittelpunkt den Mittelpunkt des Zeichenbereichs. Ihre Größe wird über das umschließende Rechteck definiert. Dieses muss im Maximum größer sein als der Zeichenbereich, damit der Zeichenbereich vollständig in die Ellipse fällt. Listing 96 zeigt die vollständige Implementierung.

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Propor-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

```

Protected Overrides Sub Fade(ByVal g As _
    System.Drawing.Graphics)

    ' Ein neuer Pfad für den Clip-Bereich
    Dim path As New GraphicsPath()

    ' Mittelpunkt der PictureBox
    Dim mx As Integer = PictureBoxRect.Width \ 2
    Dim my As Integer = PictureBoxRect.Height \ 2

    ' Hilfsvariablen
    Dim clipReg As Region
    Dim x, y As Integer

    If FromInside Then
        ' Die Ellipse beginnt klein in der Mitte
        ' Ecke links oben berechnen
        x = PictureBoxRect.Width * Counter \ Steps
        y = PictureBoxRect.Height * Counter \ Steps

        ' Ellipse zu Pfad hinzufügen
        path.AddEllipse(mx - x, my - y, x * 2, y * 2)

        ' Clip-Bereich setzen
        g.SetClip(path)

    Else
        ' Die Ellipse beginnt groß außen
        ' Ecke links oben berechnen
        x = PictureBoxRect.Width * (Steps - Counter) \ Steps
        y = PictureBoxRect.Height * (Steps - Counter) \ Steps

        ' Ellipse zu Pfad hinzufügen
        path.AddEllipse(mx - x, my - y, x * 2, y * 2)

        ' Clip-Bereich anlegen
        clipReg = New Region(path)

        ' Clip-Bereich setzen
        '(das Innere der Ellipse bleibt erhalten)
        g.ExcludeClip(clipReg)

        ' Ressourcen freigeben
        clipReg.Dispose()

    End If

    ' Bild zeichnen
    g.DrawImage(NextPic, PictureBoxRect)

End Sub Listing

```

---



Eine Besonderheit gegenüber dem vorangegangenen Beispiel ist der Fall AUßEN NACH INNEN. Hier ist nicht die Ellipse der Bereich, der gezeichnet werden soll, sondern alles außerhalb der Ellipse. Um den Außenbereich zum Clipping-Bereich zu machen gibt es mehrere Möglichkeiten. Hier wird aus dem Pfad ein Region-Objekt erzeugt und mit `ExcludeClip` der Außenbereich dieser Region zum Clipping-Bereich des `Graphics`-Objektes erklärt. Alternativ können Sie auch `Region.Complement` für das Invertieren des Region-Objektes verwenden. Auch für ein Region-Objekt gilt, dass man mittels `Dispose` die verwendeten Ressourcen wieder freigeben sollte.

Einen Schnappschuss des Überblendvorgangs sehen Sie in Abbildung 39.



Abbildung 39: Eine zentrierte Ellipse als Clipping-Bereich

Alternativ können Sie die Ellipse auch durch ein Rechteck ersetzen und erhalten so einen Überblendeffekt mit einem zentrierten rechteckigen Bereich. Auch hier, besonders wenn von innen nach außen übergeblendet wird, könnten Sie das neue Bild skalieren und kontinuierlich vergrößern. Der Phantasie sind hier keine Grenzen gesetzt.

## 73 Überblendung durch zufällige Mosaik-Muster

Im letzten Clipping-Beispiel wird der Clipping-Bereich kontinuierlich um kleine rechteckige Ausschnitte erweitert, bis die ganze Bildfläche abgedeckt ist. Die Rechtecke werden in jedem Einzelschritt zufällig ausgewählt. Ihre Größe ist einstellbar. Einen Schnappschuss zeigt Abbildung 40.

Die für jeden Schritt benötigten Informationen werden in Member-Variablen der Klasse gehalten:

```
' Breite und Höhe für die Kästchen
Private FieldWidth As Integer = 10
Private FieldHeight As Integer = 10

' Liste der Kästchen
Private ClipRectangles As New ArrayList()
```

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung  
Win-  
dows  
Forms

Controls

Propor-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

```

' Clip-Region für Überblendung
Private ClipRgn As New Region()

' Anzahl der Felder pro Überblendschritt
Private FieldsPerStep As Integer

' Zufallszahlengenerator
Private RD As New Random()

```

---

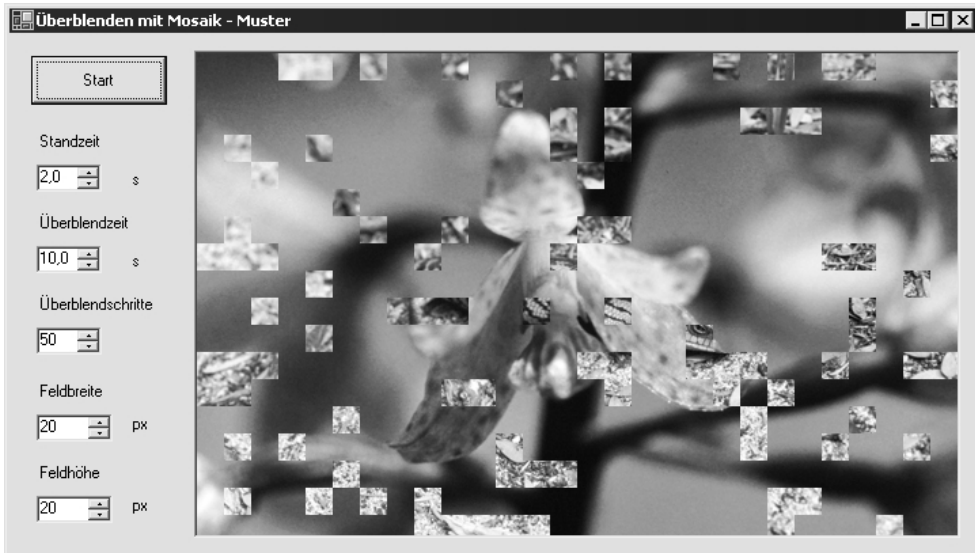


Abbildung 40: Überblendung mit Mosaik-Muster

Vor einem Überblendvorgang werden anhand der Variablen `FieldWidth` und `FieldHeight` alle zur Abdeckung der Zeichenfläche benötigten Rechtecke errechnet und der Liste `ClipRectangles` hinzugefügt. In jedem Überblendschritt wird dann eine zuvor bestimmte Anzahl (Variable `FieldsPerStep`) zufällig ausgewählter Rechtecke der Liste entnommen und dem Clipping-Bereich hinzugefügt. Am Ende des Überblendvorgangs ist die Liste leer und der Clipping-Bereich vollständig. Diese Strategie verhindert, dass der Zufallszahlengenerator Rechtecke mehrfach auswählen kann, und stellt sicher, dass alle Rechtecke gesetzt werden. In Listing 97 sehen Sie die Implementierung der Methode `Fade`.

---

```

Protected Overrides Sub Fade(ByVal g As System.Drawing.Graphics)
    If counter = 1 Then
        ' Initialisierung vornehmen

        Dim x, y As Integer

        ' Alle Rechtecke für das Clipping anlegen und in der Liste
        ' "ClipRectangles" verwalten
        Do While y < PictureBox.Height

```

---

Listing 97: Kontinuierliche Erweiterung der Clipping-Region durch zufällig ausgewählte Rechtecke

```

x = 0
Do While x < PictureBox.Width
    ClipRectangles.Add(New Rectangle(x, y, FieldWidth, _
        FieldHeight))
    x += FieldWidth
Loop
y += FieldHeight
Loop

' Berechnen, wie viele Rechtecke pro Schritt verwendet werden
' sollen
FieldsPerStep = ClipRectangles.Count \ Steps

' Clip-Region löschen
ClipRgn.MakeEmpty()

End If

Dim i, z As Integer
For i = 1 To FieldsPerStep
    ' Index eines zufällig ausgewählten Rechtecks
    z = RD.Next(ClipRectangles.Count)

    ' Rechteck der Clip-Region hinzufügen
    ClipRgn.Union(DirectCast(ClipRectangles(z), Rectangle))

    ' und aus der Liste entfernen
    ClipRectangles.RemoveAt(z)
Next

' Clipping setzen
g.Clip = ClipRgn

' Neues Bild zeichnen
g.DrawImage(NextPic, PictureBox)

End Sub

```

Listing 97: Kontinuierliche Erweiterung der Clipping-Region durch zufällig ausgewählte Rechtecke

Auch dieses Beispiel können Sie mit wenig Aufwand erweitern. Ein netter Effekt ergäbe sich, wenn die Kästchen nicht zufällig ausgewählt werden, sondern z.B. spiralenförmig oder im Zickzack gezielt gesetzt werden. Hierfür muss lediglich die Auswahl der Kästchen umprogrammiert werden. Der Rahmen kann unverändert übernommen werden.

## 74 Überblendung durch Transparenz

Im letzten Beispiel zur Überblendung von Bildern wird ein völlig anderer Weg eingeschlagen. Statt das neue Bild zu beschneiden, wird es in jedem Schritt in voller Größe über das alte Bild gezeichnet. Allerdings wird die Deckkraft kontinuierlich von völlig transparent bis deckend schrittweise gesteigert. Dadurch ergibt sich ein Effekt ähnlich der Überblendung bei Diaprojekto-

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tung  
Win-  
dows  
Forms

Controls

Propor-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

Während Sie zum Zeichnen und Füllen transparenter geometrischer Figuren mit `Graphics.Draw...` und `Graphics.Fill...` lediglich ein Pen- oder Brush-Objekt mit einer teilweise transparenten Farbe übergeben müssen, ist der Aufwand bei Bitmaps etwas größer.

Basis der Berechnungen ist der Alpha-Wert, der die Transparenz angibt. Ein Wert von Null bedeutet vollständige Transparenz, das Bild ist also nicht sichtbar, ein Wert von Eins bedeutet vollständige Deckung, so dass der Untergrund nicht mehr sichtbar ist. Um eine Bitmap transparent zu machen, benötigt man ein `ColorMatrix`-Objekt aus dem Namespace `System.Drawing.Imaging`. Hierbei handelt es sich um eine vordefinierte 5 x 5 Matrix, die für Farboperationen verwendet wird. Im Feld (3,3) kann ein Alpha-Wert festgelegt werden. Eine Instanz der Klasse `ImageAttributes` aus dem gleichen Namespace verwendet u.a. diese Matrix, um bei der Ausgabe eines Bildes mit `DrawImage` Farben und Transparenz zu verändern.

Einige der ca. 30 Überladungen der `DrawImage`-Methode akzeptieren ein `ImageAttributes`-Objekt als Parameter. Leider gibt es keine, die beide Rechtecke als `Rectangle`-Strukturen annimmt. Daher muss das `Source-Rectangle` durch vier Integer-Werte angegeben werden. Listing 98 zeigt die Implementierung.

---

```
Protected Overrides Sub Fade(ByVal g As System.Drawing.Graphics)
```

```
    ' Transparenz ermitteln
```

```
    ' 0: transparent
```

```
    ' 0..1: durchscheinend
```

```
    ' 1: deckend
```

```
    Dim alpha As Single = CSng(Counter / Steps)
```

```
    ' Farbmatrix anlegen
```

```
    Dim ColMat As New ColorMatrix()
```

```
    ColMat.Matrix33 = alpha
```

```
    ' Bildattribute anlegen
```

```
    Dim imgAtt As New ImageAttributes()
```

```
    imgAtt.SetColorMatrix(ColMat)
```

```
    ' Buffer für Zeichenoperationen
```

```
    Dim bm As New Bitmap(PictureBox.Width, PictureBox.Height)
```

```
    ' Graphics-Objekt für Buffer holen
```

```
    Dim g2 As Graphics = Graphics.FromImage(bm)
```

```
    ' Das alte Bild deckend in den Buffer zeichnen
```

```
    g2.DrawImage(Me.ActivePic, PictureBoxRect)
```

```
    ' Das neue Bild transparent in den Buffer zeichnen
```

```
    g2.DrawImage(Me.NextPic, PictureBoxRect, 0, 0, PictureBoxRect.Width, _  
        PictureBoxRect.Height, GraphicsUnit.Pixel, imgAtt)
```

```
    ' Das Ergebnis auf die PictureBox ausgeben
```

```
    g.DrawImage(bm, PictureBoxRect)
```

```
    ' Entsorgungen
```

```
    g2.Dispose()
```

---

Listing 98: Alpha-Blending mit Bitmaps

```
bm.Dispose()  
imgAtt.Dispose()
```

End Sub

*Listing 98: Alpha-Blending mit Bitmaps (Forts.)*

Da das Zeichnen von transparenten Bitmaps sehr aufwändig ist (jedes Pixel des zu zeichnenden Bildes muss mit dem entsprechenden Pixel des Hintergrundbildes verrechnet werden), dauert ein Überblendschritt entsprechend lang. In der obigen Variante muss zudem das alte Bild in jedem Schritt neu gezeichnet werden. Das führt zu einem unschönen Flackern.

Um das Flackern zu unterdrücken, wird die Zeichnung erst in einem Buffer erstellt, bevor sie auf das Graphics-Objekt der PictureBox ausgegeben wird. GDI+ stellt einige Hilfsmittel zur Verfügung, um diesen Mechanismus zu realisieren.

Benötigt wird zunächst ein Bitmap-Objekt in der Größe der Zeichnung. Dieses lässt sich leicht mit Hilfe des Konstruktors `Bitmap(int, int)` erstellen. Das Zeichnen auf diese Bitmap erfordert ein Graphics-Objekt, das man mit der statischen Methode `Graphics.FromImage` erhält. Alle Zeichenoperationen werden dann auf diesem Objekt ausgeführt.

Ist die Zeichnung komplett, dann kann die erzeugte Bitmap einfach mit `DrawImage` auf die PictureBox ausgegeben werden. Danach sollten noch die Ressourcen freigegeben werden.

Abbildung 41 zeigt eine Momentaufnahme der Überblendung. Der wirkliche Effekt lässt sich aber nur am laufenden Beispielprogramm erfassen.



Abbildung 41: Einsatz von Transparenz für die Überblendung

Statt in jedem Schritt das alte Bild zu zeichnen und das neue mit berechneter Transparenz darüber zu zeichnen, kann man auch einen anderen Weg einschlagen:

Das alte Bild wird nicht neu gezeichnet, sondern kontinuierlich mit dem neuen überdeckt. Hierfür legt man einen Alpha-Wert fest, der ungefähr dem Kehrwert der Überblendschritte entspricht.

Das Ergebnis ist zwar nicht mathematisch genau, da im letzten Überblendschritt das erzeugte Bild nicht exakt dem neuen Bild entsprechen muss, aber der Unterschied fällt kaum auf. Nach Ablauf des Überblendvorgangs ersetzt die Basisklasse `FadingBase` ja ohnehin das alte durch das neue Bild. Ob dadurch auf den Buffer verzichtet werden kann, hängt von der Leistungsfähigkeit des Rechners ab und muss in einem Versuch ermittelt werden.

## 75 Bilder verzerrungsfrei maximieren

Soll ein Bild auf einem Ausgabegerät (z.B. Fenster, `PictureBox` oder Druckerseite) mit maximaler Größe dargestellt werden, reicht es nicht aus, beim Zeichnen das Bild auf die Abmessungen des Zielrechtecks zu transformieren. Denn selten haben Bild und Ausgabefläche dasselbe Seitenverhältnis und das Bild würde verzerrt wiedergegeben.

Breite und Höhe müssen daher mit dem gleichen Skalierungsfaktor multipliziert werden, um Verzerrungen zu vermeiden. Dieser Faktor muss so bemessen sein, dass eine Seite des Bildes (Breite oder Höhe) die gleiche Ausdehnung hat wie die des Zielrechtecks, die andere Bildseite aber nicht über das Zielrechteck hinausgeht. Die Klasse `ScaleImage` (Listing 99) bietet zwei Funktionen (`DrawMaximizedPicture`), um ein Bild nach dieser Vorgabe zu skalieren.

Zur Berechnung des Skalierungsfaktors werden die Seitenverhältnisse (Breite / Höhe) des zu zeichnenden Bildes und des Ausgaberechtecks verglichen. Ob Bild und Zielrechteck Querformat (Landscape) oder Hochformat (Portrait) haben, spielt dabei keine Rolle. Ist das Seitenverhältnis des Bildes größer als das des Zielrechtecks, dann muss die Bildbreite auf die Breite des Rechtecks skaliert werden. Die Höhe des gezeichneten Bildes ist dadurch kleiner oder maximal gleich der Höhe des Zielrechtecks. Ist das Verhältnis umgekehrt, wird die Bildhöhe auf die Höhe des Rechtecks umgerechnet.

Über den Parameter `align` kann die Ausrichtung des Bildes innerhalb des angegebenen Rechtecks eingestellt werden. Analog zur Enumeration `StringAlignment` werden die Werte `Near`, `Center` und `Far` angeboten, um das Bild bündig links (bzw. oben), horizontal (bzw. vertikal) zentriert oder bündig rechts (bzw. unten) zu zeichnen. Die Bezeichnungen `Near` und `Far` wurden gewählt, um eine Unterscheidung von horizontaler und vertikaler Ausrichtung zu umgehen und die Ausrichtung evtl. später länderspezifisch anzupassen.

---

```
Public Class ScaleImage

    Public Enum Alignment
        Near      ' Links bzw. oben
        Center    ' Zentriert
        Far       ' Rechts bzw. unten
    End Enum

    ' Bild in maximaler Größe unverzerrt zeichnen
    ' g:           Das Graphics-Objekt, auf das gezeichnet werden soll
    ' picture:     Das zu zeichnende Bild
    ' destRect:    Zielrechteck, in das gezeichnet werden soll
    ' align:       Ausrichtung
    ' Rückgabewert: verwendeter Skalierungsfaktor
    Public Shared Function DrawMaximizedPicture(ByVal g As Graphics _
```

---

Listing 99: `ScaleImage` ermöglicht die verzerrungsfreie maximierte Darstellung eines Bildes in einem vorgegebenen Rechteck

```

, ByVal picture As Image, ByVal destRect As Rectangle, _
ByVal align As Alignment) As Double

' Abmessungen des Bildes und des Zielrechtecks
Dim pw As Integer = picture.Width
Dim ph As Integer = picture.Height
Dim dw As Integer = destRect.Width
Dim dh As Integer = destRect.Height

' Ungültige Abmessungen ausschließen
If ph <= 0 Or dh <= 0 Or pw <= 0 Or dw <= 0 Then
    Throw New ArgumentException( _
        "Breiten und Höhen müssen größer sein als Null")
End If

' Skalierungsfaktor
Dim scaleFactor As Double

' Seitenverhältnisse vergleichen
If pw / ph > dw / dh Then
    ' Breite maximieren
    scaleFactor = dw / pw
Else
    ' Höhe maximieren
    scaleFactor = dh / ph
End If

' Offset für Ausrichtung im Rechteck berechnen
' Ausgangswert für "Near"
Dim x As Integer = destRect.Left
Dim y As Integer = destRect.Top

' Ausrichtung berücksichtigen
Select Case align
    Case Alignment.Center
        x += CInt((dw - scaleFactor * pw) / 2)
        y += CInt((dh - scaleFactor * ph) / 2)
    Case Alignment.Far
        x += CInt(dw - scaleFactor * pw)
        y += CInt(dh - scaleFactor * ph)
End Select

' Zielrechteck bestimmen
' Gleicher Skalierungsfaktor für Breite und Höhe
Dim r As New Rectangle(x, y, _
    CInt(scaleFactor * pw), CInt(scaleFactor * ph))

' Bild in diesem Rechteck zeichnen
g.DrawImage(picture, r)

```

Listing 99: ScaleImage ermöglicht die verzerrungsfreie maximierte Darstellung eines Bildes in einem vorgegebenen Rechteck (Forts.)

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nen**Bild-  
bearbei-  
tung**Win-  
dows  
Forms

Controls

Proper-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

---

```

' Zur Info den Skalierungsfaktor zurückgeben
Return scaleFactor

End Function

' Bild in maximaler Größe unverzerrt zeichnen
' g: Das Graphics-Objekt, auf das gezeichnet werden soll
' picture: Das zu zeichnende Bild
' Rückgabewert: verwendeter Skalierungsfaktor
Public Shared Function DrawMaximizedPicture(ByVal g As Graphics _
, ByVal picture As Image) As Double

' Parameter aus Graphics-Objekt ermitteln
Return DrawMaximizedPicture(g, picture, _
Rectangle.Round(g.VisibleClipBounds), Alignment.Center)

End Function

End Class

```

---

*Listing 99: ScaleImage ermöglicht die verzerrungsfreie maximierte Darstellung eines Bildes in einem vorgegebenen Rechteck (Forts.)*

Die zweite Variante von `DrawMaximizedPicture` benötigt lediglich die Referenz des `Graphics`-Objektes und des auszugebenden Bildes. Sie ermittelt das Zielrechteck aus der Eigenschaft `VisibleClipBounds` des `Graphics`-Objektes und zeichnet das Bild zentriert in diesem Rechteck.

Ein kleines Beispielprogramm auf der CD zeigt die verschiedenen Möglichkeiten. Abbildung 42 und Abbildung 43 zeigen zwei Darstellungen für ein Bild im Querformat, Abbildung 44 und Abbildung 45 für ein Bild im Hochformat. Abbildung 46 demonstriert die maximierte Darstellung in einem Rechteck. Die Steuerung der Ausgabe erfolgt über ein Kontextmenü des Beispielfensters, die eigentliche Bildausgabe sehen Sie in Listing 100. Nicht abgedruckt sind die Member-Variablen für die verschiedenen Einstellungen und die Ereignis-Handler für das Kontextmenü. Sie finden sie gut kommentiert im Sourcecode des Beispielprogramms auf der CD.

---

```

Protected Overrides Sub OnPaint(ByVal e As System.Windows.Forms.PaintEventArgs)

```

```

' Skalierung im Rechteck oder Vollbild?
If PictInRect Then
' Rechteck berechnen (50 Pixel Rand)
Dim r As New Rectangle(50, 50, Me.ClientSize.Width - _
100, Me.ClientSize.Height - 100)

' Bild in diesem Rechteck zeichnen
ScaleImage.DrawMaximizedPicture(e.Graphics, _
Pictures(PictureIndex), r, PictureAlignment)

' Umrisse des Rechtecks zeichnen
e.Graphics.DrawRectangle(New Pen(Color.DarkBlue, 4), r)
Else

```

---

*Listing 100: Beispiel für die maximierte Bildausgabe*



```

' Bild auf Fenster maximieren
ScaleImage.DrawMaximizedPicture(e.Graphics, _
    Pictures(PictureIndex), Me.ClientRectangle, _
    PictureAlignment)
End If

```

```
End Sub
```

Listing 100: Beispiel für die maximierte Bildausgabe (Forts.)



Abbildung 42: Bild im Querformat, Breite maximiert, zentrierte Ausgabe

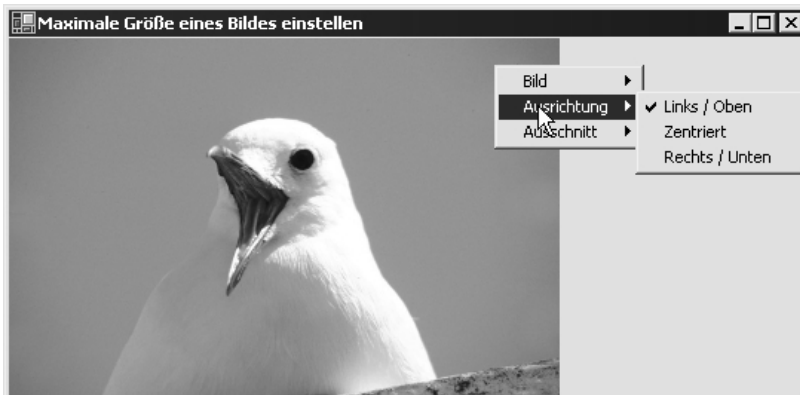


Abbildung 43: Bild im Querformat, Höhe maximiert, Ausgabe linksbündig

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

**Bild-  
bearbei-  
tung**

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes



Abbildung 44: Bild im Hochformat, Breite maximiert, zentrierte Ausgabe



Abbildung 45: Bild im Hochformat, Höhe maximiert, Ausgabe rechtsbündig



Abbildung 46: Bild im Querformat, Höhe maximiert, zentrierte Ausgabe im Rechteck

## 76 Blockierung der Bilddatei verhindern

Ein bekanntes Problem beim Anlegen eines Bitmaps mit Hilfe des Bitmap-Konstruktors, der einen Dateipfad entgegennimmt, ist, dass die Datei bis zur Zerstörung des Bitmap-Objektes für Schreib- und Löschzugriffe gesperrt ist:

```
Dim pic As Image = New Bitmap("ThePicture.bmp")
```

Muss man diese Blockierung vermeiden, weil z.B. andere Programme Lese- und Schreibzugriff auf die Datei haben sollen, dann kann ein anderer Konstruktor der Bitmap-Klasse herangezogen werden:

```
' Stream öffnen
Dim sr As New System.IO.FileStream("ThePicture.bmp ", _
    IO.FileMode.Open)

' Bitmap-Objekt anlegen
Pictures(0) = New Bitmap(sr)

' Stream schließen
sr.Close()
```

Nicht der Pfad wird übergeben, sondern ein geöffneter Stream, der nach der Instanzierung des Bitmaps wieder geschlossen wird. Die Datei kann anschließend gelöscht oder überschrieben werden.

## 77 Ordnerauswahl mit Miniaturenansicht der enthaltenen Bilder

Eine Miniaturenansicht, wie sie der Windows-Explorer für Bilder zur Verfügung stellt, lässt sich selbstverständlich auch mit .NET-Mitteln programmieren. Dazu benötigt man ein Steuerelement für die Ordnerauswahl, wie z.B. in Rezept [7.13](#) ((Steuerelement für Verzeichnisauswahl)) vorge-

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

**Bild-  
bearbei-  
tung**

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

stellt und ein Steuerelement zum Anzeigen der Miniaturen (im Englischen meist Thumbnails genannt). Trennt man beide Steuerelemente durch einen Splitter, hat man die gleiche Fensterkonstruktion wie beim Windows-Explorer. Abbildung 47 zeigt das Resultat dieses Rezeptes.



Abbildung 47: Miniaturenvorschau der Bilder eines ausgewählten Ordners

Im FolderBrowser-Control (links) kann im Dateisystem ein beliebiger Ordner ausgewählt werden, dessen enthaltene Bilder auf der rechten Seite in Tastenform dargestellt werden. Für jede Miniatur werden der Dateiname und die Dateigröße angezeigt. Mit Hilfe des Splitters zwischen beiden Steuerelementen kann die Aufteilung der linken und rechten Fläche vom Anwender individuell eingestellt werden. Reicht der Platz nicht für die Anzeige aller Miniaturen aus, wird automatisch eine Scrollbar bereitgestellt. Mit einem Mausklick auf eine Taste kann eine frei programmierbare Aktion ausgelöst werden.

Der erste Ansatz zur Lösung einer solchen Aufgabe ist wahrscheinlich, bei Auswahl eines Ordners eine ausreichende Anzahl von Tasten (`System.Windows.Forms.Button`) zu generieren und deren Image-Eigenschaften die entsprechenden Bilder zuzuweisen. Allerdings ergeben sich hier einige Schwierigkeiten:

- Die Miniaturen werden verzerrt, weil die Seitenverhältnisse nicht passen
- Der Aufbau der Seite kann bei großen Bilddateien, insbesondere bei großen komprimierten Bildern, zu einem Geduldsspiel werden
- Während des Seitenaufbaus lässt sich das Fenster nicht bedienen. Beim Laden großer Bilder hilft auch kein `DoEvents`
- Tasten vom Typ `System.Windows.Forms.Button` werden wie Fenster behandelt, belegen also umfangreiche GDI-Ressourcen

Es sind also einige andere Ansätze gefragt, die die genannten Probleme umgehen. Statt für jede Bildtaste ein Steuerelement zu verwenden, können die Bilder auch einfach auf ein Panel gezeichnet werden. Die Techniken hierzu wurden in den vorangegangenen Rezepten erläutert. Lediglich die Umrandung der Taste gibt den visuellen Eindruck, ob die Taste in der unteren oder oberen Position ist. Diese Darstellung lässt sich leicht mit `ControlPaint.DrawBorder3D` realisieren. Die Mausereignisse auf die virtuellen Tasten abzubilden ist dann auch kein Problem mehr.

Lange Bildladezeiten lassen sich nicht vermeiden. Allerdings kann man das Fenster bedienbar halten, wenn das Laden der Bilder im Hintergrund durch einen zweiten Thread vorgenommen wird. Der Anwender sieht, dass die Miniaturen nach und nach angezeigt werden, kann aber währenddessen bereits Bilder auswählen oder das Fenster scrollen. Abbildung 48 zeigt einen Ausschnitt der Miniaturenansicht, bei dem die Bilder noch nicht vollständig geladen worden sind.



Abbildung 48: Laden der Bilder in einem Hintergrund-Thread

Die Verwaltung der Dateinformationen sowie das Laden und Zeichnen der Bildtasten wird in die Klasse `Thumbnail` (Listing 101) ausgelagert. Der einzige Konstruktor der Klasse nimmt den Pfad der Bilddatei sowie die Tastengröße entgegen, liest Dateinformationen und speichert die Daten in geschützten Member-Variablen. Für die Ansicht der Bilder legt er ein `Bitmap` in passender Größe an, ohne jedoch das Bild selbst zu laden. Zur grafischen Darstellung wird später die Methode `Paint` aufgerufen, die zusätzlich zur Referenz des `Graphics`-Objektes Argumente für die Formatierung der

Schriften, die Position der Taste sowie deren Zustand (gedrückt, nicht gedrückt) übergeben bekommt. Letzteres wird durch die Konstanten `Border3DStyle.Sunken` bzw. `Border3DStyle.Raised`, die `ControlPaint.DrawBorder3D` als Parameter übergeben werden, visualisiert.

---

```
Public Class Thumbnail

    ' Dateiinformationen zum Thumbnail
    Public ReadOnly FInfo As FileInfo

    ' Bildspeicher
    Protected ReadOnly PicBuf As Bitmap

    ' Bild bereits geladen?
    Public PictureLoaded As Boolean = False

    ' Größenangaben
    Protected SizeInner As Integer
    Protected SizeOuter As Integer
    Protected BorderWidth As Integer

    ' Taste oben oder unten
    Public Enum ButtonState
        Up
        Down
    End Enum

    ' ctor
    Public Sub New(ByVal filepath As String, ByVal size As Integer)

        ' Angegebene Größe betrifft die Außenmaße
        SizeOuter = size

        ' Rahmen berücksichtigen
        BorderWidth = 2
        SizeInner = size - BorderWidth * 2

        ' Dateiinformationen lesen
        FInfo = New FileInfo(filepath)

        ' Bitmapspeicher nur anlegen, noch kein Bild laden!
        PicBuf = New Bitmap(SizeInner, SizeInner)

    End Sub

    ' Zeichnen des Thumbnails
    Public Sub Paint(ByVal g As Graphics, ByVal font As Font, _
        ByVal format As StringFormat, ByVal location As Point, _
        ByVal state As ButtonState)

        ' Zeichenposition
        Dim x As Integer = location.X
```

---

Listing 101: Kapselung der Methoden zum Laden der Bilder und Zeichnen der Tasten in der Klasse `Thumbnail`

```

Dim y As Integer = location.Y

' Rahmen für Taste oben oder unten zeichnen
If state = ButtonState.Up Then
    ControlPaint.DrawBorder3D(g, x, y, SizeOuter, SizeOuter, _
        Border3DStyle.Raised)
Else
    ControlPaint.DrawBorder3D(g, x, y, SizeOuter, SizeOuter, _
        Border3DStyle.Sunken)
End If

' Wenn das Bild bereits geladen ist, dieses auch zeichnen
If PictureLoaded Then g.DrawImage(PicBuf, x + BorderWidth, _
    y + BorderWidth, SizeInner, SizeInner)

' Dateiname und -größe unter dem Thumbnail ausgeben
g.DrawString(System.IO.Path.GetFileName(FInfo.FullName), _
    font, Brushes.Black, New RectangleF(x, y + SizeOuter, _
    SizeOuter, 15), format)

Dim t As String = String.Format("{0} KB", FInfo.Length _
    \ 1024)

g.DrawString(System.IO.Path.GetFileName(t), font, _
    Brushes.Black, New RectangleF(x, y + SizeOuter + 15, _
    SizeOuter, 15), format)

End Sub

' Bitmap laden
Public Sub LoadPicture()

    ' Bitmap aus Datei laden
    Dim bmp As New Bitmap(FInfo.FullName, False)

    ' Auf Zeichenpuffer zeichnen (maximiert auf Thumbnailfläche)
    Dim g As Graphics = Graphics.FromImage(PicBuf)
    ScaleImage.DrawMaximizedPicture(g, bmp)

    ' Ganz wichtig: Ressourcen freigeben
    g.Dispose()
    bmp.Dispose()

    ' Zustand merken
    PictureLoaded = True

End Sub

End Class

```

Listing 101: Kapselung der Methoden zum Laden der Bilder und Zeichnen der Tasten in der Klasse Thumbnail (Forts.)

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

**Bild-  
bearbei-  
tung**

Win-  
dows  
Forms

Controls

Propert-  
yGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

Ist das eigentliche Bild bereits geladen, wird es von der Methode `Paint` gezeichnet. Ansonsten bleibt das Feld leer. Abschließend werden Dateiname und -größe als Text unter der Bildtaste gezeichnet.

Geladen wird das zugehörige Bild in der Methode `LoadPicture`. Mit `ScaleImage.DrawMaximizedPicture` (siehe Rezept 5.9 ((Bilder verzerrungsfrei maximieren))) wird es unverzerrt in maximaler Größe auf das im Konstruktor angelegte Bitmap gezeichnet und steht danach für die Ausgabe mittels `Paint` zur Verfügung.

Auf Seite des Fensters (`ThumbNailShow`) beginnt der Ablauf mit Auswahl eines Ordners (Listing 102). In der Methode `CreateThumbnails` wird zunächst für den ausgewählten Ordner untersucht, welche Bilddateien enthalten sind. Für verschiedene Dateitypen, hier Bitmap und JPG-Komprimierungen, werden mit Hilfe von `GetFiles` alle Dateien ermittelt und der Auflistung `picFiles` hinzugefügt. Möchten Sie weitere Dateitypen berücksichtigen, müssen sie lediglich diese Abfragen für andere Dateiendungen erweitern.

Das Laden der Bilder erfolgt wie bereits erwähnt im Hintergrund über einen zweiten Thread, der über die Variable `LoadPictureThread` gesteuert werden kann. Erst wenn alle Bilder geladen worden sind, ist dieser Thread beendet. Zu berücksichtigen ist hierbei, dass der (ungeduldige) Anwender einen neuen Ordner anwählt, bevor die Bilder des aktuellen Ordners geladen worden sind. In diesem Fall muss der zuvor gestartete Thread beendet werden, was hier durch den Aufruf von `Abort` umgesetzt wird.

---

```
Private Sub FolderBrowserA1_FolderSelectionChanged(ByVal sender _
    As Object, ByVal e As GuiControls.FolderBrowserA.EventArgs) _
    Handles FolderBrowserA1.FolderSelectionChanged

    ' Thumbnails anlegen
    CreateThumbnails()
End Sub

Protected Sub CreateThumbnails()

    Dim fn As String

    ' Liste der Dateipfade
    Dim picFiles As New ArrayList()

    Try

        ' Verfügbare Bilddateien für dieses Verzeichnis abfragen
        ' Bei Bedarf weitere Abfragen mit anderen Extensions
        ' hinzufügen
        picFiles.AddRange(Directory.GetFiles( _
            FolderBrowserA1.Path, "*.bmp"))
        picFiles.AddRange(Directory.GetFiles( _
            FolderBrowserA1.Path, "*.jpg"))

        ' Wenn der Thread noch läuft, weil noch nicht alle Bilder
        ' der letzten Verzeichnisauswahl geladen worden sind,
        ' diesen Thread beenden
```

---

Listing 102: Start des Bildaufbaus nach Auswahl eines Verzeichnisses



```

If Not LoadPictureThread Is Nothing Then
    LoadPictureThread.Abort()
End If

' Liste für aktuelles Verzeichnis leeren
ThumbsOnThisPage.Clear()

' Anordnung (Zeilen, Spalten) berechnen
ArrangeThumbnails(picFiles.Count)

Dim tn As Thumbnail

' Für alle gefundenen Bilddateien
For Each fn In picFiles

    ' Gibt es schon einen Thumbnail zu dieser Datei?
    If AllThumbs.ContainsKey(fn) Then
        ' dann den vorhandenen verwenden
        tn = DirectCast(AllThumbs(fn), Thumbnail)
    Else
        ' sonst einen neuen anlegen
        tn = New Thumbnail(fn, ThumbnailSize)
        ' und der Gesamtliste hinzufügen
        AllThumbs.Add(fn, tn)
    End If

    ' Thumbnail der aktuellen Seite zuordnen
    ThumbsOnThisPage.Add(fn, tn)

    ' Wenn das Verzeichnis viele Dateien hat, kann es
    ' etwas dauern. Deswegen MessageLoop anstoßen.
    Application.DoEvents()
Next

' Thread starten, der für die einzelnen Thumbnails die
' Bilder liest
LoadPictureThread = New Thread(AddressOf LoadPictures)
LoadPictureThread.IsBackground = True
LoadPictureThread.Start()

Catch ex As Exception
    MessageBox.Show(ex.Message)
End Try

' Panel neu zeichnen
PNLThumbs.Refresh()

End Sub

```

Listing 102: Start des Bildaufbaus nach Auswahl eines Verzeichnisses (Forts.)

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

**Bild-  
bearbei-  
tung**

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

Mit `ArrangeThumbnails` wird die Anordnung der Bildtasten berechnet. Danach werden in einer Schleife die `Thumbnail`-Instanzen für die Bilder angelegt. Zwei Auflistungen, `ThumbsOnThisPage` (`SortedList`) und `AllThumbs` (`HashTable`) speichern die Referenzen der `Thumbnail`-Objekte. `ThumbsOnThisPage` speichert alle zur aktuellen Seite gehörenden, `AllThumbs` alle seit Programmstart geladenen. Wird ein Ordner wiederholt ausgewählt, müssen die Thumbnails nicht neu generiert werden, sondern werden der Liste `AllThumbs` entnommen. So lässt sich bei vertretbarem Speicheraufwand die Ladezeit bei abermaliger Verzeichnisauswahl ganz erheblich beschleunigen.

Abschließend wird der Hintergrund-Thread gestartet und das Panel zum Neuzeichnen veranlasst.

Anhand der Konstanten `ThumbnailSize` (Seitenlänge der Bildtasten), `ThumbnailSpaceX` (Abstand in horizontaler Richtung) und `ThumbnailSpaceY` (Abstand in der Vertikalen) sowie den Abmessungen des Darstellungsbereiches werden die Parameter für die Anordnung der Bildtasten in der Methode `ArrangeThumbnails` ermittelt. Berechnet werden `ThumbNailsPerRow` (Anzahl der Bildtasten in einer Zeile) und `ThumbNailLines` (Anzahl der Zeilen). Um erforderlichenfalls die vertikale Scrollbar anzuzeigen, werden die Eigenschaften `AutoScrollMinSize` und `AutoScrollPosition` des Panels gesetzt. Die Variable `Arranging` dient als Flag, um beim Ändern der Panelgröße eine ungewollte Rekursion zu verhindern. Listing 103 zeigt die Implementierung von `ArrangeThumbnails`, Listing 104 die Definition der Member-Variablen und Konstanten.

---

```
Protected Sub ArrangeThumbnails(ByVal count As Integer)

    ' Flag setzen
    Arranging = True

    ' Anzahl Thumbnails pro Zeile
    ThumbNailsPerRow = (PNLThumbs.ClientSize.Width -
        ThumbnailSpaceX) \ (ThumbnailSize + ThumbnailSpaceX)

    ' Anzahl Zeilen
    ThumbNailLines = CInt(Math.Ceiling(count / ThumbNailsPerRow))

    ' Höhe für ScrollBar einstellen
    PNLThumbs.AutoScrollMinSize = New Size(0, ThumbNailLines * _
        ThumbnailSize + (ThumbNailLines + 1) * ThumbnailSpaceY)

    ' Scrollbar auf Anfangsposition setzen
    PNLThumbs.AutoScrollPosition = New Point(0, 0)

    ' Flag löschen
    Arranging = False

End Sub
```

---

Listing 103: Anordnung der Thumbnails berechnen

---

```
' Konstanten für die Anordnung und Größe der Thumbnails
Protected Const ThumbnailSize As Integer = 150
Protected Const ThumbnailSpaceX As Integer = 10
Protected Const ThumbnailSpaceY As Integer = 40
```

---

Listing 104: Die Member der Klasse `ThumbNailShow`

```

' Berechnete Parameter
Protected ThumbNailsPerRow As Integer
Protected ThumbNailLines As Integer

' Flag zur Vermeidung von Event-Rekursionen
Protected Arranging As Boolean = False

' Liste aller bisher geladenen Thumbnail-Objekte
Protected AllThumbs As New Hashtable()

' Liste der Thumbnail-Objekte der aktuellen Seite
Protected ThumbsOnThisPage As New SortedList()

' Font und Stringformat für Datei-Info Anzeige
Protected FontFileInfo As New Font("Arial", 10)
Protected StringFormatFileInfo As New StringFormat()

' Thread zum Laden der Bilder
Protected LoadPictureThread As Thread

' Index des angeklickten Bildes
Protected IndexOfImageClicked As Integer

```

Listing 104: Die Member der Klasse ThumbnailShow (Forts.)

**Tipp**

Ein Panel (System.Windows.Forms.Panel) kann zwar Scrollbars anzeigen, jedoch lassen sich diese nicht über das Mausehrad (Mousewheel) bedienen. Abhilfe schafft der Einsatz einer eigenen Klasse, die von ScrollableControl abgeleitet ist und im Konstruktor den Stil UserMouse einschaltet.

```

Public Class ThumbnailPanel
    Inherits ScrollableControl

    ' Controlstyle setzen, damit Mousewheel funktioniert
    Public Sub New()
        SetStyle(ControlStyles.UserMouse, True)
    End Sub

End Class

```

Zwei Hilfsfunktionen, GetPointForPicture und GetThumbnailIndex, berechnen bei gegebenem Bildindex den Anfangspunkt (linke obere Ecke) bzw. bei gegebener Koordinate den Index des zugehörigen Bildes (Listing 105).

```

Protected Function GetPointForPicture(ByVal index As Integer) As Point

    ' Zeile und Spalte berechnen

```

Listing 105: Hilfsfunktionen für Koordinaten-Bildindex-Umrechnungen

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

**Bild-  
bearbei-  
tung**  
Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

```

Dim row As Integer = index \ ThumbNailsPerRow
Dim column As Integer = index Mod ThumbNailsPerRow

' Koordinaten berechnen
Dim x As Integer = ThumbnailSpaceX + (ThumbnailSpaceX + _
    ThumbnailSize) * column

Dim y As Integer = ThumbnailSpaceY \ 2 + (ThumbnailSpaceY + _
    ThumbnailSize) * row

' Position des Scroll-Balkens berücksichtigen
y += PNLThumbs.AutoScrollPosition.Y

' Als Point-Struktur zurückgeben
Return New Point(x, y)

End Function

Protected Function GetThumbnailIndex(ByVal x As Integer, _
    ByVal y As Integer) As Integer

' Zeile und Spalte berechnen
Dim row As Integer = (y - PNLThumbs.AutoScrollPosition.Y - _
    ThumbnailSpaceY \ 2) \ (ThumbnailSpaceY + ThumbnailSize)

Dim column As Integer = (x - ThumbnailSpaceX) \ _
    (ThumbnailSpaceX + ThumbnailSize)

' Index berechnen
Dim index As Integer = column + row * ThumbNailsPerRow

' Startpunkt (links oben) des berechneten Thumbnails ermitteln
Dim p As Point = GetPointForPicture(index)

' Sicherstellen, dass sich an (x,y) kein Leerraum befindet
' Ansonsten -1 zurückgeben
If p.X > x Or p.X + ThumbnailSize < x Or p.Y > y Or p.Y + _
    ThumbnailSize < y Or index >= ThumbsOnThisPage.Count Then
    Return -1
End If

' Berechneten Index zurückgeben
Return index

End Function

```

---

Listing 105: Hilfsfunktionen für Koordinaten-Bildindex-Umrechnungen (Forts.)

Das Neuzeichnen des Panels erfolgt im Paint-Eventhandler dadurch, dass in einer Schleife für alle zur Seite gehörenden Thumbnail-Instanzen die Paint-Methode aufgerufen wird. Im Resize-Ereignis wird die Anordnung mittels ArrangeThumbnails neu errechnet und anschließend das Panel neu gezeichnet. Listing 106 zeigt die Implementierung der beiden Methoden.

```

Private Sub PNLThumbs_Paint(ByVal sender As Object, ByVal e As _
    System.Windows.Forms.PaintEventArgs) Handles PNLThumbs.Paint

    Dim g As Graphics = e.Graphics
    Dim i As Integer = 0
    Dim tn As Thumbnail

    ' Für alle Thumbnails des aktuellen Verzeichnisses
    For Each tn In ThumbsOnThisPage.Values
        ' Thumbnail zeichnen
        tn.Paint(g, FontFileInfo, StringFormatFileInfo, _
            GetPointForPicture(i), Thumbnail.ButtonState.Up)

        i += 1
    Next

End Sub

' Größe des Panels geändert
Private Sub PNLThumbs_Resize(ByVal sender As Object, ByVal e As _
    System.EventArgs) Handles PNLThumbs.Resize

    ' Ist das Flag gesetzt, wurde das Ereignis per Programm
    ' ausgelöst. In diesem Fall ignorieren
    If Arranging Then Exit Sub

    ' Anordnung neu berechnen
    ArrangeThumbnails(ThumbsOnThisPage.Count)

    ' Panel neu zeichnen
    PNLThumbs.Refresh()

End Sub

```

Listing 106: Die Paint- und Resize-Ereignis-Handler

Besondere Aufmerksamkeit gebührt der Thread-Methode `LoadPictures` (Listing 107). In einer `For Each`-Schleife wird die Auflistung `ThumbsOnThisPage` durchlaufen und für jede `Thumbnail`-Instanz, die das zugehörige Bild noch nicht geladen hat, `LoadPicture` aufgerufen. Wenn, wie oben beschrieben, der Anwender zwischenzeitlich ein anderes Verzeichnis auswählt, ändert sich die Auflistung und der für die Schleife geladene Enumerator wird ungültig. Daher wird der Thread mit `Abort` abgebrochen (siehe Listing 107). Dieser Abbruch löst in der Methode eine `Exception` aus, die abgefangen werden muss.

Innerhalb der Schleife erfolgt das Laden der Bilder synchron, das heißt, alle Bilder werden nacheinander geladen. Nachdem ein Bild vollständig geladen wurde, soll es automatisch neu gezeichnet werden, damit der Anwender den Fortschritt sehen kann. Das direkte Zeichnen innerhalb des Threads verbietet sich jedoch von selbst, da der Zugriff auf Oberflächenelemente nur dem Thread gestattet ist, der diese auch angelegt hat. Über den Umweg mit `BeginInvoke` wird daher eine Rückrufmethode, hier `CallbackPaint`, aufgerufen, die das Zeichnen übernimmt. Listing 108 zeigt diese Rückrufmethode und die zugehörige `Delegate`-Klasse.

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Proper-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

---

```

Protected Sub LoadPictures()

    Dim tn As Thumbnail
    Dim i As Integer = 0

    ' Für alle Thumbnails im aktuellen Ordner
    For Each tn In ThumbsOnThisPage.Values
        Try
            ' Laden nur, wenn das Bild nicht bereits geladen ist
            If Not tn.PictureLoaded Then
                ' Bitmap laden
                tn.LoadPicture()

                ' Thumbnail zeichnen
                PNLThumbs.BeginInvoke(New CallbackPaintDelegate( _
                    AddressOf CallbackPaint), New Object() {tn, i})

            End If

            i += 1

        Catch ex As Exception
            ' Es können Fehler beim Laden des Bildes auftreten
            Debug.WriteLine(ex.Message)
        End Try
    Next

End Sub

```

---

Listing 107: Die Thread-Methode zum Laden der Bilder im Hintergrund

---

```

Private Delegate Sub CallbackPaintDelegate( _
    ByVal tn As Thumbnail, ByVal index As Integer)

' Rückrufmethode zum Zeichnen des Thumbnails
Private Sub CallbackPaint(ByVal tn As Thumbnail, _
    ByVal index As Integer)

    ' Graphics-Objekt ermitteln
    Dim g As Graphics = PNLThumbs.CreateGraphics()

    ' Thumbnail zeichnen
    tn.Paint(g, FontFileInfo, StringformatFileInfo, _
        GetPointForPicture(index), Thumbnail.ButtonState.Up)

    g.Dispose()

End Sub

```

---

Listing 108: Delegate-Klasse und Rückrufmethode zum Zeichnen des soeben geladenen Bildes

Nun steht alles zur Verfügung, um nach Auswahl eines Ordners die Miniaturen anzuzeigen und dem Anwender die Navigation mittels vertikaler Scrollbar zu ermöglichen. Was fehlt, ist die Animation der Tasten, wenn sie mit der Maus betätigt werden.

Im `MouseDown`-Ereignis des Panels wird hierzu ermittelt, auf welche Bildtaste der Anwender geklickt hat, und der Index in `IndexOfImageClicked` gespeichert. Ist er kleiner Null, dann wurde auf einen Zwischenraum geklickt. Anderenfalls wird die Taste im gedrückten Zustand neu gezeichnet. Das Ergebnis sehen Sie in Abbildung 49.

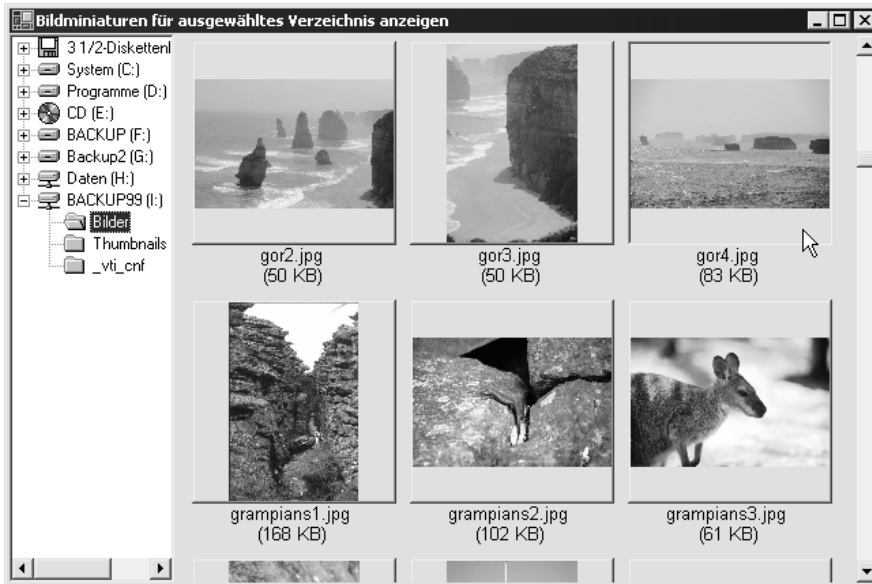


Abbildung 49: Neuzeichnen eines Thumbnails als gedrückte Taste

Nach Loslassen der Maustaste, im `MouseUp`-Ereignis, wird, wenn zuvor nicht auf einen Leerraum geklickt worden ist, die Taste erneut im Ausgangszustand gezeichnet und eine frei programmierbare überschreibbare Aktion (Methode `Action`) aufgerufen. Listing 109 zeigt die beiden Ereignis-Handler und ein Beispiel für die Methode `Action`, die hier das angeklickte Bild in einem neuen Fenster darstellt.

```
Private Sub PNLThumbs_MouseDown(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.MouseEventArgs) _
    Handles PNLThumbs.MouseDown

    ' Index des Thumbnails an der Mausposition berechnen
    Dim index As Integer = GetThumbnailIndex(e.X, e.Y)

    ' Index speichern für MouseUp-Event
    IndexOfImageClicked = index

    ' Wenn nicht auf ein Thumbnail geklickt wurde, fertig
    If index < 0 Then Exit Sub
```

Listing 109: Reaktionen auf das Anklicken der Bildtasten

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Prop-  
ertyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

```

' Thumbnail als gedrückte Taste neu zeichnen
Dim g As Graphics = PNLThumbs.CreateGraphics()
Dim tn As Thumbnail = DirectCast( _
    ThumbsOnThisPage.GetByIndex(index), Thumbnail)

tn.Paint(g, FontFileInfo, StringformatFileInfo, _
    GetPointForPicture(index), Thumbnail.ButtonState.Down)

g.Dispose()

End Sub

Private Sub PNLThumbs_MouseUp(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.MouseEventArgs) _
    Handles PNLThumbs.MouseUp

' Wenn zuvor nicht auf ein Thumbnail geklickt worden war,
' Event ignorieren
If IndexOfImageClicked < 0 Then Exit Sub

' Thumbnail als nicht gedrückte Taste neu zeichnen
Dim g As Graphics = PNLThumbs.CreateGraphics()

Dim tn As Thumbnail = DirectCast( _
    ThumbsOnThisPage.GetByIndex(IndexOfImageClicked), Thumbnail)

tn.Paint(g, FontFileInfo, StringformatFileInfo, _
    GetPointForPicture(IndexOfImageClicked), _
    Thumbnail.ButtonState.Up)

g.Dispose()

' Programmierte Aktion ausführen
Action(tn)

End Sub

' Programmierte Aktion
Protected Overridable Sub Action(ByVal thumb As Thumbnail)
' Nur zur Demo: Anzeige des Bildes in einem Fenster
MaximizedPictureBox.ShowPicture(thumb.FInfo.FullName)
End Sub

```

---

*Listing 109: Reaktionen auf das Anklicken der Bildtasten (Forts.)*

## Vorschläge für eigene Ergänzungen und Erweiterungen

Bildbearbeitungsprogramme wie Paint Shop Pro speichern einmal erzeugte Miniaturansichten in geeigneter Form (Dateien, Datenbank), um sie bei abermaliger Auswahl schneller anzeigen zu können. Die hier vorgestellte Version speichert die Bilder nur während der Laufzeit des Programms. Um auch bei weiteren Programmstarts die Thumbnails schnell laden zu können, kann man sie in einem entsprechenden Unterverzeichnis abspeichern und bei Bedarf wieder laden.



Besonders, wenn die Bilddateien mehrere Megabyte groß sind, stellt das für den Anwender eine angenehme Bereicherung des Funktionsumfangs dar. Allerdings sollte auch jeweils anhand der Dateiinformationen überprüft werden, ob die gespeicherten Miniaturen noch aktuell sind oder ob die zugehörigen Bilder zwischenzeitlich verändert oder gelöscht worden sind.

Während die oben beschriebenen Darstellungen und Ereignis-Handler zwar Mauseaktionen berücksichtigen, ist die Auswahl einer Bildtaste über die Tastatur nicht implementiert. Die Tastendrucke abzufangen ist aber nicht weiter schwierig und über die `ControlPaint`-Klasse lässt sich auch ein Rahmen zeichnen, der eine Taste als ausgewähltes Steuerelement visualisiert.

## 78 Drehen und Spiegeln von Bildern

Sicher haben Sie schon einmal gesehen, dass Bildverarbeitungsprogramme Bilder drehen und spiegeln können. Gedrehte Ausgaben erreichen Sie unter GDI+ grundsätzlich dadurch, dass Sie vor der Ausgabe das Koordinatensystem drehen:

```
g.RotateTransform(35)
g.DrawImage(...)
```

`RotateTransform` erlaubt die Drehung um einen beliebigen Winkel. Allerdings muss beachtet werden, dass die Pixel des Bildes auf die Pixel der gedrehten Fläche abgebildet werden müssen. Handelt es sich bei dem Winkel nicht um Vielfache von 90°, dann ist in jedem Fall ein Qualitätsverlust (Unschärfe) die Folge.

Rotationen um 90°, 180° oder 270° lassen sich aber bereits für das Bitmap selbst vornehmen, ohne dass es auf einem `Graphics`-Objekt gezeichnet werden muss. Die Klasse `Image` stellt hierzu die Methode `RotateFlip` bereit. Sie nimmt als Parameter eine Enumeration vom Typ `RotateFlipType` an (siehe Tabelle 12). Es gibt zwar 16 verschiedene Konstanten, jedoch bedingt durch die Kombinationen nur acht verschiedene Möglichkeiten. Abbildung 50 zeigt diese acht Variationen.

| Wert | Name                                                            | Bedeutung                                               |
|------|-----------------------------------------------------------------|---------------------------------------------------------|
| 0    | <code>Rotate180FlipXY</code><br><code>RotateNoneFlipNone</code> | Keine Drehung, keine Spiegelung                         |
| 1    | <code>Rotate270FlipXY</code><br><code>Rotate90FlipNone</code>   | Drehung um 90° nach rechts, keine Spiegelung            |
| 2    | <code>Rotate180FlipNone</code><br><code>RotateNoneFlipXY</code> | Drehung um 180°, keine Spiegelung                       |
| 3    | <code>Rotate270FlipNone</code><br><code>Rotate90FlipXY</code>   | Drehung um 90° nach links, keine Spiegelung             |
| 4    | <code>Rotate180FlipY</code><br><code>RotateNoneFlipX</code>     | Keine Drehung, horizontale Spiegelung                   |
| 5    | <code>Rotate270FlipY</code><br><code>Rotate90FlipX</code>       | Drehung um 90° nach rechts, dann horizontale Spiegelung |
| 6    | <code>Rotate180FlipX</code><br><code>RotateNoneFlipY</code>     | Keine Drehung, vertikale Spiegelung                     |
| 7    | <code>Rotate270FlipX</code><br><code>Rotate90FlipY</code>       | Drehung um 90° nach rechts, dann vertikale Spiegelung   |

Tabelle 12: Die acht Variationen für den Aufruf von `RotateFlip`

|                           |
|---------------------------|
| Basics                    |
| Datum/<br>Zeit            |
| Anwen-<br>dungen          |
| Zeich-<br>nen             |
| Bild-<br>bearbei-<br>tung |
| Win-<br>dows<br>Forms     |
| Controls                  |
| Propert-<br>yGrid         |
| Datei-<br>system          |
| Netz-<br>werk             |
| System                    |
| Daten-<br>banken          |
| XML                       |
| Wissen-<br>schaft         |
| Ver-<br>schiede-<br>nes   |

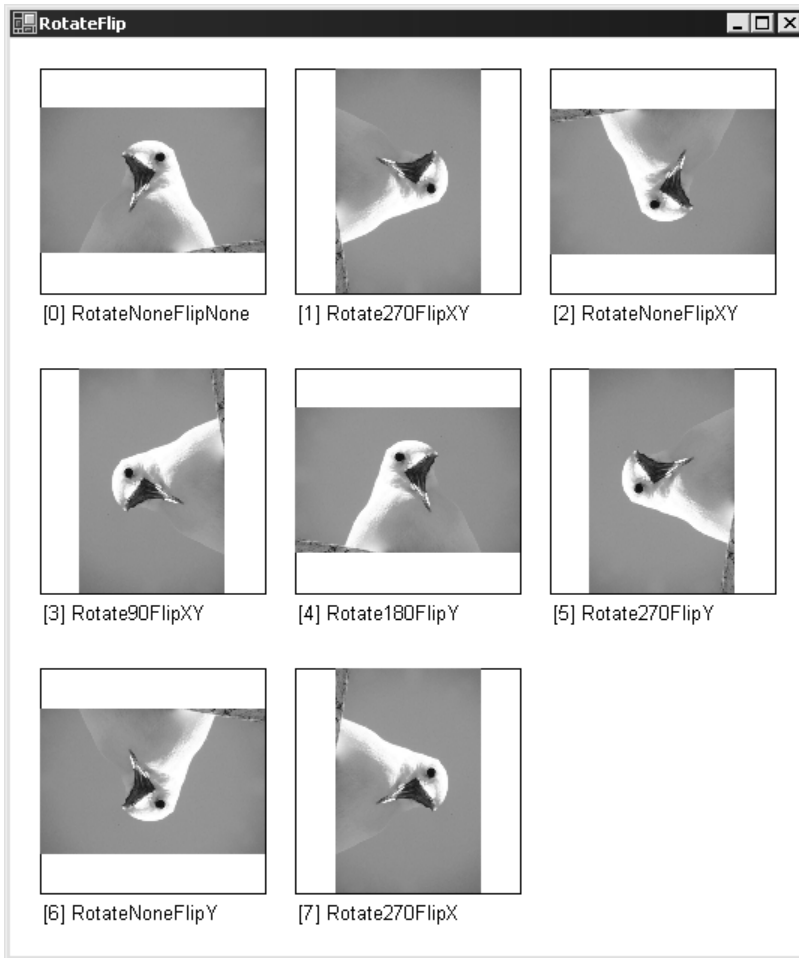


Abbildung 50: Drehen und spiegeln mit `Image.RotateFlip`

Mit dieser Methode können Sie Bilder verlustfrei drehen und spiegeln.

---

```
Dim pic As New Bitmap("...")
```

```
' Rotation mit Enum-Konstante
pic.RotateFlip(RotateFlipType.Rotate90FlipXY)
```

```
' Oder mit Zahlenwert
pic.RotateFlip(CType(3, RotateFlipType))
```

---

Ein so gedrehtes oder gespiegeltes Bild kann wie gewohnt weiter verarbeitet und verwendet werden.

# 79 Encoder für verschiedene Dateiformate zum Speichern von Bildern ermitteln

Windows stellt eine Reihe von Encodern zur Verfügung, die benutzt werden können, um Bilddateien in verschiedenen Formaten zu speichern. Über `ImageCodecInfo.GetImageEncoders` können die verfügbaren Encoder ermittelt werden. Listing 110 zeigt die Klasse `ImageEncoders`, die diese Liste abfragt und die Referenzen der Encoder-Objekte für den schnelleren Zugriff in einem Dictionary speichert.

```
Imports System.Drawing.Imaging
Imports System.Collections.Specialized

Public Class ImageEncoders

    ' Liste der verfügbaren Encoder
    Public Shared EncoderList As New HybridDictionary()

    ' Aufbau der Liste
    Shared Sub New()
        ' Liste der Encoder ermitteln
        Dim enclist() As ImageCodecInfo = _
            ImageCodecInfo.GetImageEncoders()

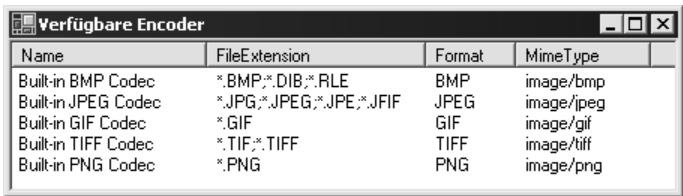
        Dim i As Integer
        For i = 0 To enclist.GetUpperBound(0)
            ' Encoder dem Dictionary hinzufügen
            ' Schlüssel ist der MimeType
            EncoderList.Add(enclist(i).MimeType, enclist(i))
        Next
    End Sub

    ' Anforderung eines bestimmten Encoders bearbeiten
    Public Shared Function GetEncoder(ByVal mimeType As String) _
        As ImageCodecInfo

        Return DirectCast(EncoderList(mimeType), ImageCodecInfo)
    End Function

End Class
```

Listing 110: Ermittlung der verfügbaren Encoder zum Speichern von Bilddateien



| Name                | FileExtension             | Format | MimeType   |
|---------------------|---------------------------|--------|------------|
| Built-in BMP Codec  | *.BMP;*.DIB;*.RLE         | BMP    | image/bmp  |
| Built-in JPEG Codec | *.JPG;*.JPEG;*.JPE;*.JFIF | JPEG   | image/jpeg |
| Built-in GIF Codec  | *.GIF                     | GIF    | image/gif  |
| Built-in TIFF Codec | *.TIF;*.TIFF              | TIFF   | image/tiff |
| Built-in PNG Codec  | *.PNG                     | PNG    | image/png  |

Abbildung 51: Diese Encoder wurden gefunden

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung  
Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

Aufgebaut wird die Liste im statischen Konstruktor. Als Schlüssel wird der `MimeType` verwendet. Über den Schlüssel kann jederzeit die Referenz eines benötigten Encoders abgefragt werden. Abbildung 51 zeigt eine `ListView`-Tabelle, die die verfügbaren Encoder und ihre wichtigsten Eigenschaften aufführt. Welche Encoder existieren, ist vom Betriebssystem und der installierten Software abhängig. Über die Eigenschaften wie `FileExtension` und `Format` lassen sich schnell Dialoge einrichten, die das Speichern von Bilddateien in verschiedenen Formaten erlauben.

Für jedes Format sind unterschiedliche Parameter zulässig (Komprimierungsverfahren, Qualität usw.). In den nächsten Rezepten werden einige ausgewählte Formate näher betrachtet.

## 80 Bilder im JPEG-Format abspeichern

Das *JPEG* (Joint Photographic Experts Group)-Format erlaubt die Speicherung von Bildern mit einer hohen Kompression. Es eignet sich gut für Fotografien, während es für Strichzeichnungen und Abbildungen mit scharfen Farbgrenzen weniger gut geeignet ist. Besonders gut erfolgt die Kompression, wenn das Bild große gleichmäßige Flächen (z.B. blauer Himmel) aufweist.

Großer Beliebtheit erfreut sich dieses Format, da es das Standard-Format für Bilder im Internet ist und dort eine große Verbreitung gefunden hat. Jeder Browser kann Bilder im JPEG-Format anzeigen, die hohe Kompression erlaubt akzeptable Übertragungsgeschwindigkeiten auch bei größeren Bildern.



Abbildung 52: Speichern eines Bildes im JPEG-Format mit verschiedenen Qualitätsstufen

Allerdings ist die Kompression nicht verlustfrei, d.h. aus einem komprimierten Bild lässt sich nicht das Ursprungsbild zurückrechnen. Beim JPEG-Format wurde die Komprimierung jedoch so gewählt, dass die Qualitätsverluste dem Betrachter oft gar nicht auffallen. Erst bei sehr hohen Komprimierungsraten treten zunehmend störende Artefakte und Unschärfen auf. Abbildung 52 zeigt vier Versionen eines Fotos, das mit dem unten angegebenen Code in verschiedenen Qualitätsstufen gespeichert wurde. Optisch wirken die Bilder mit den Qualitätsstufen 20 und 100 fast gleich, Kompressionsverluste sind kaum zu sehen. Der Unterschied in der Dateigröße ist aber deutlich: Das komprimierte Bild hat nur ca. 1/7 der Größe des unkomprimierten Bildes. Wie stark die Kompression sich auf die Dateigröße auswirkt, hängt, wie schon erwähnt, sehr von der Art des Bildes ab.

Wenn Sie Bilder im JPEG-Format speichern wollen, können Sie die Qualität selbst als einen Wert von 1 (schlechte Qualität, hohe Kompression) bis 100 (gute Qualität, keine Kompression) vorgeben. Gespeichert werden die Bilder mit folgender Überladung der `Image.Save`-Methode:

---

```
Image.Save (String, ImageCodecInfo, EncoderParameters)
```

---

Sie erwartet als Parameter neben dem Dateinamen die Angabe eines Encoders sowie zusätzlicher formatspezifischer Parameter. Der Encoder kann mit der in Rezept 5.13 ((Encoder für verschiedene Dateiformate zum Speichern von Bildern ermitteln)) vorgestellten Klasse ermittelt werden, die Parameterliste besteht hier nur aus der Qualitätsangabe:

---

```
' Encoder für JPEG-Format
Dim enc As ImageCodecInfo
enc = ImageEncoders.GetEncoder("image/jpeg")

' Parameterliste mit einem Parameter
Dim encps As EncoderParameters
encps = New EncoderParameters(1)

Dim quality As Long = ...
' Parameter definieren
encps.Param(0) = New EncoderParameter(Encoder.Quality, quality)

' Bild mit diesen Einstellungen speichern
pic.Save("...", enc, encps)
```

---

Je nach Encoder-Typ können auch noch weitere Parameter übergeben werden. Auch der JPEG-Encoder erlaubt noch mehr Parameter, die hier aber nicht weiter betrachtet werden sollen. Für zusätzliche Parameter ist das `EncoderParameters`-Objekt entsprechend zu modifizieren.

## 81 Bilder im GIF-Format speichern

Ebenfalls im Internet weit verbreitet ist das *GIF* (Graphics Interchange Format)-Format. Es erlaubt, im Gegensatz zum JPEG-Format, eine verlustfreie Komprimierung, so dass auch aus einem komprimierten Bild das Original wieder rekonstruiert werden kann. Die Kompression ist allerdings längst nicht so effektiv wie bei JPEG. Dafür eignet es sich aber auch für Strichzeichnungen und Screenshots. Eine gebräuchliche Komprimierungsart des GIF-Formats ist der LZW (Lempel Ziv Welch)-Algorithmus. Sie können ihn als Encoder-Parameter beim Speichern von Bilddateien angeben:

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Propert-  
yGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

---

```

' Encoder für JPEG-Format
Dim enc As ImageCodecInfo
enc = ImageEncoders.GetEncoder("image/gif")

' Parameterliste mit einem Parameter
Dim encps As EncoderParameters
encps = New EncoderParameters(1)
' LZW-Kompression verwenden
encps.Param(0) = New EncoderParameter(Encoder.Compression, _
    EncoderValue.CompressionLZW)
' Speichern
pic.Save("rosella.gif", enc, encps)

```

---

Der Aufbau des Codes ist der gleiche wie zum Speichern im JPEG-Format im vorigen Rezept. Lediglich der `MimeType` und die zu übergebenden Parameter sind unterschiedlich. Nach diesem Muster können Sie auch Dateien in den anderen Formaten speichern.

## 82 Thumbnails für Web-Seiten erstellen

Web-Seiten bestehen meist aus Text, vermischt mit Bildern. Oft werden die Bilder nur als Miniaturen (Thumbnails) in den Text eingebettet. Ein Klick auf ein solches Bildchen führt über einen entsprechenden Verweis dann meist zum Öffnen des Bildes in Originalgröße.

Mit Hilfe der vorangegangenen Rezepte lässt sich eine Routine erstellen, die für alle Bilder eines Verzeichnisses solche Miniaturen erstellt. Für ein einheitliches Aussehen auf der Web-Seite ist es von Vorteil, wenn die Bildgrößen einheitlich sind. Nun werden in der Regel die Bilder nicht alle mit dem gleichen Seitenverhältnis vorliegen, so dass man einen Kompromiss eingehen muss. Der kann z.B. darin bestehen, dass man eine maximale Kantenlänge vorgibt und jedes Bild so skaliert, dass seine längere Seite auf diese Kantenlänge abgebildet wird. In Listing 111 sehen Sie eine Lösung, bei der eine statische Methode (`CreateThumbnailfiles`) der Klasse `ThumbnailGenerator` für alle Dateien eines Verzeichnisses Miniaturbilder in einem anderen Verzeichnis anlegt.

---

```

Public Class ThumbnailGenerator

    ' Generieren von Thumbnails
    ' pathSourcePictures: Quellverzeichnis (z.B. Quelle\
    ' searchPattern: Suchmaske (z.B. *.jpg)
    ' pathDestinationPictures: Zielverzeichnis (z.B. Thumbnails\
    ' maxLateralLength: größte Kantenlänge in Pixel (z.B. 150)
    '
    Public Shared Sub CreateThumbnailfiles(ByVal pathSourcePictures _
        As String, ByVal searchPattern As String, _
        ByVal pathDestinationPictures As String, _
        ByVal maxLateralLength As Integer)

        Dim BmpSource As Bitmap
        Dim width, height As Integer
        Dim fn As String
    
```

---

Listing 111: Automatisches Erstellen von Thumbnails aus allen Bildern eines angegebenen Verzeichnisses

```

Dim ici As ImageCodecInfo
Dim encps As EncoderParameters
' Bildqualität einstellen
ici = ImageEncoders.GetEncoder("image/jpeg")
encps = New EncoderParameters(1)
encps.Param(0) = New EncoderParameter(Encoder.Quality, 40)

' Für alle Dateien
For Each fn In Directory.GetFiles(pathSourcePictures, _
    searchPattern)

    Dim fi As New FileInfo(fn)

    ' Datei lesen
    BmpSource = New Bitmap(fn)

    ' Hoch- oder Querformat?
    ' Seitenlängen berechnen
    If BmpSource.Width > BmpSource.Height Then
        width = maxLateralLength
        height = CInt(maxLateralLength * BmpSource.Height / _
            BmpSource.Width)
    Else
        height = maxLateralLength
        width = CInt(maxLateralLength * BmpSource.Width / _
            BmpSource.Height)
    End If

    ' Neue Bitmapinstanz in passender Größe
    Dim BmpDestination As New Bitmap(width, height)

    ' Bild unverzerrt mit ausgewähltem Interpolationsmodus
    ' auf Zielbitmap zeichnen
    Dim g As Graphics = Graphics.FromImage(BmpDestination)
    g.InterpolationMode = _
        Drawing.Drawing2D.InterpolationMode.Bicubic
    g.DrawImage(BmpSource, New Rectangle(0, 0, width, height) _
        , New Rectangle(0, 0, BmpSource.Width, _
            BmpSource.Height), GraphicsUnit.Pixel)
    g.Dispose()

    ' Bild mit diesem Encoder als jpeg-Datei speichern
    Dim fd As String = pathDestinationPictures & fi.Name
    BmpDestination.Save(fd, ici, encps)

    ' Aufräumen
    BmpDestination.Dispose()
    BmpSource.Dispose()

Next
End Sub
End Class

```

Listing 111: Automatisches Erstellen von Thumbnails aus allen Bildern eines angegebenen Verzeichnisses (Forts.)

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

**Bild-  
bearbei-  
tung**  
Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

Encoder und EncoderParameter werden einmalig für alle Dateien angelegt. In einer Schleife wird für alle Dateien des Quellverzeichnis, die dem Suchmuster entsprechen, das Bild eingelesen, so skaliert, dass die längere Seite die maximal angegebene Kantenlänge hat, und als JPEG-Datei im Zielverzeichnis abgespeichert. Bei Bedarf können Sie den Qualitätsfaktor, der hier auf 40 eingestellt wurde, auch verändern. Je nach Art des Bildmaterials muss hier ein Optimum zwischen geringer Dateigröße (für Thumbnails nicht mehr als 4 KB) und guter Darstellung gewählt werden. Auch wenn die Miniaturen nur Verweise auf die großen Originale sind, so muss für den Betrachter jedoch erkennbar sein, was dargestellt wird.

## 83 Invertieren eines Bildes

Um die Farben eines Bildes zu verändern, muss man nicht zwangsläufig die einzelnen Bildpunkte bearbeiten. Viele Aufgaben lassen sich mit der Definition einer Farbmatrix bewerkstelligen. Auch die Invertierung aller Farbinformationen lässt sich so realisieren.

Einige Überladungen der Methode `DrawImage` nehmen einen Parameter vom Typ `ImageAttributes` entgegen. Einer Instanz von `ImageAttributes` kann man mit Hilfe der Member-Funktion `SetColorMatrix` eine Matrix zuordnen, die zur Berechnung der Bildpunkte herangezogen wird. Dabei wird der Farbvektor (ARGB) eines Bildpunktes mit dieser Matrix multipliziert. Das Ergebnis ist ein neuer Farbvektor für das entsprechende Pixel des Ausgabebildes. So lassen sich mit einem einzigen Aufruf von `DrawImage` die Farbwerte aller Bildpunkte gezielt umwandeln.

Zur Durchführung der Invertierung benötigt man daher eine Matrix, die jeden Farbwert unabhängig von den anderen Farbwerten invertiert. Hierzu reicht es aus, den drei Farbmultiplikatoren auf der Hauptdiagonalen den Wert -1 zuzuweisen:

---

|    |    |    |   |   |
|----|----|----|---|---|
| -1 | 0  | 0  | 0 | 0 |
| 0  | -1 | 0  | 0 | 0 |
| 0  | 0  | -1 | 0 | 0 |
| 0  | 0  | 0  | 1 | 0 |
| 0  | 0  | 0  | 0 | 1 |

---

In den ersten drei Zeilen/Spalten stehen die Multiplikatoren für die Farbwerte Rot, Grün und Blau. Die vierte Zeile/Spalte ist für den Alpha-Werte (Transparenz) zuständig. Die fünfte Zeile/Spalte wird für die Berechnung von Offsets benötigt. Durch Multiplikation der ARGB-Vektoren mit der obigen Matrix werden somit alle Farbwerte invertiert, die Transparenz aber nicht angetastet (Faktor 1).

Wie schon in Rezept 5.8 ((Überblendung durch Transparenz)) gezeigt, können Sie eine Matrix unter Verwendung des Standard-Konstruktors der Klasse `ColorMatrix` erzeugen. Der Standard-Konstruktor legt eine Identitätsmatrix an, die auf der Hauptdiagonalen Einsen aufweist und ansonsten mit Nullen gefüllt ist. Sie können dann einzelne Werte der Matrix verändern.

Eine andere Alternative ist, dem Konstruktor eine Array-Konstruktion zu übergeben, die die Werte der Matrix enthält. Listing 112 zeigt ein Beispiel, in dem alle Werte der Matrix explizit definiert werden. Das Image-Objekt einer PictureBox wird mit `DrawImage` und der Invertierungsmatrix auf sich selbst gezeichnet. Anschließend wird die Darstellung der PictureBox aufgefrischt. So wird das zuvor dargestellte Bild invertiert.



```

Dim g As Graphics = Graphics.FromImage(PCBSample.Image)
Dim colmat As New ColorMatrix(New Single()() { _
    New Single() {-1, 0, 0, 0, 0}, _
    New Single() {0, -1, 0, 0, 0}, _
    New Single() {0, 0, -1, 0, 0}, _
    New Single() {0.0, 0.0, 0.0, 1, 0}, _
    New Single() {0.0, 0.0, 0.0, 0, 1}})

Dim imgAttr As New ImageAttributes()
imgAttr.SetColorMatrix(colmat)
g.DrawImage(PCBSample.Image, PCBSample.ClientRectangle, _
    0, 0, PCBSample.Width, PCBSample.Height, _
    GraphicsUnit.Pixel, imgAttr)
g.Dispose()
PCBSample.Refresh()

```

Listing 112: Invertierung des auf einer PictureBox dargestellten Bildes

Durch wiederholten Aufruf dieser Methode wird das Bild erneut invertiert und so wieder in seinen Ausgangszustand gebracht. Abbildung 53 zeigt das Original und das invertierte Bild.



Abbildung 53: Originalbild und invertierte Darstellung

## 84 Farbbild in Graustufenbild wandeln

Für Druckausgaben wie z. B. in diesem Buch oder für die technische Bildverarbeitung werden oft Schwarz / Weiß-Bilder benötigt. Liegt ein Farbbild vor, dann muss es in ein Graustufenbild gewandelt werden. Diese Umwandlung lässt sich genau wie im vorangegangenen Beispiel sehr elegant mit einer Farbmatrix durchführen. Jedes neue Pixel setzt sich dann wie folgt aus den Farbwerten des Ursprungspixels zusammen:

```

NeuRot = f1 * QuelleRot + f2 * QuelleGrün + f3 * QuelleBlau
NeuGrün = NeuRot
NeuBlau = NeuRot

```

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Propert-  
yGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

Da die drei Grundfarben des neuen Pixels (NeuRot, NeuGrün und NeuBlau) alle den gleichen Wert erhalten, ergeben sich ausschließlich Grautöne zwischen Schwarz und Weiß.  $f_1$ ,  $f_2$  und  $f_3$  sind dabei Wichtungsfaktoren. Bewertet man alle Farben gleich, dann ergeben sich die Faktoren zu:

---

```
f1 = 1/3
f2 = 1/3
f3 = 1/3
```

---

In der Regel verwendet man jedoch Faktoren, die die Luminanz korrigieren und ein ausgewogeneres Bild erzeugen. Rot, Grün und Blau werden dann wie folgt bewertet:

---

```
f1 = 0.3
f2 = 0.59
f3 = 0.11
```

---

In den beiden Beispielen (Listing 113 und Listing 114) wird als Quelle das Image-Objekt einer PictureBox mit `DrawImage` unter Verwendung einer Farbmatrix auf das Image-Objekt des Ziels (eine andere PictureBox) gezeichnet. Im ersten Beispiel wird eine gleich gewichtete Matrix verwendet, im zweiten Beispiel eine Matrix mit korrigierten Faktoren.

---

```
Dim g As Graphics = Graphics.FromImage(PCBDestination.Image)
Dim colmat As New ColorMatrix(New Single()() { _
    New Single() {0.33, 0.33, 0.33, 0, 0}, _
    New Single() {0.33, 0.33, 0.33, 0, 0}, _
    New Single() {0.33, 0.33, 0.33, 0, 0}, _
    New Single() {0.0, 0.0, 0.0, 1, 0}, _
    New Single() {0.0, 0.0, 0.0, 0, 1}})

Dim imgAttr As New ImageAttributes()
imgAttr.SetColorMatrix(colmat)
g.DrawImage(PCBSource.Image, PCBDestination.ClientRectangle, _
    0, 0, PCBSource.Width, PCBSource.Height, _
    GraphicsUnit.Pixel, imgAttr)
g.Dispose()
PCBDestination.Refresh()
```

---

*Listing 113: Erzeugung eines Graustufenbildes mit gleicher Gewichtung von Rot, Grün und Blau*

---

```
Dim g As Graphics = Graphics.FromImage(PCBDestination.Image)
Dim colmat As New ColorMatrix(New Single()() { _
    New Single() {0.3, 0.3, 0.3, 0, 0}, _
    New Single() {0.59, 0.59, 0.59, 0, 0}, _
    New Single() {0.11, 0.11, 0.11, 0, 0}, _
    New Single() {0.0, 0.0, 0.0, 1, 0}, _
    New Single() {0.0, 0.0, 0.0, 0, 1}})

Dim imgAttr As New ImageAttributes()
imgAttr.SetColorMatrix(colmat)
```

---

*Listing 114: Erzeugung eines Graustufenbildes mit korrigierter Wichtung der Grundfarben*

```
g.DrawImage(PCBSource.Image, PCBDestination.ClientRectangle, _
    0, 0, PCBSource.Width, PCBSource.Height, _
    GraphicsUnit.Pixel, imgAttr)
g.Dispose()
PCBDestination.Refresh()
```

Listing 114: Erzeugung eines Graustufenbildes mit korrigierter Wichtung der Grundfarben (Forts.)

Vielleicht erwarten Sie an dieser Stelle den üblichen Hinweis wie »Abbildung ... zeigt das Ergebnis«. Leider würde die Abbildung im Buch nur zeigen, wie ein Graustufenbild in ein Graustufenbild umgesetzt wird. Somit können wir Sie nur auf das Beispielprogramm auf der CD verweisen. Dort finden Sie ein Beispielfenster mit den erwähnten zwei PictureBoxen sowie zwei Tasten, mit denen Sie die beschriebenen Transformationen auslösen können. Sie können die Umwandlungen wechselseitig mit beiden Faktorkombinationen betrachten und bewerten.

## 85 Weitere Bildmanipulationen mit Hilfe der ColorMatrix

Wenn die beiden vorangegangenen Beispiele Ihr Interesse geweckt haben und Sie noch weitere Bildmanipulationen ausprobieren möchten, nutzen Sie einfach einmal das Beispielprogramm auf der CD (siehe Abbildung 54). Über 25 NumericUpDown-Felder lässt sich die Matrix elementweise verändern und das Ergebnis sofort betrachten. Viele weitere Manipulationsmöglichkeiten wie Helligkeits- und Sättigungseinstellungen, deren Beschreibung hier den Rahmen des Buches sprengen würde, lassen sich mit dem Programm testen. Das Programm selbst ähnelt den Konstruktionen der beiden vorangegangenen und soll daher nicht näher beschrieben werden, da der größte Teil des Codes der Verwaltung der Steuerelemente dient.

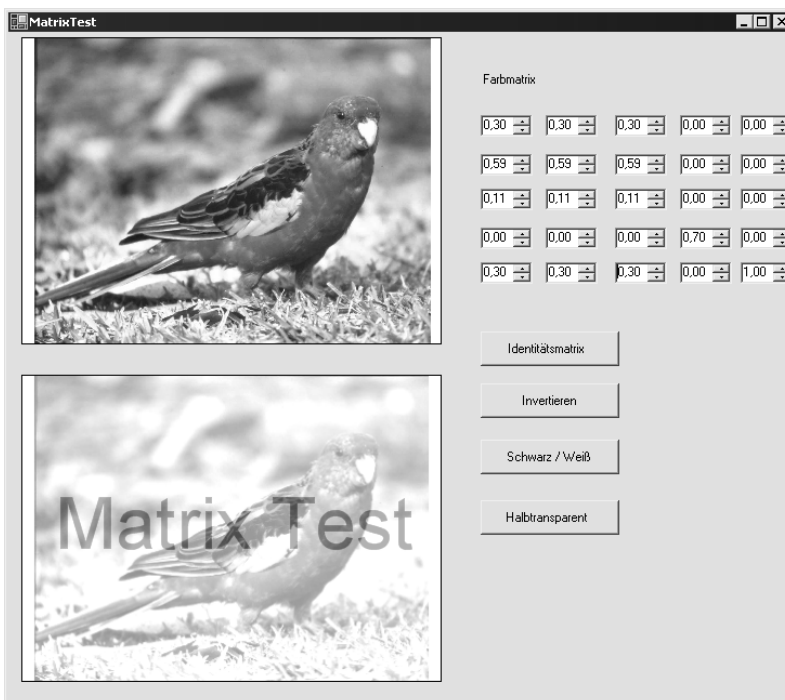


Abbildung 54: Testprogramm zur Demonstration der Farbmatrix

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung  
Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

## 86 Bitmapdaten in ein Array kopieren

Will man die Bilddaten einer Bitmap verändern, so kann man dies am effektivsten durchführen, wenn man sich die Pixelwerte in ein zweidimensionales Array kopiert. Die Funktionen `GetPixel()` und `SetPixel()` sind zwar für einzelne Pixel geeignet, doch Aktionen wie Filterberechnung zeigen schnell die Grenzen dieser Funktionen. Glücklicherweise gibt es über die Methode `BitmapData` die Möglichkeit, auf die reinen Pixel-Daten direkt zugreifen zu können. Man muss nur darauf achten, dass die Garbage-Collection nicht gleichzeitig die Daten im Hauptspeicher verschiebt, während man selber zum Beispiel über `Marshal.ReadByte` darauf zugreift!

Um bei der Filterberechnung die Übersicht zu behalten, beschäftigen wir uns hier nur mit Grautönen. Sieht man von privaten (Urlaubs-)Bildern einmal ab, ist diese Form auch die am häufigsten genutzte Form der Bilddatenanalyse. Um nun an diese Bilddaten heranzukommen, wird in dem Listingausschnitt unten eine Bitmap durch den Aufruf eines Konstruktors mit Dateiname der Bitmap aufgerufen. Nach einem Test auf das richtige Bildformat werden die Breite und Höhe des eigentlichen Bildes ermittelt. Mit diesen Werten wird ein zweidimensionales Array aus Byte-Werten erstellt. Hier kann man sich auf Byte-Werte beschränken, da das gesuchte angewandte Format mit 256 Graustufen (`PixelFormat.Format8bppIndexed`) exakt in einem Byte Platz finden.

---

```
...
' picture = Dateiname der Bitmap
bm = New Bitmap(picture)
If bm.PixelFormat = PixelFormat.Format8bppIndexed Then
    mWidth = bm.Width
    mHeight = bm.Height

    Matrix = Array.CreateInstance(GetType(Byte), mHeight, mWidth)
'...
Else
'...
End If
'...
```

---

Die Daten des eigentlichen Bildes sind hinter einer Struktur namens `BitmapData` verborgen. Sie bekommt man über die Bitmap.

---

```
' Zugriff auf die Pixeldaten ermöglichen
bmd = bm.LockBits(PixelRect, ImageLockMode.ReadOnly, _
    PixelFormat.Format8bppIndexed)
mStride = bmd.Stride
Start = bmd.Scan0
```

---

Im verwalteten System ist die Speicheradresse einer Variablen oder Datenstruktur nicht festgelegt. Daher muss bei Speicherzugriffen der entsprechende Bereich gesperrt werden. Da an dieser Stelle die Pixelwerte des Originalbildes eingelesen werden sollen, wird der entsprechende Bereich der Bitmap im Lesemodus und dem entsprechenden Pixelformat gesperrt. Die Methode `LockBits()` liefert ein `BitmapData`-Objekt zurück, dem wir den `Stride`-Wert entnehmen können. Anschließend wird die Speicheradresse des ersten Pixels in der Variablen `Start` abgelegt.

```

For i = 0 To mHeight - 1
    For j = 0 To mWidth - 1
        Matrix(i, j) = Marshal.ReadByte(Start, i * mStride + j)
    Next
Next

```

In der Bitmap werden die Pixeldaten als ein Strom von Pixelwerten abgelegt. Zum Lesen des entsprechenden Pixelwertes muss die Methode für den Zugriff auf unverwaltete Zeiger genutzt werden: `Marshal.ReadXXX`, wobei XXX für den zu nutzenden Datentyp steht. Dazu wird die überladene Methode, die im ersten Parameter den Beginn des Speicherbereiches und als zweiten Parameter den Offset in diesem Speicherbereich für das zu lesende Byte enthält.

Wichtig hier ist die Tatsache, dass die Breite der Bitmap nicht unbedingt mit der Breite einer Pixelzeile übereinstimmen muss. Pixelzeilen werden immer in Vielfachen von 4 Byte abgespeichert, so dass beispielsweise bei einer Pixelgröße von 1 Byte und einem Bild mit einer Zeilenlänge von 5 Pixeln (5 Byte) eine reale Zeilenlänge von 8 Byte vorliegt. Dieser Umstand wird mit der Veränderlichen `mStride` erfasst.

## Hinweis

Eine Bitmap wird durch die Strukturen `BITMAPFILEHEADER`, `BITMAPINFO`, `BITMAPINFOHEADER`, `RGBQUAD` beschrieben. So findet sich zum Beispiel an Offset 0x04 des `BITMAPINFOHEADER` der Wert `biWidth`, der Breite der Bitmap in Pixeln. Bei Offset 0x0e findet sich der Wert `biBitCount`, die Anzahl der Bits pro Pixel. Die Länge einer Zeile ergibt sich damit zu  $((biWidth * biBitCount + 31) / 32) * 4$ . Weitere Informationen finden Sie in der Hilfe unter den Strukturnamen.

Damit stehen die Pixelwerte des Bildes im zweidimensionalen Array `Matrix(i,j)` und können nun beliebig verändert werden.

## 87 Array in Bitmap kopieren

Hat man ein zweidimensionales Array aus Werte (mathematisch berechnet oder als Veränderung von Bilddaten), so kann man dieses Array in eine Bitmap-Struktur kopieren und somit als Bild darstellen. Bei den hier betrachteten Bildverarbeitungsfunktionen empfiehlt es sich, die neue Bitmap mit den Daten der Originalbitmap zu initialisieren. Will man eine Bitmap komplett neu aufbauen, so muss man diese Funktion so ändern, dass eine neue Bitmap mit den entsprechenden Werten erstellt wird.

```

Private Sub Copy2BMP()
    ' Kopieren der neuen Bildmatrix in die Bitmap
    Dim StartNew As IntPtr
    Dim pPixel As Int32

    NewBitmap = bm.Clone(PixelRect, PixelFormat.Format8bppIndexed)
    NewBmd = NewBitmap.LockBits(PixelRect, ImageLockMode.ReadWrite, _
        PixelFormat.Format8bppIndexed)

```

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Proper-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

Nach Deklaration der Variablen für den Speicherzugriff für die Pixelwerte wird eine neue Bitmap aus der in der Klasse vorhandenen Bitmap des Originalbildes erstellt und der Zugriff auf den Datenbereich ermöglicht.

---

```
mStride = NewBmd.Stride
StartNew = NewBmd.Scan0
pPixel = StartNew.ToInt32

Dim mPixel(mHeight * mStride) As Byte
Dim offset As Int32
```

---

In `mStride` wird die effektive Zeilenlänge gespeichert. In den beiden folgenden Zeilen liegen die Voraussetzungen für den Zugriff mit unverwalteten Zeigern. Wie oben schon beschrieben, werden die Pixel als Datenstrom abgelegt. Dieser Strom wird hier durch das eindimensionale Array `mPixel` realisiert.

---

```
For i = 0 To mHeight - 1
    For j = 0 To mWidth - 1
        Try
            offset = i * mStride + j
            mPixel(offset) = NewMatrix(i, j)
        Catch ex As Exception
            ...
        End Try
    Next
Next
```

---

An dieser Stelle wird aus der zweidimensionalen Matrix der eindimensionale Strom für die Bitmap erstellt.

---

```
Win32API.CopyArrayTo(pPixel, mPixel, mHeight * mStride)
NewBitmap.UnlockBits(NewBmd)
End Sub
```

---

Zum Schluss wird dieses Array in die Bitmap kopiert und die Speichersperre aufgehoben. Damit sind die errechneten Pixeldaten in der darzustellenden Bitmap angekommen. Dabei kommt eine Klasse zum Einsatz, die innerhalb der Bildverarbeitungs-klasse deklariert wird:

---

```
' Definition einer WIN32-Copy-Funktion
Private Class Win32API
    <DllImport("KERNEL32.DLL", EntryPoint:="RtlMoveMemory", _
        SetLastError:=True, CharSet:=CharSet.Auto, _
        ExactSpelling:=True, _
        CallingConvention:=CallingConvention.StdCall)> _
    Public Shared Sub CopyArrayTo( _
        <[In]()>, MarshalAs(UnmanagedType.I4)> ByVal hpvDest As Int32, _
```

---

```
<[In>(), Out()]> ByVal hpvSource() As Byte, _
    ByVal cbCopy As Integer)
    ' Der Prozedurenkörper ist leer
End Sub
End Class
```

Listing 115: CopyArrayTo-Klasse (Forts.)

Realisiert wird hier die Kernel-Funktion `RtlMoveMemory`. Sie findet sich in der `KERNEL32.DLL` und hat den C-Prototypen (Näheres im Windows DDK):

```
VOID
RtlMoveMemory(
    IN VOID UNALIGNED *Destination,
    IN CONST VOID UNALIGNED *Source,
    IN SIZE_T Length
);
```

Im `DllImport()` werden einige Einstellungen gesetzt, die mit der Realisierung der Funktion im Kernel durch C-Aufrufkonventionen zu tun haben. Man erkennt auch die »Relikte« von MIDL, der Microsoft Interface Definition Language.

## 88 Allgemeiner Schärfefilter

Der Schärfefilter wird in einer eigenen Klasse per DLL realisiert und kann so direkt in eigene Projekte übernommen werden.

Um das Prinzip der Programmierung kenntlich zu halten wurden keine mathematischen Optimierungen in die Rechenroutinen der DLL eingebaut. Die Konsequenz hieraus ist die natürliche Behändigkeit des Rechenverfahrens.

Die Klasse startet mit der Deklaration benötigter Veränderlicher und der Ereignisprozedur:

```
Public Class Processing
    Private mWidth As Integer          ' Breite der Bitmap
    Private mHeight As Integer         ' Höhe der Bitmap
    Private PixelRect As Rectangle     ' Rechteck zum Kopieren
    Private Matrix As Array            ' Matrix des Originalbildes
    Private NewMatrix As Array         ' Matrix des neuen Bildes
    Private bm As Bitmap               ' Bitmap des Originalbildes
    Private bmd As BitmapData          ' BitmapData des Originalbildes
    Private NewBitmap As Bitmap        ' Bitmap des neuen Bildes
    Private NewBmd As BitmapData       ' BitmapData des neuen Bildes
    Private mStride As Integer         ' Zeilenlänge der Bitmap
    Private Start As IntPtr            ' Zeiger (!) auf Pixeldaten
    Private bBitmap As Boolean          ' Neues Bild erzeugt?
    Private i As Integer               ' Laufparameter
    Private j As Integer               ' Laufparameter
```

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung  
Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

Hier werden die benötigten Variablen der alten und neuen `Bitmap` und die entsprechenden Hilfsvariablen als private Klassenmember deklariert.

Um im aufrufenden Programm einen Hinweis darauf zu haben, wie weit die Bearbeitung des Bildes bereits fortgeschritten ist, wird eine `Event-Prozedur` definiert.

---

```
Public Event Percent(ByVal percentage As Integer)
```

---

Für die Filterfunktion selber werden nach Deklaration einiger Variablen die Ausdehnung der Filter in Breite und Höhe festgelegt. Die Variable `bDim` dient der Kontrolle, ob Länge und Breite durch eine ungerade Zahl dargestellt werden können.

---

```
' Allgemeine Filterfunktion
Public Function Filter(ByVal mask(,) As Integer) As Bitmap

    Dim mMaskCnt As Integer      ' zur Berechnung Maskengröße
    Dim k As Integer             ' Laufparameter
    Dim l As Integer             ' Laufparameter
    Dim radius As Integer        ' "Radius" des Filters
    Dim FilterSumme As Double    ' :-))
    Dim Dummy As Double          ' Zwischengröße

    Dim bDim0 As Boolean = False
    Dim nDim0 As Integer = mask.GetUpperBound(0) + 1
    Dim nDim1 As Integer = mask.GetUpperBound(1) + 1
```

---

Da die Filterausdehnung für diesen Filtertyp exakt quadratisch sein muss, wird dies direkt bei Methodenstart getestet:

---

```
If nDim0 <> nDim1 Then
    Throw New System.Exception _
        ("Filter: Dim(0) und Dim(1) müssen identisch sein!")
    Return New Bitmap(1, 1)
End If
```

---

Da es sich um eine Methode handelt, die mittels Funktion realisiert wird, muss ein Rückgabewert existieren. Eine `Bitmap` der Größe 1 x 1 Pixel verhilft hier zu einem vernünftigen Ausstieg.

Eine weitere Bedingung ist, dass Weite und Höhe des Filters durch eine ungerade Zahl von Werten dargestellt werden müssen. Dies wird erreicht durch die Schleife

---

```
For mMaskCnt = 1 To 6
    If nDim0 = 2 * mMaskCnt + 1 Then
        bDim0 = True
        Exit For
    End If
Next
```

---



In `nDim0` ist die Breite des Filters festgehalten. Durch die kleine Berechnung auf der rechten Seite der `if`-Abfrage erhält man nacheinander die Werte 3, 5, 7, 9, 11, 13. Ist die Bedingung erfüllt, so stimmt die Ausdehnung des Filters und der »Radius« des Filters ist ebenfalls bestimmt (s.u.).

Sollte der Filter die Größe 13 x 13 überschreiten oder eine falsche Anzahl Werte enthalten, so kann der Filter nicht arbeiten und die Methode wird verlassen.

Sollte es noch keine neue Bitmap geben, so wird jetzt eine erstellt und die Summe der Filterwerte errechnet, da diese später benötigt wird:

---

```

If bBitmap = False Then
    create()
End If

FilterSumme = 0.0
For k = 0 To nDim0 - 1
    For l = 0 To nDim1 - 1
        FilterSumme += mask(k, l)
    Next
Next

```

---

Nun beginnt die eigentliche Arbeit des Filters. Man geht innerhalb der Matrix des Originalbildes von links oben nach rechts unten und schaut sich für jeden Bildpunkt die Umgebung in der Größe des Filters an. Innerhalb dieses Filterfensters multipliziert man den Bildwert mit dem Filterwert und summiert über das Fenster auf. Anschließend dividiert man diesen Wert durch die Summe der Filterwerte. Da dabei Werte entstehen können, die geringfügig über dem erlaubten Bereich liegen (Byte), werden die entsprechenden Werte gekappt. Dann wird dieser Wert in die Matrix des neuen Bildes geschrieben:

---

```

radius = mMaskCnt

For i = radius To mHeight - 1 - radius
    RaiseEvent Percent((i / (mHeight - 1)) * 100)
    For j = radius To mWidth - 1 - radius
        Dummy = 0.0
        For k = -mMaskCnt To mMaskCnt
            For l = -mMaskCnt To mMaskCnt
                Dummy += (Matrix(i - k, j - l) * _
                    mask(k + mMaskCnt, l + mMaskCnt))
            Next
        Next
        If FilterSumme > 0 Then
            Dummy /= FilterSumme
        End If
        If Dummy > 255.0 Then
            Dummy = 255.0
        End If
        If Dummy < 0.0 Then
            Dummy = 0.0
        End If
        NewMatrix(i, j) = CByte(Dummy)
    Next
Next

```

---

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tung  
Win-  
dows  
Forms

Controls

Propert-  
yGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

---

```
Next
Next
```

---

Wie man an den beiden äußeren FOR-Schleifen erkennt, darf die Laufvariable nicht bei 0 beginnen und bei `mHeight / mWidth` enden, da für jeden Bildpunkt die benachbarten Pixel in die Berechnung eingehen. Dies hätte in den Randbereichen zumindest programmtechnische Konsequenzen.

Bei jedem Zeilenwechsel wird errechnet, wie viel Prozent der Zeilen schon bearbeitet sind, und über ein `RaiseEvent` an das aufrufende Programm gemeldet.

Bei diesen Berechnungen können Werte entstehen, die geringfügig aus dem gültigen Wertebereich der Variablen (hier `Byte`) fallen. Aus diesem Grund werden die überlaufenden Werte abgeschnitten.

Abschließend wird die Matrix in die neue Bitmap kopiert und an das aufrufende Programm zurückgegeben:

---

```
Copy2BMP()
Return NewBitmap
End Function
```

---

## 89 Schärfe nach Gauss

Um den oben beschriebenen Filter anwenden zu können, muss im Programm vorher mittels

---

```
Imports Imaging2.Processing
```

---

die Bildverarbeitungs-klasse eingebunden und mit

---

```
Dim WithEvents mSharp As Imaging2.Processing
```

---

eine Instanz deklariert werden. Die Erstellung der Klasse geschieht dann mittels

---

```
mSharp = New Imaging2.Processing(mFileName)
```

---

Hierbei wird dem Konstruktor der Klasse der Dateiname einer zu bearbeitenden Bitmap übergeben. Diese Vorarbeiten gelten naturgemäß auch für die im Weiteren beschriebenen Filter.

Da die Klasse mit Event-Verarbeitung deklariert wurde, kann dies mit einer Funktion ähnlich Listing 116 geschehen. Die Ereignisroutine liefert den Prozentwert zurück, zu dem die gerade laufende Bildbearbeitungsroutine fertig gestellt ist. Da die Ereignisprozedur der DLL bei größeren Bildern mehrfach hintereinander den gleichen Wert liefert, wird ein nicht notwendiges Erneuern des Fortschrittbalkens – und damit ein Flackern des Balkens durch die `if`-Abfrage verhindert.

```

Public Sub progress(ByVal percentage As Integer) _
    Handles mSharp.Percent

    Static Dim perc As Integer

    If percentage = perc Then
        Return
    End If

    perc = percentage
    ProgressBar1.Value = percentage
    ProgressBar1.Update()
    ProgressBar1.Refresh()
    ProgressBar1.Invalidate()
End Sub

```

Listing 116: Bearbeiten der Fortschrittsanzeige

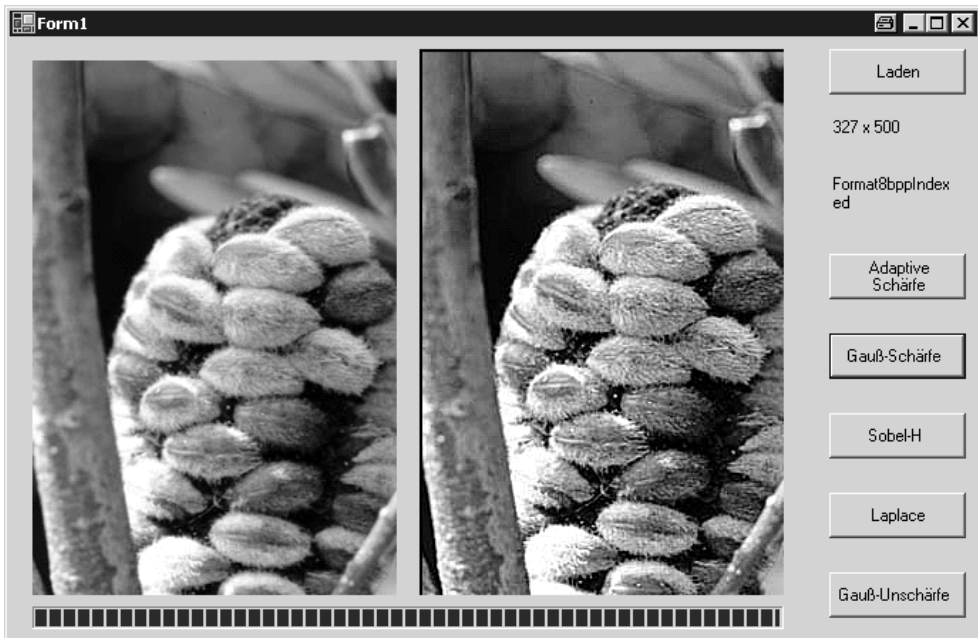


Abbildung 55: Schärfe nach dem Gauß-Verfahren

Ein Beispiel, wie man es von bekannten Programmen her kennt, stellt die Berechnung der Schärfe mit dem Gauß-Verfahren dar (Abbildung 55).

Zum Einsatz kommt hierbei die zweite Prozedur, die ein allgemeines Berechnungsverfahren zur Verfügung stellt. Die Berechnungsmaske wird der Prozedur übergeben.

```

Private Sub btnGauss_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnGauss.Click

    Dim gauss(,) As Integer = {{-2, -8, -12, -4, -2}, _
                                {-8, -32, -48, -32, -8}, _
                                {-12, -48, 686, -48, -10}, _
                                {-8, -32, -48, -32, -8}, _
                                {-2, -8, -12, -4, -2}}

    Try
        mBitmapNew = mSharp.Filter(gauss)
        pbNew.Image = New Bitmap(mBitmapNew)
    Catch ex As System.Exception
        Debug.WriteLine(ex.ToString)
    End Try
End Sub

```

Listing 117: Schärfenberechnung mit dem Gauß-Verfahren

Wie man am Beispiel von Listing 117 sieht, wird ein zweidimensionales Array erstellt, dem Filter, mit dem das Bild bearbeitet wird. In einem Anwendungsprogramm kann man so einige Filter vorgeben und zusätzlich dem Anwender die Möglichkeit bieten, eigene Filter zum Einsatz zu bringen. Der Name »Gauß-Filter« leitet sich von der mathematischen Funktion her, die zur Erstellung der Zahlen verwandt wurde, einer zweidimensionalen Gauß-Funktion.

## 90 Schärfe mittels Sobel-Filter

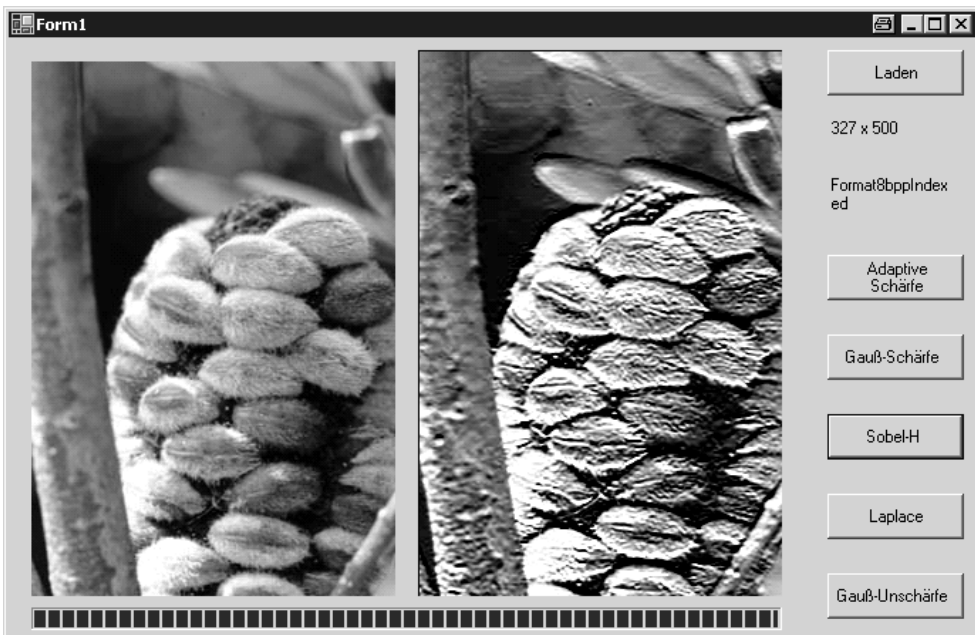


Abbildung 56: Horizontaler Sobel-Filter

```

Private Sub btnSobelH_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnSobelH.Click

    Dim Sobel_H(,) As Integer = {{-1, -2, -1}, _
                                {0, 1, 0}, _
                                {1, 2, 1}}

    Try
        mBitmapNew = mSharp.Filter(Sobel_H)
        pbNew.Image = New Bitmap(mBitmapNew)
    Catch ex As System.Exception
        Debug.WriteLine(ex.ToString)
    End Try
End Sub

```

Listing 118: Anwendung des horizontalen Sobel-Filters

Wie man vermuten kann, gibt es auch einen vertikalen Sobel-Filter. Dazu müssen die Zahlen des Filters um 90 Grad gedreht werden.

## 91 Schärfe mittels Laplace-Filter



Abbildung 57: Laplace-Filter

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Propor-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

Die Realisierung des Laplace-Filters geschieht wieder mittels eines Arrays, das wie folgt definiert wird:

---

```
Dim Laplace(,) As Integer = {{-1, -1, -1}, _
                             {-1, 9, -1}, _
                             {-1, -1, -1}}
```

---

## 92 Der Boxcar Unschärfefilter

Diese Filter-Mechanismen sind so weit tragend, dass man auch das genaue Gegenteil vom bisherigen Vorgehen erreichen kann: ein Bild unscharf rechnen. Mit einem Mittelwert-Filter wird ein Pixel durch den Mittelwert seiner angrenzenden Nachbarn ersetzt. So kann ein solcher Filter wie folgt definiert werden:

---

```
Dim boxcar(,) As Integer = {{1, 1, 1, 1, 1, 1, 1}, _
                             {1, 1, 1, 1, 1, 1, 1}, _
                             {1, 1, 1, 1, 1, 1, 1}, _
                             {1, 1, 1, 2, 1, 1, 1}, _
                             {1, 1, 1, 1, 1, 1, 1}, _
                             {1, 1, 1, 1, 1, 1, 1}, _
                             {1, 1, 1, 1, 1, 1, 1}}
```

---

Das Ergebnis dieses 7x7-Filters ist deutlich in Abbildung 58 zu sehen.

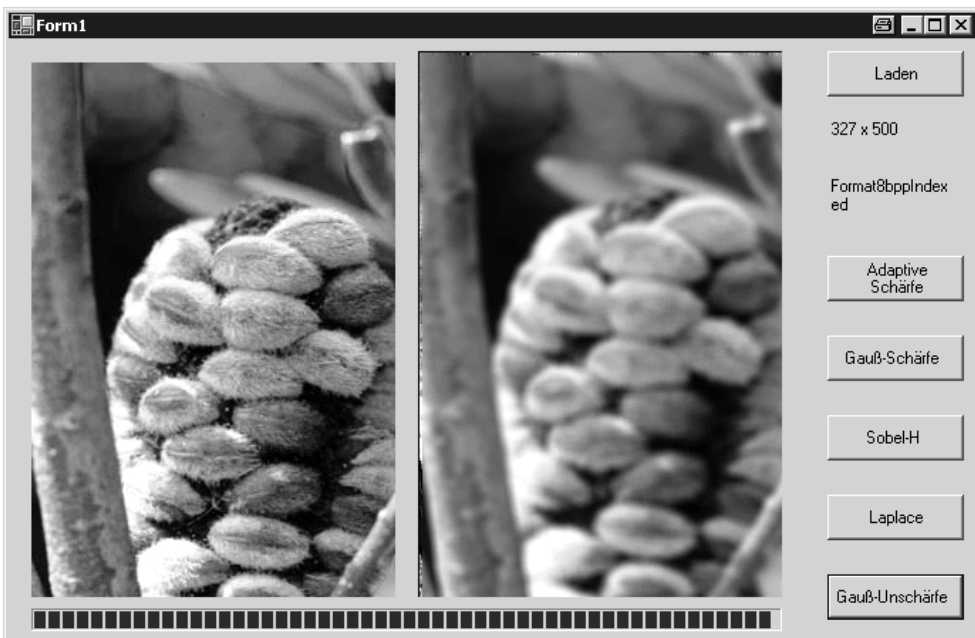


Abbildung 58: Boxcar – Unschärfe-Filter

## 93 Adaptive Schärfe

Eine sehr wirkungsvolle Methode der Schärfenberechnung ist die so genannte »Methode der adaptiven Schärfe«. In diesem Verfahren wird von jedem Pixel des Bildes die unmittelbare Umgebung betrachtet und eine »Schärfengüte« ermittelt. Je nach »Güte« wird nun für dieses Pixel die Schärfe »nachgezogen«. Dieses Verfahren gehört zu den mathematisch eher aufwändigen Verfahren der Gattung Schärfenberechnung und liefert auch nicht immer den erwünschten Erfolg. Dieser hängt sehr stark von der Struktur des Bildes ab. Einen Versuch ist dieses Verfahren aber immer wert. Das Ergebnis für das gerade geladene Bild ist in Abbildung 59 zu sehen. Man erkennt Einzelheiten auf diesem Bild, von denen man im Original nur schwer etwas erkennen kann. Dafür leidet die Ästhetik etwas ;-). Nun kann diese Prozedur nicht zaubern, sie versucht nur, Informationen zu verarbeiten, die dem menschlichen Auge in der Form nicht zur Verfügung stehen. Man sollte das Ergebnis mit der gebührenden Skepsis betrachten. So sind in Abbildung 59 die Details des Hintergrundes im oberen Bereich wohl eher zu den so genannten Artefakten zu rechnen.

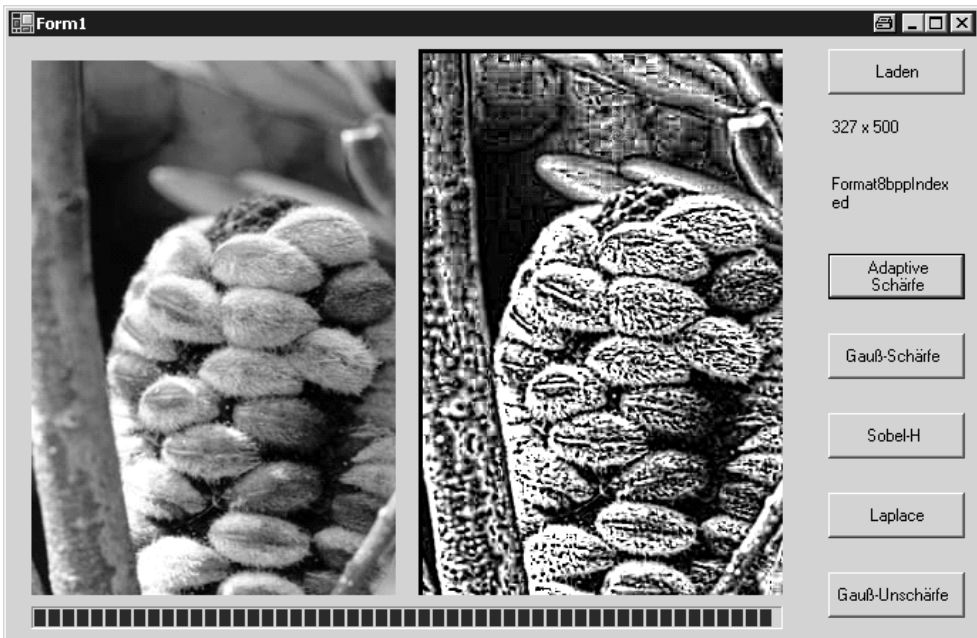


Abbildung 59: Adaptive Schärfenberechnung

Der Aufruf dieser Methode geschieht mittels

```
mBitmapNew = mSharp.Sharpen(3)
```

Der Methode wird die Größe der zu betrachtenden Pixelumgebung übergeben. Daraus folgt für den obigen Aufruf, dass die 3 x 3 Umgebung aller Pixel für die Berechnung herangezogen werden. Die Berechnung mittels des adaptiven Schärfeverfahrens ist vom Aufbau her identisch, nur die Filterberechnung ist aufwändiger:

```
' adaptiver Schärfefilter
```

---

```

For i = radius To mHeight - 1 - radius
    RaiseEvent Percent((i / (mHeight - 1)) * 100)
    For j = radius To mWidth - 1 - radius
        Pixels = 0
        PointValueMean = 0
        For k = i - radius To i + radius
            For l = j - radius To j + radius
                Pixels += 1
                Try
                    PointValueMean += Cdbl(Matrix(k, l))
                Catch ex As Exception
                    Throw New ApplicationException("Imaging2: i=" & _
                        i.ToString & "; j=" & j.ToString & "; k=" & _
                        k.ToString & "; l=" & l.ToString, ex)
                End Try
            Next
        Next
        PointValueMean /= Cdbl(Pixels)
        Difference = Cdbl(Matrix(i, j)) - PointValueMean
        Signal = 0
        For k = i - radius To i + radius
            For l = j - radius To j + radius
                Signal += (Cdbl(Matrix(k, l)) - PointValueMean) ^ 2
            Next
        Next
        Signal = Math.Sqrt(Signal / Cdbl(Pixels - 1))
        If Signal > 0 Then
            Difference *= (PointValueMean / Signal)
        End If
        Dim d As Int64 = Matrix(i, j) + CInt(Difference)
        Try
            If d > 255 Then
                d = 255
            End If
            If d < 0 Then
                d = 0
            End If
            NewMatrix(i, j) = CByte(d)
        Catch eov As System.OverflowException
            Throw New System.ApplicationException("Sharpen-Overflow: " & _
                d.ToString, eov)
        End Try
    Next
Next

```

---



# Windows Forms

In diesem Verfahren wird ein neuer Bildpunkt errechnet, in dem man die quadratischen Abweichungen vom dem Mittelwert betrachtet, der sich mit den Bildpunkten des aktuellen Filterfensters ergibt.

Der Umgang mit Fenstern hat sich mit dem Übergang von VB6 nach VB.NET wesentlich geändert. Objektorientierte Programmierung und die Klassen des Frameworks bieten Gestaltungsmöglichkeiten, wie sie noch nie zur Verfügung standen. Unregelmäßige und transparente Fenster sind nur zwei von vielen neue Varianten. Durch Vererbung lassen sich die Fenster einer Applikation einheitlich gestalten, ohne dass die gemeinsamen Teile (Code und Steuerelemente) für jedes Fenster neu definiert werden müssen.

In diesem Abschnitt finden Sie eine Reihe von Rezepten zum Umgang mit Fenstern, ergänzend dazu in Kapitel 7 ((Windows Controls)) Rezepte zur Programmierung von Steuerelementen.

## 94 Fenster ohne Titelleiste anzeigen

War es in VB6 immer noch etwas umständlich, ein Fenster ohne Titelleiste zu erzeugen (es mussten mindestens vier Eigenschaften geändert werden), so können Sie das mit VB.NET durch Setzen der Eigenschaft `FormBorderStyle` der Klasse `Form` auf den Wert `FormBorderStyle.None` erreichen. Egal, wie die Eigenschaften `Text`, `MaximizeBox`, `MinimizeBox` etc. gesetzt sind, das Fenster wird ohne Titelleiste angezeigt (siehe Abbildung 60).



Abbildung 60: Durch Setzen der `FormBorderStyle`-Eigenschaft auf `None` wird die Titelleiste unterdrückt

Ein Fenster ohne Titelleiste lässt sich nicht mehr über eine Mausaktion schließen, da mit der Titelleiste natürlich auch die entsprechenden Controls wegfallen. Achten Sie daher darauf, dass Sie dem Anwender andere Möglichkeiten zum Schließen des Fensters (z.B. eine Schaltfläche) zur Verfügung stellen. Nicht jeder kennt die Tastatur-Bedienung (z.B. `Alt+F4`).

## 95 Fenster ohne Titelleiste verschieben

Um ein Fenster unter Windows zu verschieben, zieht man für gewöhnlich die Titelleiste mit der Maus, bis das Fenster die gewünschte Position eingenommen hat. Wird die Titelleiste des Fensters unterdrückt, dann entfällt diese Möglichkeit und Sie müssen selbst das Verschieben des Fensters programmieren.

Dazu wird im `MouseDown`-Ereignis die aktuelle Position des Mauszeigers (`StartDragLocation`) sowie des Fensters (`StartLocation`) gespeichert und im `MouseMove`-Ereignis kontinuierlich neu gesetzt (Listing 119). Alle Koordinaten werden in Bildschirmkoordinaten umgerechnet. Die neue Position ergibt sich aus

---

```
NeuePosition = StartLocation + (Mausposition - StartDragLocation).
```

---

Der Anwender kann das Fenster verschieben, indem er den Mauszeiger auf einen beliebigen Punkt des Fensterhintergrunds führt, die linke Maustaste drückt und bei gedrückter Taste das Fenster zieht. Hervorzuheben ist, dass das Ziehen nur funktioniert, wenn zu Beginn der Mauszeiger über dem Fenster und nicht über einem Steuerelement steht, da sonst die Mausereignisse an das Steuerelement gesendet werden.

Soll das Fenster auch verschoben werden, wenn der Ziehvorgang beispielsweise auf einem Label beginnt, dann müssen Sie die beiden Mausereignisse dieses Steuerelementes ebenfalls an die beiden Handler binden (entweder durch Erweiterung der `Handles`-Klausel oder über `AddHandler`).

---

```
' Position des Fensters zu Beginn des Ziehvorgangs
Protected StartLocation As Point

' Mausposition zu Beginn des Ziehvorgangs
Protected StartDragLocation As Point

Private Sub NoTitlebarWindow_MouseDown(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.MouseEventArgs) _
    Handles MyBase.MouseDown

    ' Nur bei gedrückter linker Maustaste
    If e.Button = MouseButtons.Left Then
        ' Aktuelle Position des Fensters speichern
        ' (in Bildschirmkoordinaten)
        StartLocation = Me.Location

        ' Mausposition in Bildschirmkoordinaten speichern
        StartDragLocation = Me.PointToScreen(New Point(e.X, e.Y))
    End If

End Sub

Private Sub NoTitlebarWindow_MouseMove(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.MouseEventArgs) _
    Handles MyBase.MouseMove

    ' Nur bei gedrückter linker Maustaste
    If e.Button = MouseButtons.Left Then

        ' Aktuelle Mausposition in Bildschirmkoordinaten umrechnen
        Dim p As New Point(e.X, e.Y)
        p = Me.PointToScreen(p)
```

---

```
' Neue Fensterposition festlegen
Me.Location = New Point( _
    StartLocation.X + p.X - StartDragLocation.X, _
    StartLocation.Y + p.Y - StartDragLocation.Y)
```

End If

End Sub

Listing 119: Verschieben eines Fensters ohne Titelleiste mit der Maus ermöglichen (Forts.)

## 96 Halbtransparente Fenster

Ab Windows 2000 steht Ihnen die Möglichkeit offen, Fenster teilweise transparent zu gestalten. Durch Festlegung der Eigenschaft `Opacity` setzen Sie die Deckkraft des Fensters. Ein Wert von 1 entspricht 100%, also vollständig deckend, ein Wert von 0 entspricht 0%, also vollständig transparent. Zwischenwerte lassen das Fenster halbtransparent erscheinen, das heißt, die Bildpunkte des Fensters werden mit den Bildpunkten des Hintergrundes gemischt (siehe Abbildung 61).



Abbildung 61: Halbtransparentes Fenster vor Hintergrundbild

Beachten Sie bitte, dass die Zuweisung von Werten an die Eigenschaft im Code im Wertebereich von 0 .. 1 erfolgt, z. B.:

```
Me.Opacity = 0.6
```

während die Einstellung des Wertes im Eigenschaftsfenster des Designers in Prozent vorgenommen wird (Abbildung 62).

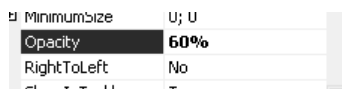


Abbildung 62: Einstellen der Deckkraft des Fensters

## 97 Unregelmäßige Fenster und andere Transparenzeffekte

Es gibt verschiedene Möglichkeiten, Fenster zu erzeugen, die nicht die üblichen rechteckigen Abmessungen besitzen. Eine Variante, die z. B. auch für die Bildüberblendmechanismen in Rezept 5.4 ff. ((Horizontal und vertikal überblenden)) verwendet wird, ist Clipping. Hierzu wird ein `Region`-Objekt festgelegt, das die Flächen, auf denen das Fenster gezeichnet werden kann, beschreibt.

Das in Abbildung 63 gezeigte Fenster ist entstanden, indem im `Load`-Ereignis eine Ellipse als Clipping-Bereich definiert wurde (siehe Listing 120). Zunächst wird ein `GraphicsPath`-Objekt angelegt, dem anschließend Bereiche (hier eine Ellipse) hinzugefügt werden. Aus diesem `GraphicsPath`-Objekt wird ein `Region`-Objekt generiert und dieses der Eigenschaft `Region` des Fensters zugewiesen.

Bei dieser Vorgehensweise können Sie den Client-Bereich des Fensters ebenso beschneiden wie die Titelleiste. Die `Region` bezieht sich auf alle Zeichenoperationen des Fensters.



Abbildung 63: Beschneiden der Fensterfläche durch Setzen des Clipping-Bereiches

---

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

    ' Pfad anlegen
    Dim path As New System.Drawing.Drawing2D.GraphicsPath

    ' Dem Pfad Bereiche hinzufügen, hier eine Ellipse
    path.AddEllipse(New Rectangle(0, 0, 300, 100))

    ' Aus dem Pfad ein neues Region-Objekt erzeugen und dem
    ' Fenster als Clipping-Bereich zuweisen
    Me.Region = New Region(path)

End Sub
```

---

Listing 120: Festlegen des Clipping-Bereichs auf eine elliptische Fläche

Als Alternative haben Sie die Möglichkeit, eine bestimmte Farbe als transparente Farbe zu definieren. Das erfolgt durch Setzen der Eigenschaft `TransparencyKey`. Für alle Bildpunkte des Fensters,

die diese Farbe besitzen, erscheint das Fenster durchsichtig, d.h. die Bildpunkte werden durch die des Hintergrundes ersetzt. Die Darstellung in Abbildung 64 zeigt ein Fenster, dessen Eigenschaft `TransparencyKey` auf `SystemColors.Control` gesetzt wurde, also der Farbe, die der normalen Hintergrundfarbe von Fenstern, Labels und Schaltflächen entspricht.

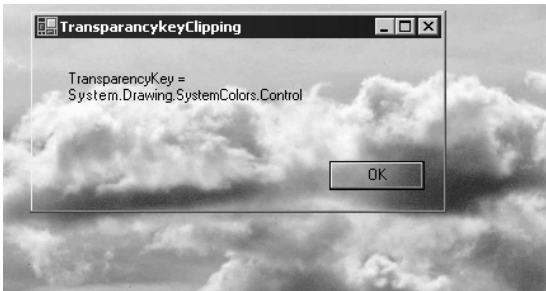


Abbildung 64: Eine Farbe als Transparenzschlüssel definieren

Diese Vorgehensweise lässt sich nun dazu einsetzen, bestimmte Bereiche des Client-Bereiches auszublenden. Dazu zeichnet man die auszublendenden Bereiche in einer Farbe, die sonst nirgends im Fenster zum Einsatz kommt, und legt anschließend diese Farbe als Transparenzschlüssel fest (siehe Abbildung 65 und Abbildung 66). Im Beispiel werden zwei Kreise mit der Farbe `Color.Red` gefüllt (Listing 121) und diese Farbe der Eigenschaft `TransparencyKey` zugewiesen.

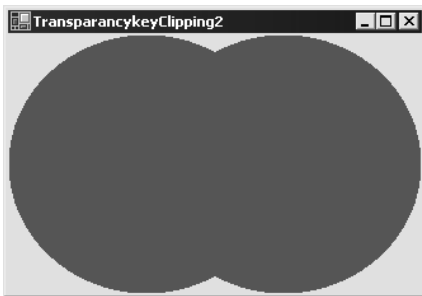


Abbildung 65: Zeichnen des auszublendenden Bereiches in einer anderen Farbe



Abbildung 66: Definition des Transparenzschlüssels zum Ausblenden des farbigen Bereiches

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Propert-  
yGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

```
Protected Overrides Sub OnPaint(ByVal e As _
    System.Windows.Forms.PaintEventArgs)

    ' Zwei überlappende Kreise zeichnen
    Dim mlx As Integer = Me.ClientSize.Width \ 3
    Dim w As Integer = Me.ClientSize.Width * 2 \ 3

    e.Graphics.FillEllipse(Brushes.Red, 0, 0, w, ClientSize.Height)
    e.Graphics.FillEllipse(Brushes.Red, mlx, 0, w, ClientSize.Height)

End Sub
```

---

*Listing 121: Zeichnen zweier Kreise mit roter Füllung, die als Transparenzschablone verwendet werden*

Dreht man die Farbgebung um, also im Beispiel die Füllung der Kreise in Grau und die Außenfläche rot und blendet zusätzlich noch die Titelleiste aus, dann erhält man ein nicht rechteckiges Fenster, ähnlich wie es durch Definition eines Clipping-Bereiches erzeugt werden kann (Abbildung 67).

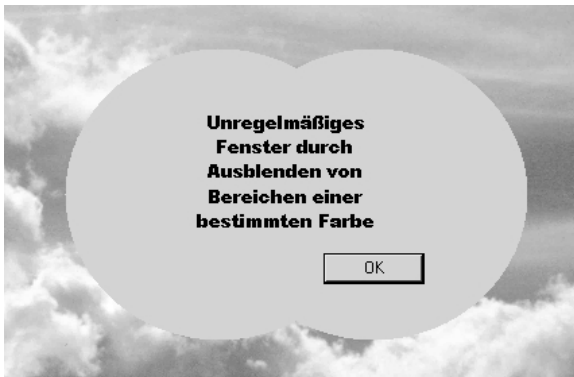


*Abbildung 67: Nicht rechteckiges Fenster durch Setzen eines Transparenzschlüssels*

Mit beiden Varianten, Setzen der Clipping-Region oder Festlegen des Transparenzschlüssels, lassen sich ähnliche Effekte erzielen. Das Setzen der Clipping-Region ist in der Regel vorzuziehen, da nicht ungewollt Bereiche transparent werden, die zufällig die als Transparenzschlüssel definierte Farbe aufweisen. Insbesondere, wenn Bilder auf dem Fenster angezeigt werden, ist die Verwendung des Transparenzschlüssels riskant.

Wenn, wie in Abbildung 67, das Fenster nicht mehr über eine Titelleiste verfügt, sollten Sie dem Anwender zusätzliche Möglichkeiten zum Verschieben des Fensters anbieten (siehe Rezept [6.2](#) ((Fenster ohne Titelleiste verschieben))).

## 98 Startbildschirm

Viele Wege führen nach Rom und ebenso viele Wege gibt es, die eigene Applikation mit der Anzeige eines Startbildschirms zu beginnen. Hier sollen zwei einfache Möglichkeiten vorgestellt werden. Die erste zeigt einen Startbildschirm an, der sich selbst nach einer vorgegebenen Zeit wie-

der schließt, die andere steuert den Startbildschirm vom Hauptprogramm aus, zeigt einen Fortschrittsbalken an und entfernt das Fenster nach Abschluss der Initialisierungsphase.

Beiden Varianten gemein ist, dass das Hauptfenster wie üblich mit `Application.Run` gestartet wird und die Anzeige des Startbildschirms mit `ShowDialog` in einem zweiten Thread erfolgt. So wird sichergestellt, dass die `MessageLoops` beider Fenster unabhängig voneinander ablaufen können und Ereignisse zu jeder Zeit in beiden Fenstern verarbeitet werden können. Dadurch haben Sie die Möglichkeit, innerhalb des Startfensters weitere Animationen zu programmieren, die neben der Initialisierung des Hauptprogramms ablaufen können, ohne dass sich beide gegenseitig beeinflussen.

Für die erste Variante benötigt man ein beliebiges Fenster als Startfenster, das beliebig parametrisiert und aufgebaut sein kann. Lediglich ein Timer, eingestellt auf die gewünschte Standzeit des Fensters, muss vorhanden sein, die `Enabled`-Eigenschaft auf `True` gesetzt. Im `Timer_Tick`-Ereignis schließt man mit `Me.Close` das Fenster. Alles andere geschieht in der Klasse des Hauptfensters.

Zwei statische Variablen in der Klasse des Hauptfensters speichern die Referenzen der beiden Fenster:

---

```
Private Shared StartupWindow As SplashScreen
Public Shared ApplicationMainWindow As MainWindow
```

---

`SplashScreen` ist die Klasse des Startfensters, `MainWindow` die des Hauptfensters. Gestartet wird die Anwendung über `Sub Main`. Listing 122 zeigt die Implementierung. Die beiden Fenster werden instanziiert, ein neuer Thread angelegt und gestartet, die Initialisierungsphase (`DoProgramInitialization`), die beliebig gestaltet werden kann, begonnen und abschließend das Hauptfenster geöffnet.

---

```
Public Shared Sub Main()
    ' Startfenster instanzieren
    StartupWindow = New SplashScreen()

    ' Hauptfenster instanzieren
    ApplicationMainWindow = New MainWindow()

    ' Thread für Startfenster anlegen und starten
    Dim T As New Thread(AddressOf ShowSplashScreen)
    T.Start()

    ' Beliebige Initialisierungsphase
    DoProgramInitialization()

    ' Anzeigen des Hauptfensters
    Application.Run(ApplicationMainWindow)

End Sub
```

---

Listing 122: Sub Main öffnet ein Startfenster und startet das Programm

Die Thread-Prozedur `ShowSplashScreen` (siehe Listing 123) zeigt mittels `ShowDialog` das Startfenster modal an und ruft nach dessen Schließung die Methode `SplashScreenTerminated` auf, um das Hauptfenster hierüber zu informieren. Geschlossen wird das Startfenster wie oben beschrieben über den eigenen Timer.

---

Basics

---

Datum/  
Zeit

---

Anwen-  
dungen

---

Zeich-  
nen

---

Bild-  
bearbei-  
tung

---

Win-  
dows  
Forms

---

Controls

---

Proper-  
tyGrid

---

Datei-  
system

---

Netz-  
werk

---

System

---

Daten-  
banken

---

XML

---

Wissen-  
schaft

---

Ver-  
schiede-  
nes

---

---

```

Private Shared Sub ShowSplashScreen()

    ' Startfenster modal anzeigen
    StartupWindow.ShowDialog()

    ' Diese Zeile wird erst erreicht, wenn das Startfenster
    ' geschlossen wurde
    StartupWindow.Dispose()

    ' Hauptfenster über vollständige Schließung informieren
    ApplicationMainWindow.BeginInvoke( _
        New ThreadStart( _
            AddressOf ApplicationMainWindow.SplashScreenTerminated))

End Sub

```

---

*Listing 123: Thread-Methode ShowSplashScreen zeigt das Startfenster an*

SplashScreenTerminated wird wegen des asynchronen Aufrufs mit BeginInvoke wieder vom Haupt-Thread, der auch das Hauptfenster erzeugt hat, ausgeführt. Hier wird dem Hauptfenster der Fokus zugewiesen. Sie können an dieser Stelle aber auch weiteren Code einfügen, der auf Steuerelemente des Hauptfensters zugreift und ausgeführt werden soll, nachdem das Startfenster geschlossen worden ist. Listing 124 zeigt die Minimal-Implementierung der Methode.

---

```

Public Sub SplashScreenTerminated()
    ' Hauptfenster aktivieren
    Me.Activate()
    Debug.WriteLine("Hauptfenster selbständig")
End Sub

```

---

*Listing 124: SplashScreenTerminated wird vom Haupt-Thread abgearbeitet und kann auf das Hauptfenster zugreifen*

Bei längeren Initialisierungsphasen sollten Sie den Anwender über den aktuellen Stand informieren, damit er nicht aus Ungeduld zum Taskmanager greift. In der zweiten Variante wird daher das Startfenster um eine Fortschrittsanzeige (ProgressBar) erweitert. Der Timer wird entfernt oder stillgelegt, das Schließen des Fensters erfolgt über eine Methode (CloseWindow).

---

```

' Fenster schließen
Protected Sub CloseWindowCallBack()
    Me.Close()
End Sub

Public Sub CloseWindow()

    ' Asynchroner Aufruf der CallBack-Methode
    BeginInvoke(New ThreadStart(AddressOf CloseWindowCallBack))

```

---

*Listing 125: Erweiterung der Klasse des Startfensters um Methoden zur Aktualisierung einer Fortschrittsanzeige und zum Schließen des Fensters*



```

End Sub

' Delegate für den Rückruf
Protected Delegate Sub SetProgressDelegate(ByVal progress As Integer)

' Die Rückrufmethode setzt die ProgressBar
Protected Sub SetProgressCallback(ByVal progress As Integer)
    ProgressBar1.Value = progress
End Sub

Public Sub SetProgress(ByVal progress As Integer)

    ' Asynchroner Aufruf der Callback-Methode
    BeginInvoke(New SplashScreen.SetProgressDelegate( _
        AddressOf SetProgressCallback), _
        New Object() {progress})

End Sub

```

*Listing 125: Erweiterung der Klasse des Startfensters um Methoden zur Aktualisierung einer Fortschrittsanzeige und zum Schließen des Fensters (Forts.)*

Wiederum ist es erforderlich, die Methoden über `BeginInvoke` asynchron aufzurufen, da nur der zweite Thread auf die Oberflächenelemente des Startfensters zugreifen darf. Für `SetProgress` wurde eine Delegate-Klasse definiert (`SetProgressDelegate`, siehe Listing 125), die die Signatur der Methode festlegt und die Übergabe des Parameters ermöglicht. Beide Methoden werden aufgeteilt in eine öffentliche threadsichere Methode, die vom Hauptfenster aus aufgerufen werden kann, und eine geschützte Callback-Methode, die über `BeginInvoke` aufgerufen wird.

Der Aufruf der beiden Methoden kann an beliebiger Stelle im Hauptprogramm vorgenommen werden, also z.B. innerhalb von `Sub Main` oder im Konstruktor oder Ereignis-Handlern des Hauptfensters. Listing 126 zeigt beispielhaft die Aufrufe innerhalb der Methode `DoProgramInitialization`.

```

Private Shared Sub DoProgramInitialization()

    ' Hier kann beliebiger Code für die Initialisierung stehen
    Debug.WriteLine("Initializing...")
    Dim i As Integer
    For i = 0 To 100
        ' Startfenster über aktuellen Fortschritt informieren
        StartupWindow.SetProgress(i)

        ' Initialisierungsphase
        ' Thread.Sleep ist nur ein Dummy!
        System.Threading.Thread.Sleep(50)
    Next

    Debug.WriteLine("... Ready")

```

*Listing 126: Steuerung des Startfensters über `SetProgress` und `CloseWindow`*

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Proper-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

---

```
' Startfenster schließen
StartupWindow.CloseWindow()
```

---

End Sub

---

*Listing 126: Steuerung des Startfensters über SetProgress und CloseWindow (Forts.)*



*Abbildung 68: Beispiel für ein Startfenster*

Die Implementierung der Callback-Aufrufe innerhalb der Startfenster-Klasse ermöglicht Ihnen einen flexiblen und threadsicheren Einsatz des Startfensters für verschiedene Anwendungen. Abbildung 68 zeigt ein Beispiel für den beschriebenen Startbildschirm, Ihrer Kreativität sind hier aber keine Grenzen gesetzt. Dank des eigenen Threads können Sie im Startfenster beliebige Animationen ablaufen lassen, ohne auf die Vorgänge im Hauptfenster Rücksicht nehmen zu müssen.

Möchten Sie ein besonders originelles Startfenster für Ihre Anwendung anzeigen, dann erzeugen Sie doch ein Fenster ohne Titelleiste mit einer ungewöhnlichen Form, wie es in den vorangegangenen Rezepten beschrieben wurde.

## 99 Dialoge kapseln

War in der MFC-Programmierung von jeher bedingt durch die Document-View-Struktur die Kapselung von Dialogen gang und gäbe, so gab es unter VB keine einheitlichen Vorgehensweisen. Auch in VB4 war es bereits möglich, die Funktionalität eines Dialoges in dessen Fensterklasse unterzubringen und den Dialog über eine einzige Funktion von außen zu steuern. Allerdings wurde von dieser Möglichkeit kaum Gebrauch gemacht. Vielmehr findet man in vielen alten VB-Programmen unstrukturierte Kreuz- und Querverweise, etwa von der rufenden Methode auf Steuerelemente des Dialogs und umgekehrt von Methoden innerhalb der Dialogklasse auf Steuerelemente oder Methoden des Hauptprogramms.

Da es auch in den .NET-Newsgruppen oft Fragen gibt, wie man von Methoden des einen Fensters auf Steuerelemente eines anderen zugreifen kann, soll hier die prinzipielle Vorgehensweise anhand von zwei Beispielen erläutert werden. In diesem Rezept wird erklärt, wie man einen einfa-

chen Dialog aufbaut, der die Änderung der vorgegebenen Daten ermöglicht, im nächsten Rezept wird dieser Dialog um eine Rückrufmethode, einer Übernehmen-Schaltfläche, erweitert.

Will man ein universell einsetzbares Dialogfenster programmieren, so ist es wichtig, alle Zuständigkeiten exakt festzulegen. Der Dialog soll über eine Methode angezeigt werden, die übergebenen Daten anzeigen und durch den Anwender ändern lassen und nach dem Schließen eine Information zurückgeben, ob der Anwender seine Zustimmung (OK-Taste) gegeben hat oder seine Änderungen verworfen hat (Abbrechen-Taste). Im Dialog selbst darf nicht auf Elemente fremder Objekte, z.B. Steuerelemente anderer Fenster, zugegriffen werden. Sonst könnte man zum einen den Dialog nicht in einem anderen Kontext verwenden und zum anderen ergeben sich oft unerwünschte Seiteneffekte. In der anderen Richtung, also beim Aufruf des Dialogs, darf auch nicht von außen auf Interna des Dialogfensters zugegriffen werden. Stattdessen kann man den Aufruf des Dialogs ganz gezielt über eine einzige statische Methode vornehmen.

Zum Verständnis sollen hier zunächst die Randbedingungen des Beispielsprogramms erläutert werden. Eine Datenklasse (*Vehicle*, siehe Listing 127) definiert einige Eigenschaften eines Fahrzeugs. Mehrere Instanzen dieser Klasse werden angelegt und die Informationen in einer *ListView* im Hauptfenster angezeigt (siehe Listing 128). Die letzte Spalte der *ListView* zeigt die Farbe zum einen als Text und zum anderen als Hintergrundfarbe. Jeder Zeile der *ListView* ist somit genau eine Instanz von *Vehicle* zugeordnet, deren Referenz in der *Tag*-Eigenschaft des *ListViewItems* gespeichert wird. Abbildung 69 zeigt das Hauptfenster des Beispiels.

```
Public Class Vehicle

    ' Eigenschaften des Fahrzeugs
    Public Manufacturer As String
    Public VehicleType As String
    Public VehicleColor As Color

    ' Konstruktor
    Public Sub New(ByVal manufacturer As String, _
        ByVal vehicleType As String, _
        ByVal vehicleColor As Color)

        Me.Manufacturer = manufacturer
        Me.VehicleType = vehicleType
        Me.VehicleColor = vehicleColor
    End Sub

End Class
```

Listing 127: Beispielklasse *Vehicle*

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

    ' Ein paar Fahrzeuge hinzufügen
    AddVehicle(New Vehicle("Mercedes", "200D", Color.White))
    AddVehicle(New Vehicle("VW", "Golf", Color.Silver))
    AddVehicle(New Vehicle("Opel", "Astra", Color.LightGreen))
```

Listing 128: *ListView* mit Beispieldaten füllen

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

```

AddVehicle(New Vehicle("Ford", "Mondeo", Color.LightCyan))

End Sub

' Zeile in der ListView für Vehicle-Objekt anlegen
Public Sub AddVehicle(ByVal v As Vehicle)

    ' Neuer ListView-Eintrag
    Dim lvi As ListViewItem = LVVehicles.Items.Add(v.Manufacturer)

    ' Farbdarstellung für letzte Spalte ermöglichen
    lvi.UseItemStyleForSubItems = False

    ' Referenz des Fahrzeugobjektes speichern
    lvi.Tag = v

    ' Typ eintragen
    lvi.SubItems.Add(v.VehicleType)

    ' Farbe als Text und Hintergrundfarbe eintragen
    Dim lvs1 As ListViewItem.ListViewSubItem = _
        lvi.SubItems.Add(v.VehicleColor.ToString())
    lvs1.BackColor = v.VehicleColor

End Sub

```

---

Listing 128: ListView mit Beispieldaten füllen (Forts.)

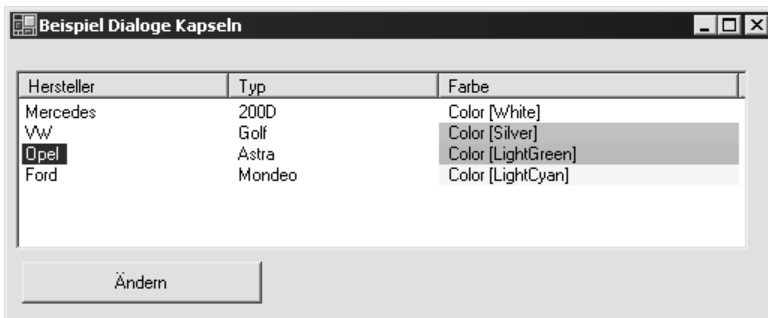


Abbildung 69: Beispielprogramm zur Demonstration gekapselter Dialoge

In der Click-Methode der Schaltfläche `BTNChange` wird ein Dialog aufgerufen, der die Daten der ausgewählten Zeile anzeigt. Listing 129 zeigt den Aufruf des Dialogs, Abbildung 70 das Dialogfenster.

---

```

Private Sub BTNChange_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles BTNChange.Click

    ' Referenz des Vehicle-Objektes für die aktuelle Auswahl holen

```

---

Listing 129: Einzige Schnittstelle zum Dialog: `CreateAndShow`

```

Dim v As Vehicle = DirectCast(LVehicles.FocusedItem.Tag, _
    Vehicle)

' Dialog anzeigen
DialogA.CreateAndShow(v)

' Änderungen übernehmen
' Hier evtl. Abfrage auf DialogResult.OK
SetData(v)

End Sub

```

Listing 129: Einzige Schnittstelle zum Dialog: CreateAndShow (Forts.)



Abbildung 70: Beispieldialog mit OK- und Abbrechen-Taste

Dem Dialog wird die Referenz des Datenobjektes, das zu der ausgewählten Zeile gehört, übergeben. Er stellt die Daten dar, lässt Änderungen zu und gibt nach Betätigung der OK-Schaltfläche die geänderten Daten zurück. Diese müssen dann im Hauptfenster in der ListView-Darstellung aktualisiert werden, was durch die Methode `SetData` des Hauptfensters (Listing 130) realisiert wird.

```

Private Sub SetData(ByVal v As Vehicle)
' Aktuelle Auswahl ermitteln
Dim lvi As ListViewItem = LVehicles.FocusedItem

' Texte eintragen
lvi.SubItems(0).Text = v.Manufacturer
lvi.SubItems(1).Text = v.VehicleType
lvi.SubItems(2).Text = v.VehicleColor.ToString()

' Farbe aktualisieren
lvi.SubItems(2).BackColor = v.VehicleColor

End Sub

```

Listing 130: Aktualisierung des ListViews mit den geänderten Daten

Auf der Seite des Hauptfensters erfolgt somit kein einziger Zugriff auf die Interna des Dialogs. Die einzige Schnittstelle ist die Methode `CreateAndShow`. Diese ist eine statische Methode der Dialogklasse `DialogA` (Listing 131). Sie übernimmt die Instanziierung des Dialogfensters, so dass auf der rufenden Seite kein Aufruf von `New` notwendig ist. Nach Initialisierung der Steuerelemente mit

den als Parameter übergebenen Daten zeigt sie mit `ShowDialog` das Fenster modal an. Die Methode wird erst nach Schließen des Fensters fortgesetzt. Hatte der Anwender durch Betätigung der OK-Schaltfläche seine Zustimmung zu seinen Änderungen bekundet, werden die Werte aus den Steuerelementen zurückgelesen und in die Datenstruktur eingetragen. Die Methode gibt `DialogResult.OK` bzw. `DialogResult.Cancel` zurück, je nach Entscheidung des Anwenders.

Für die TextBoxen ist hier kein zusätzlicher Programmieraufwand notwendig, lediglich für die Farbschaltfläche wird ein bisschen Code benötigt, um den Farbauswahl-Dialog anzuzeigen. Dieser Code ist hier aber irrelevant, Sie finden ihn auf der CD.

Wichtig für die Steuerung des Dialogs ist die Festlegung einiger Eigenschaften, so dass auch für die Reaktion auf Betätigen der OK- bzw. ABBRECHEN-Schaltfläche kein zusätzlicher Code benötigt wird. Tabelle 13 zeigt diese Einstellungen.

---

```
Public Shared Function CreateAndShow(ByVal data As Vehicle) _
    As DialogResult

    ' Instanz der Dialogklasse anlegen
    Dim dlg As New DialogA()

    ' Steuerelemente mit Daten initialisieren
    dlg.TBManufacturer.Text = data.Manufacturer
    dlg.TBType.Text = data.VehicleType
    dlg.BTNColor.BackColor = data.VehicleColor

    ' Dialog modal anzeigen
    Dim dr As DialogResult = dlg.ShowDialog()

    ' Prüfen, ob OK oder Abbrechen gedrückt wurde
    If dr = System.Windows.Forms.DialogResult.OK Then
        ' Daten zurückübertragen
        data.Manufacturer = dlg.TBManufacturer.Text
        data.VehicleType = dlg.TBType.Text
        data.VehicleColor = dlg.BTNColor.BackColor
    End If

    ' OK oder Cancel zurückgeben
    Return dr

End Function
```

---

Listing 131: Instanziierung und Steuerung des Dialogs in der statischen Methode `CreateAndShow`

| Typ    | Name                              | Eigenschaft  | Wert      |
|--------|-----------------------------------|--------------|-----------|
| Button | BTNOK                             | DialogResult | OK        |
| Button | BTNCancel                         | DialogResult | Cancel    |
| Button | <b>alle anderen Schaltflächen</b> | DialogResult | None      |
| Form   | DialogA                           | AcceptButton | BTNOK     |
|        |                                   | CancelButton | BTNCancel |

Tabelle 13: Diese Eigenschaften steuern das Verhalten des Dialogs

Mit minimalem Aufwand wird so eine strikte Trennung von Hauptfenster und Dialog erreicht. Der Dialog erhält über die Parameterliste der Methode `CreateAndShow` die Daten und ist selbst für deren Darstellung verantwortlich. Hier wurden die Daten als Referenz eines `Vehicle`-Objektes übergeben. Selbstverständlich können Sie die Übergabe auch z.B. durch `ByRef`-Parameter gestalten, ganz so, wie es Ihre Anwendung erfordert. Durch die Belegung der entsprechenden Eigenschaften der Schaltflächen bzw. des Dialogfensters erfolgt das Schließen des Dialogs vollautomatisch durch das Framework. Auch über die Tastatur lässt sich der Dialog steuern (`[Enter]`-Taste, `[Esc]`-Taste), ohne dass dafür auch nur eine Zeile programmiert werden müsste.

## 100 Gekapselter Dialog mit Übernehmen-Schaltfläche

Oft möchte man in einem Dialog mit Hilfe einer Übernehmen-Schaltfläche dem Anwender ermöglichen, die geänderten Daten sofort zu akzeptieren und die Änderungen in den geöffneten Ansichten zu betrachten, ohne den Dialog zu schließen. In der Konstruktion aus dem vorigen Beispiel ist das nicht direkt möglich, da die Daten erst nach Ende von `ShowDialog`, also erst nach Schließen des Dialogfensters übertragen werden. Über ein `Delegate` lässt sich die gewünschte Funktionalität jedoch erreichen. Betrachten wir zunächst die Ergänzungen, die im Dialog (Klasse `DialogB`, Abbildung 71) notwendig sind.



Abbildung 71: Dialog mit Übernehmen-Schaltfläche

```
' Definition der Delegate-Klasse für Rückrufe
Public Delegate Sub ApplyChangesDelegate(ByVal data As Vehicle)

' Referenz des Rückruf-Delegates
Protected ApplyChangesCallback As ApplyChangesDelegate

' Referenz der Daten
Protected Data As Vehicle
```

Die `Delegate`-Definition legt den Aufbau einer Rückrufmethode fest. Sie soll als Parameter die Referenz der Daten annehmen. `ApplyChangesCallback` speichert die Referenz der Rückrufmethode, `Data` die der Daten.

Der Aufbau der öffentlichen Methode `CreateAndShow` (Listing 132) wird etwas umfangreicher. Als zusätzlicher Parameter wird die Rückrufmethode übergeben. Damit später als Reaktion auf die Betätigung der ÜBERNEHMEN-Schaltfläche auf diese Methode und die Daten zurückgegriffen werden kann, werden die Informationen in `Data` bzw. `ApplyChangesCallback` gespeichert. Alles Weitere ist identisch zur Methode in `DialogA`.

---

```

Public Shared Function CreateAndShow(ByVal data As Vehicle, _
    ByVal callBack As ApplyChangesDelegate) As DialogResult

    ' Instanz der Dialogklasse anlegen
    Dim dlg As New DialogB()

    ' Daten merken
    dlg.Data = data

    ' Rückrufmethode merken
    dlg.ApplyChangesCallBack = callBack

    ' Steuerelemente mit Daten initialisieren
    dlg.TBManufacturer.Text = data.Manufacturer
    dlg.TBType.Text = data.VehicleType
    dlg.BTNColor.BackColor = data.VehicleColor

    ' Dialog modal anzeigen
    Dim dr As DialogResult = dlg.ShowDialog()

    ' Prüfen, ob OK oder Abbrechen gedrückt wurde
    If dr = System.Windows.Forms.DialogResult.OK Then
        ' Daten zurückübertragen
        data.Manufacturer = dlg.TBManufacturer.Text
        data.VehicleType = dlg.TBType.Text
        data.VehicleColor = dlg.BTNColor.BackColor
    End If

    ' OK oder Cancel zurückgeben
    Return dr

End Function

```

---

Listing 132: CreateAndShow mit Übergabe einer Rückrufmethode für die Übernehmen-Schaltfläche

Listing 133 zeigt die Realisierung der Ereignis-Methode der ÜBERNEHMEN-Schaltfläche. Die Daten werden aus den Steuerelementen zurückgelesen und in die Datenstruktur eingetragen. Dann erfolgt der Aufruf der Rückrufmethode. Wie und wo die Methode implementiert ist, ist an dieser Stelle ohne Bedeutung. Der Dialog braucht nichts über die Interna des rufenden Programms zu wissen.

---

```

Private Sub BTNApply_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles BTNApply.Click

    ' Daten aus Steuerelementen übertragen
    Data.Manufacturer = Me.TBManufacturer.Text
    Data.VehicleType = Me.TBType.Text
    Data.VehicleColor = Me.BTNColor.BackColor

    ' Rückrufmethode aufrufen

```

---

Listing 133: Übernehmen-Schaltfläche gedrückt – Rückrufmethode aufrufen



---

```
ApplyChangesCallBack(Data)
```

```
End Sub
```

---

*Listing 133: Übernehmen-Schaltfläche gedrückt – Rückrufmethode aufrufen (Forts.)*

Im Hauptfenster muss der Aufruf des Dialogs entsprechend geändert werden. Der einzige Unterschied zum vorangegangenen Beispiel ohne ÜBERNEHMEN-Schaltfläche ist, dass zusätzlich die Rückrufmethode angegeben werden muss. Da bereits mit `SetData` eine zur `Delegate`-Definition passende Methode existiert, ist kein weiterer Programmieraufwand erforderlich. Listing 134 zeigt den Aufruf des Dialogs.

---

```
Private Sub BTNChangeB_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles BTNChangeB.Click

    ' Referenz des Vehicle-Objektes für die aktuelle Auswahl holen
    Dim v As Vehicle = DirectCast(LVehicles.FocusedItem.Tag, _
        Vehicle)

    ' Dialog anzeigen
    DialogB.CreateAndShow(v, _
        New DialogB.ApplyChangesDelegate(AddressOf SetData))

    ' Änderungen übernehmen
    SetData(v)

End Sub
```

---

*Listing 134: Aufruf des Dialogs und Übergabe der Rückrufmethode*

Nach wie vor ist der Dialog vom Hauptfenster unabhängig und kann auch in einem anderen Kontext verwendet werden. Für den Rückruf kann eine beliebige Methode bereitgestellt werden. Sie muss lediglich der durch `DialogB.ApplyChangesDelegate` festgelegten Signatur entsprechen. Der Dialog selbst muss nicht wissen, wer die Rückrufmethode wie ausführt. Andererseits muss die rufende Methode keine Kenntnis von z.B. einer ÜBERNEHMEN-Schaltfläche haben. Die Aufgaben sind klar getrennt und die gesamte Funktionalität des Dialogs in der Dialogklasse gekapselt. So erhalten Sie ein klares und robustes Design im Umgang mit Dialogfenstern.

## 101 Dialog-Basisklasse

In einem Projekt, das viele Dialoge beinhaltet, ist es sinnvoll, für alle Dialoge eine gemeinsame Basisklasse zu definieren. So stellen Sie sicher, dass das Aussehen der Dialogfenster einheitlich ist. Zusätzlich können die gemeinsamen Funktionen zentral in der Basisklasse definiert werden. Abbildung 72 zeigt beispielhaft einen möglichen Aufbau des Basisdialogs.

Positionieren Sie die OK-, ABBRECHEN- und ÜBERNEHMEN-Schaltflächen nach Ihren Wünschen, verwenden Sie dabei möglichst die `Anchor`-Eigenschaften, um die Positionen der Schaltflächen bei abgeleiteten Dialogfenstern anderer Größe an einer einheitlichen Stelle zu fixieren. Ergänzen Sie weitere Elemente wie Hilfe-Tasten etc., die all Ihren Dialogen gemein sein sollen.

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Propert-  
yGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

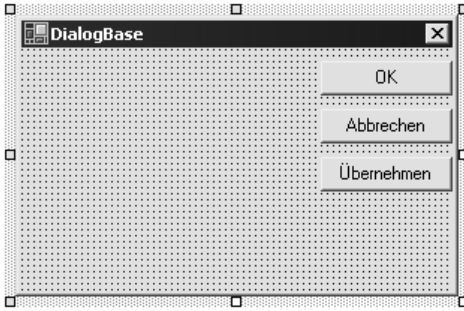


Abbildung 72: Dialog als Basis für alle anderen Dialoge eines Projektes

Die Methode `CreateAndShow` (Listing 135) hat einen ähnlichen Aufbau wie in den vorangegangenen Beispielen. Damit sie von außen nicht direkt aufgerufen werden kann, wird sie als geschützte Funktion deklariert. Aufgerufen wird später nur die entsprechende Überladung der abgeleiteten Klasse. Als zusätzlicher Parameter wird die Referenz der Dialog-Instanz übergeben. Die Daten können hier nur allgemein als `Object`-Referenz übernommen werden.

---

```
Protected Shared Function CreateAndShow( _
    ByVal dialog As DialogResult, ByVal data As Object, _
    ByVal callBack As ApplyChangesCallBackDelegate) As DialogResult

    ' Parameterinformationen speichern
    dialog.Data = data
    dialog.ApplyChangesCallBack = callBack

    ' Übernehmen-Schaltfläche freischalten oder sperren
    dialog.BTNApply.Enabled = Not callBack Is Nothing

    ' Daten in Steuerelemente kopieren
    dialog.TransferDataToControls()

    ' Dialog modal anzeigen
    Dim dr As DialogResult = dialog.ShowDialog()

    ' Datentransfer, wenn OK gedrückt wurde
    If dr = System.Windows.Forms.DialogResult.OK Then
        dialog.TransferControlsToData()
    End If

    ' Rückgabe OK oder Cancel
    Return dr

End Function
```

---

Listing 135: `CreateAndShow` der Basisklasse

Abhängig davon, ob eine Rückrufmethode verwendet werden soll oder nicht, wird die ÜBERNEHMEN-Schaltfläche freigeschaltet oder gesperrt. Über die Methode `TransferDataToControls`, die von der abgeleiteten Klasse überschrieben werden muss, werden die Informationen aus dem

Datenobjekt in die Steuerelemente kopiert. Anschließend erfolgt die modale Anzeige des Dialogs und, falls der Anwender die OK-Taste gedrückt hat, die Rückübertragung der Daten aus den Steuerelementen in die Datenstruktur.

Aus Gesichtspunkten der Objektorientierten Programmierung müssten die beiden Methoden `TransferDataToControls` und `TransferControlsToData` als abstrakte Funktionen (Mustoverride) deklariert werden. Dann wäre auch die Klasse `DialogBase` abstrakt und niemand könnte versehentlich eine Instanz der Basisklasse erstellen. Doch leider kommt der Designer des Visual Studios nicht mit abstrakten Basisklassen zurecht. Zum einen werden im Dialog der Vererbungsauswahl zum Anlegen abgeleiteter Formulare nur nicht abstrakte Klassen als mögliche Basisklassen angezeigt, zum anderen kann der Formular-Designer nicht mit abstrakten Basisklassen umgehen, da er eine Instanz der Basisklasse anlegen muss.

Es muss daher ein Kompromiss getroffen werden, der auch den Umgang mit dem Designer ermöglicht. Die beiden Methoden werden daher als virtuelle Methoden (Overridable) implementiert, die Implementierung besteht aber lediglich aus dem Auslösen einer Exception (Listing 136). So wird zwar erst zur Laufzeit ein Fehler gemeldet, wenn vergessen wurde, die Methoden in der abgeleiteten Klasse zu überschreiben, dafür kann der Designer im vollen Umfang genutzt werden.

```
Protected Overridable Sub TransferDataToControls()
    Throw New Exception( _
        "TransferDataToControls wurde nicht überschrieben")
End Sub
Protected Overridable Sub TransferControlsToData()
    Throw New Exception( _
        "TransferControlsToData wurde nicht überschrieben")
End Sub
```

*Listing 136: Kompromisslösung zu Gunsten des Designers: Implementierung als nicht abstrakte Methoden*

Da die Daten in der Basisklasse allgemein gehalten werden müssen, tritt an die Stelle des Typs `Vehicle` aus den vorangegangenen Beispielen der Typ `Object`. Auch die Ereignis-Methode der ÜBERNEHMEN-Schaltfläche kann hier schon implementiert werden (siehe Listing 137). Die virtuelle Methode `TransferControlsToData` übernimmt das Kopieren der Daten.

```
' Delegate-Klasse für die Rückrufmethode
Public Delegate Sub ApplyChangesCallBackDelegate(ByVal data _
    As Object)

' Rückrufmethode und Daten
Protected ApplyChangesCallBack As ApplyChangesCallBackDelegate
Protected Data As Object

' Unternehmen-Schaltfläche gedrückt
Private Sub BTNApply_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles BTNApply.Click

    ' Datenobjekt mit Informationen aus den Steuerelementen füllen
    TransferControlsToData()
```

*Listing 137: Event-Routine für Unternehmen-Schaltfläche und Member-Variablen der Basisklasse*

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Propor-  
talityGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

```

' Rückrufmethode aufrufen
If Not ApplyChangesCallBack Is Nothing Then _
    ApplyChangesCallBack(Data)
End Sub

```

Listing 137: Event-Routine für Übernehmen-Schaltfläche und Member-Variablen der Basisklasse (Forts.)

Mit dieser Definition der Basisklasse ist der Dialog in seinen Grundfunktionen weitestgehend fertig gestellt. Die speziellen Implementierungen für den Datenfluss zwischen dem `Data`-Objekt und den Steuerelementen werden in den o.g. Transfer-Methoden durch Überschreibung in der abgeleiteten Klasse vorgenommen. Abbildung 73 zeigt ein Beispiel für eine Anwendung von `DialogBase`.

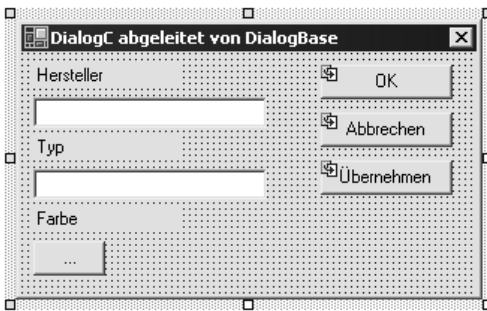


Abbildung 73: `DialogC` als Ableitung von `DialogBase`

Zu implementieren ist hier zunächst die statische Methode `CreateAndShow` (Listing 138), die hier einen speziellen Datentyp annimmt, damit der Compiler die Überprüfung vornehmen kann. Die Methode delegiert die Aufgabe direkt an die entsprechende Methode der Basisklasse. Sie legt hierfür eine Instanz der abgeleiteten Klasse (`DialogC`) an und übergibt sie und die anderen Parameter an `DialogBase.CreateAndShow`.

```

' Überladung der statischen Methode zum Anzeigen des Dialogs
Public Overloads Shared Function CreateAndShow( _
    ByVal data As Vehicle, _
    ByVal applyChangesCallBack As ApplyChangesCallBackDelegate) _
    As DialogResult

    ' Delegation an die statische Methode der Basisklasse
    Return CreateAndShow(New DialogC(), data, applyChangesCallBack)

End Function

```

Listing 138: Die Implementierung der Methode `CreateAndShow` in der abgeleiteten Klasse delegiert die Aufgaben an die Basisklasse

In den Überschreibungen der Transfer-Methoden werden die Daten zwischen den Steuerelementen und dem `Data`-Objekt ausgetauscht (Listing 139). Die Methoden werden in der Basisklasse aufgerufen.

```

' Datentransfer Daten -> Steuerelemente
Protected Overrides Sub TransferDataToControls()
    Dim data As Vehicle = DirectCast(Me.Data, Vehicle)
    TBManufacturer.Text = data.Manufacturer
    TBType.Text = data.VehicleType
    BTNColor.BackColor = data.VehicleColor
End Sub

' Datentransfer Steuerelemente -> Daten
Protected Overrides Sub TransferControlsToData()
    Dim data As Vehicle = DirectCast(Me.Data, Vehicle)
    data.Manufacturer = TBManufacturer.Text
    data.VehicleType = TBType.Text
    data.VehicleColor = BTNColor.BackColor
End Sub

```

Listing 139: Datentransfer Oberfläche zu Data-Objekt

Das sind alle Ergänzungen, die zur Bereitstellung der Grundfunktionalität notwendig sind. Zusätzliche Funktionen, wie z.B. das Anzeigen des Farbdialogs, können entsprechend hinzugefügt werden.

## 102 Validierung der Benutzereingaben

Immer dann, wenn Anwender Daten in die Dialoge ihrer Programme eingeben können, müssen sie mit Fehlern rechnen. Die Ursachen hierfür können durchaus vielfältig sein. Sei es, dass die geforderten Eingaben unklar oder mehrdeutig sind, dass der Anwender die Daten unvollständig eingibt oder sich schlichtweg verschreibt. Je nach Benutzerkreis müssen Sie auch mit der vorsätzlichen Eingabe falscher Informationen rechnen. Die Liste der Fehlerquellen lässt sich beliebig fortsetzen.

Umso wichtiger ist es daher, den Anwender bei der Eingabe von Daten an einer sehr kurzen Leine zu führen, ihn mit sinnvollen Informationen zu unterstützen und die Daten auf Plausibilität zu prüfen. Erst, wenn alle Datenfelder gültige Werte enthalten, darf ein Dialog mit OK geschlossen werden können. Natürlich muss dem Anwender auch angezeigt werden, was er falsch gemacht hat bzw. welche Daten noch fehlen oder inkorrekt sind.

Oft ist es sinnvoll vorzusehen, dass ein Eingabefeld erst dann verlassen werden kann, wenn es gültige Werte enthält. Vorsicht ist jedoch geboten, wenn sich die Gültigkeitsprüfung über mehrere Steuerelemente gleichzeitig erstreckt. Schnell stellen sich dabei Deadlock-Situationen ein, wenn z.B. ein Feld nicht verlassen werden kann, weil ein anderes Feld ungültige Werte enthält, aber keine Möglichkeit besteht, diese Werte zu ändern.

Bevor Sie damit beginnen, die Validierung der Benutzereingaben bei komplexeren Dialogen umzusetzen, sollten Sie sich daher unbedingt ein Konzept niederschreiben, in dem festgelegt wird, wie die Daten zu prüfen sind, in welcher Reihenfolge der Anwender die Daten eingeben kann oder muss und wie bei fehlerhaften Werte verfahren werden soll.

An einem einfachen Beispiel wird erläutert, welche Möglichkeiten Ihnen unter .NET zur Verfügung stehen. Der Beispieldialog enthält drei Steuerelemente, die mit Werten zu füllen sind. In das erste Feld soll ein Name eingegeben werden. Dieser Name darf nicht aus einer leeren Zeichenkette bestehen. Im zweiten Feld soll das Geburtsdatum eingetragen werden. Die Person muss mindes-

tens 18 Jahre alt sein, Eingaben, die ein Alter über 100 Jahre ergeben, sollen abgewiesen werden. Das dritte Eingabeelement ist eine CheckBox, mit der der Benutzer seine Zustimmung zu etwaigen Bedingungen geben muss. Nur, wenn alle Kriterien erfüllt sind, dürfen die Daten angenommen werden. Abbildung 74 zeigt den Aufbau des Dialogfensters im Ausgangszustand. Namensfeld und CheckBox sind leer, das Feld für die Eingabe des Geburtsdatums zeigt das aktuelle Datum und ist somit ebenfalls ungültig.

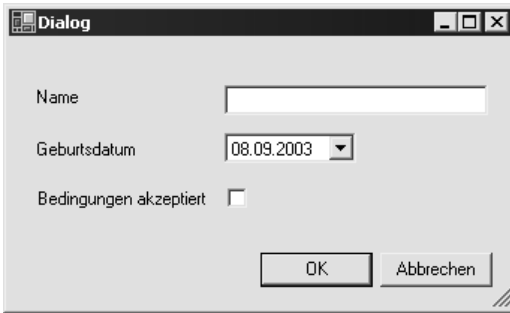


Abbildung 74: Nur wenn alle Daten korrekt eingegeben wurden, sollen die Werte übernommen werden

Nun muss entschieden werden, wann und wie die Eingaben zu prüfen sind. Folgende Vorgaben werden festgelegt:

- ▶ Grundsätzlich soll der Anwender die Eingaben in beliebiger Reihenfolge vornehmen können, sofern nicht andere Einschränkungen dagegen sprechen.
- ▶ Ein Steuerelement, das ausgewählt wurde (also den Fokus hat), darf nur verlassen werden, wenn es einen zulässigen Wert enthält.
- ▶ Steuerelemente mit fehlerhaften Eingaben sollen markiert werden, damit direkt ins Auge fällt, wo nachgebessert werden muss.
- ▶ Der Anwender muss bei unzulässigen Eingaben darüber informiert werden, was falsch ist und welche Werte gültig, richtig und erforderlich sind.
- ▶ Die Eingabe darf jederzeit, auch wenn Steuerelemente unzulässige Werte enthalten, abgebrochen werden. Mit Betätigen der Schaltfläche ABBRECHEN oder entsprechenden Mechanismen soll das Dialogfenster geschlossen werden und den Abbruch als `DialogResult` weitermelden.
- ▶ Die Schaltfläche OK darf nur dann zum Schließen des Fensters führen, wenn alle Eingaben für korrekt befunden worden sind.

Es gibt eine Reihe von Mechanismen, die Ihnen das Framework zur Verfügung stellt, um die Benutzerführung vorzunehmen. Insbesondere zwei Ereignisse dienen zur Überprüfung der Eingaben und können dazu benutzt werden, das Verlassen des Eingabefeldes zu verhindern:

- ▶ Validating-Event
- ▶ Validated-Event

Für Steuerelemente, die überprüft werden sollen, werden diese beiden Ereignisse ausgelöst, bevor ein anderes Steuerelement den Eingabefokus erhält. Im Validating-Ereignis wird ein Parameter vom Typ `CancelEventArgs` übergeben, mit dessen Hilfe Sie das Verlassen des Steuerelementes verhindern können. Nur, wenn die `Cancel`-Eigenschaft dieses Parameters den Ausgangswert `False` aufweist, wird das Validated-Ereignis ausgelöst und der Eingabefokus weitergegeben.

Voraussetzung für das Auslösen der beiden Ereignisse ist, dass die Eigenschaft `CausesValidation` sowohl des Steuerelementes, das den Fokus besitzt, als auch des Steuerelementes, das den Fokus erhalten soll, den Wert `True` hat. Um also zu erreichen, dass die Validierung beim Wechsel zwischen den Steuerelementen erfolgt, wird `CausesValidation` für die drei Eingabefelder auf `True` gesetzt. Damit der Dialog jederzeit geschlossen werden kann, erhält die `CausesValidation`-Eigenschaft der ABBRECHEN-Schaltfläche den Wert `False`.

Nun könnte man leicht in Versuchung geraten die `CausesValidation`-Eigenschaft der OK-Schaltfläche auf `True` zu setzen, in der Erwartung, dass das Fenster damit ja nur geschlossen werden kann, wenn die vorherige Validierung nicht verhindert, dass das aktuelle Steuerelement den Fokus verliert. Doch Vorsicht! Die Events werden ja nur beim Verlassen eines Steuerelementes ausgelöst. Es können aber andere Felder ungültig sein, die in diesem Falle nicht überprüft würden.

Besser ist es daher, auch für die OK-Schaltfläche `CausesValidation` auf `False` zu setzen und stattdessen im `Closing`-Ereignis des Fensters alle Steuerelemente auf Gültigkeit zu prüfen. Auch dieses Ereignis stellt einen Parameter bereit, mit dessen Hilfe Sie in diesem Fall das Schließen des Fensters verhindern können.

Bleibt noch zu klären, wie die fehlerhaften Eingaben visualisiert werden können. .NET stellt Ihnen zu diesem Zweck die Klasse `ErrorProvider` zur Verfügung, eine Komponente, die Sie ganz einfach aus der Toolbox auf Ihr Fenster ziehen können. Durch Aufruf der Methode `SetError` können Sie für ein bestimmtes Steuerelement einen Fehlertext setzen oder löschen. Handelt es sich bei dem Text nicht um einen leeren String, dann wird neben dem Steuerelement ein roter Kreis mit einem weißen Ausrufezeichen angezeigt. Sie können über die Methode `SetIconAlignment` vorgeben, an welcher Position die Markierung angezeigt werden soll. Der Standardwert ist rechts neben dem Steuerelement. Auch der Abstand lässt sich mit der Methode `SetIconPadding` einstellen.

Der zugewiesene Text wird in Form eines Tooltips angezeigt, wenn der Mauszeiger für einen Moment über der Markierung gehalten wird. In Abbildung 75 sehen Sie einen typischen Anwendungsfall.

Bei erstmaligem Auftreten eines Fehlers (identifiziert durch den angegebenen Text) blinkt die Markierung einige Male und wird dann statisch angezeigt. Dieses Verhalten lässt sich über die Eigenschaften `BlinkRate` und `BlinkStyle` steuern. Entfernt wird die Markierung, wenn `SetError` eine leere Zeichenkette übergeben wird.

Achtung

Zeigen Sie Eingabefehler in Dialogen nie mit `MessageBoxen` an, da diese den Eingabefluss unterbrechen und vom Benutzer gesondert quittiert werden müssen.

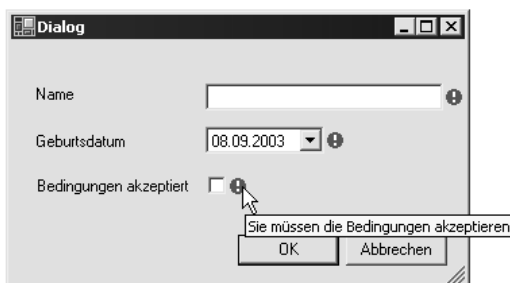


Abbildung 75: Markieren fehlerhafter Eingaben und Anzeigen von Hilfstexten

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Propor-  
tityGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

| Typ            | Name         | Eigenschaft      | Wert         |
|----------------|--------------|------------------|--------------|
| Fenster        | Dialog       | AcceptButton     | ButtonOK     |
|                |              | CancelButton     | ButtonCancel |
| Schaltfläche   | ButtonOK     | CausesValidation | False        |
|                |              | DialogResult     | OK           |
| Schaltfläche   | ButtonCancel | CausesValidation | False        |
|                |              | DialogResult     | Cancel       |
| TextBox        | TBName       | CausesValidation | True         |
| DateTimePicker | DTPBirthday  | CausesValidation | True         |
| CheckBox       | CHKAccept    | CausesValidation | True         |

Tabelle 14: Einstellung der wesentlichen Eigenschaften des Dialogfensters und der Steuerelemente

In Tabelle 14 finden Sie die für dieses Beispiel wichtigen Eigenschaften der Steuerelemente und des Fensters. Instanziert und angezeigt wird der Dialog wieder über eine statische Methode `CreateAndShow`.

Für alle drei Eingabefelder wird in getrennten Methoden das jeweilige `Validating`-Ereignis abgefangen (Listing 140). Im Fehlerfall wird die `SetError`-Methode der `ErrorProvider`-Komponente aufgerufen und verhindert, dass das Steuerelement den Fokus verliert.

---

```

Private Sub TBName_Validating(ByVal sender As Object, _
    ByVal e As System.ComponentModel.CancelEventArgs) _
    Handles TBName.Validating

    ' Der Eingabetext darf nicht leer sein
    If TBName.Text = "" Then

        ' Fehlermarkierung anzeigen
        ErrorProvider1.SetError(TBName, "Bitte Namen eingeben")

        ' Verlassen des Steuerelementes verhindern
        e.Cancel = True
    End If

End Sub

Private Sub DTPBirthday_Validating(ByVal sender As Object, _
    ByVal e As System.ComponentModel.CancelEventArgs) _
    Handles DTPBirthday.Validating

    If DateTime.Now.Subtract(DTPBirthday.Value).TotalDays _
        < 18 * 365 Then
        ' Mindestalter 18 Jahre
        ErrorProvider1.SetError(DTPBirthday, "Sie sind zu jung")
        e.Cancel = True
    End If

```

---

Listing 140: Individuelle Prüfung der Inhalte in den `Validating`-Eventhandlern



```

ElseIf DateTime.Now.Subtract(DTPBirthday.Value).TotalDays _
    > 36500 Then

    ' Höchstalter für die Eingabe 100 Jahre
    ErrorProvider1.SetError(DTPBirthday, "Sie sehen jünger aus")
    e.Cancel = True

End If

End Sub

Private Sub CHKAccept_Validating(ByVal sender As Object, _
    ByVal e As System.ComponentModel.CancelEventArgs) _
    Handles CHKAccept.Validating

    ' Der Haken in der CheckBox muss gesetzt sein
    If Not CHKAccept.Checked Then
        ' Fehlermarkierung anzeigen
        ErrorProvider1.SetError(CHKAccept, _
            "Sie müssen die Bedingungen akzeptieren")
        ' Verlassen des Steuerelementes verhindern
        e.Cancel = True
    End If
End Sub

```

Listing 140: Individuelle Prüfung der Inhalte in den Validating-Eventhandlern (Forts.)

Wird bei der Validierung eines Steuerelementes kein Fehler festgestellt, muss mit `SetError` eine eventuell vorhandene Markierung wieder gelöscht werden. Eine günstige Position für diesen Aufruf ist das `Validated`-Ereignis, das ja nur dann aufgerufen wird, wenn das `Validating`-Ereignis nicht mit einem Fehler abgebrochen wurde. Da die Vorgehensweise für alle Steuerelemente gleich ist, reicht eine gemeinsame Prozedur aus (Listing 141). Für das im Parameter `sender` übergebene Steuerelement wird dann die Fehlermeldung zurückgesetzt.

```

Private Sub Control_Validated(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles CHKAccept.Validated, _
    DTPBirthday.Validated, TBName.Validated

    ' Fehlermeldung löschen
    ErrorProvider1.SetError(DirectCast(sender, Control), "")

End Sub

```

Listing 141: Zurücksetzen der Fehlermeldung im gemeinsamen Validated-Ereignis

Wenn das Fenster geschlossen werden soll, dann wird der Eventhandler `Closing` aufgerufen (Listing 142). Hier wird zunächst überprüft, ob der Anwender die Eingaben akzeptieren oder verwerfen will. In letzterem Fall, also bei `DialogResult = Cancel`, soll das Fenster ohne weitere Prüfung geschlossen werden. Sollen die Eingaben übernommen werden, dann muss für jedes Steuerelement eine Gültigkeitsprüfung vorgenommen werden, damit auch die Steuerelemente

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Propor-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

erfasst werden, die bislang noch nicht den Eingabefokus hatten und somit noch nie überprüft worden sind.

---

```
Private Sub Dialog_Closing(ByVal sender As Object, _
    ByVal e As System.ComponentModel.CancelEventArgs) _
    Handles MyBase.Closing

    ' Wurde die Abbrechen-Schaltfläche gedrückt, dann
    ' das Schließen des Fensters zulassen
    If Me.DialogResult = DialogResult.Cancel Then Return

    ' Sonst alle Steuerelemente überprüfen
    CHKAccept_Validating(Me, e)
    DTPBirthday_Validating(Me, e)
    TBName_Validating(Me, e)

End Sub
```

---

*Listing 142: Überprüfung aller Steuerelemente beim Schließen des Fensters*

Für die Überprüfung können im einfachsten Fall die `Validating`-Methoden der Steuerelemente aufgerufen werden. Sie besitzen, genau wie die `Closing`-Ereignismethode, einen Parameter vom Typ `CancelEventArgs`. Dieser Parameter wird bei allen Aufrufen übergeben. Liegt ein Fehler bei einem oder mehreren Steuerelementen vor, dann hat anschließend die Eigenschaft `Cancel` den Wert `True`, liegt kein Fehler vor, `False`. Im `Closing`-Event führt das Setzen der `Cancel`-Eigenschaft auf `True` dazu, dass das Fenster nicht geschlossen wird.

Beachten sollten Sie ferner noch, wie Sie die Tabulator-Reihenfolge der Steuerelemente festlegen. Zum einen sollte sie natürlich so sein, dass der Anwender bequem mit der Tabulatortaste von links nach rechts und von oben nach unten navigieren kann, zum anderen ist es in Bezug auf die Validierung von großer Bedeutung, welches Steuerelement am Anfang dieser Reihenfolge steht. Denn dieses Steuerelement erhält den Fokus, wenn das Fenster angezeigt wird.

Wichtig ist die Betrachtung deswegen, weil der Anwender bei negativer Validierung dieses Steuerelement erst verlassen kann, wenn er korrekte Werte eingegeben hat. Wird also beispielsweise der `TextBox` der `TabIndex 0` zugewiesen, so dass sie automatisch den Fokus erhält, dann kann erst dann das Geburtsdatum geändert werden, wenn in der `TextBox` der Name eingetragen wurde. Das jedoch widerspricht der ersten Vorgabe, nach der der Anwender die Reihenfolge der Eingabe weitestgehend frei bestimmen können soll.

Daher sollte zu Beginn ein Steuerelement den Fokus erhalten, das entweder initial gültige Werte enthält oder das überhaupt nicht überprüft wird. Im Beispiel wird der Fokus zunächst der OK-Schaltfläche zugewiesen. So hat der Anwender die freie Wahl, bei welchem Steuerelement er die Eingabe beginnen möchte.

## 103 Screenshots erstellen

Im Framework finden Sie so gut wie keine Unterstützung für Zugriffe auf die Fenster anderer Prozesse. Daher sind ein paar API-Kniffe notwendig, will man den Inhalt anderer Fenster oder der gesamten Bildschirmarbeitsfläche kopieren. Screenshot-Programme gibt es viele auf dem Markt. Eines der bekanntesten, das auch zum Erstellen vieler Bilder in diesem Buch verwendet wurde, ist

Paint Shop Pro von Jasc Software. Dessen Umfang wollen wir in diesem Rezept aber nicht erreichen. Vielmehr geht es darum, die grundsätzlichen Vorgehensweisen zu zeigen.

Um mit API-Funktionen auf andere Fenster zugreifen zu können, benötigt man ein Handle. Im ersten Schritt wird daher eine Liste erzeugt, die für die interessanten Fenster jeweils den Titel und das zugehörige Handle erstellt. Die Informationen werden in Instanzen der Klasse `WindowInfo` (Listing 143) gespeichert. Alle benötigten Methoden sowie die Klasse `WindowInfo` sind innerhalb der Klasse `WindowManagement` gekapselt.

---

```

' Hilfsklasse mit Window-Informationen
Public Class WindowInfo
    ' Handle
    Public ReadOnly HWnd As IntPtr
    ' Text der Titelzeile
    Public ReadOnly Title As String

    ' Konstruktor nimmt Handle entgegen
    Sub New(ByVal hwnd As IntPtr)

        Me.HWnd = hwnd

        ' Text ermitteln und speichern
        Dim t As String = New String("x"c, 255)
        Dim i As Integer = API.GetWindowText(hwnd, t, 255)
        Title = t.Substring(0, i)

    End Sub

    Public Overrides Function ToString() As String
        Return Title
    End Function

End Class

```

---

Listing 143: Klasse `WindowInfo` speichert Handle und Titel eines Fensters

Das Erstellen der Fensterliste erfolgt in der Methode `GetInfoOfWindows` (Listing 144). Die Methode ruft in einer Schleife solange `GetWindow` auf, bis Null zurückgegeben wird. Für die Rückgabe wird ein Array mit `WindowInfo`-Objekten erstellt.

---

```

Public Shared Function GetInfoOfWindows(ByVal hwnd As IntPtr) _
    As WindowInfo()

    ' temporäre ArrayList
    Dim windowList As New ArrayList

    ' Schleife, bis kein Fenster mehr gefunden wird
    Do
        ' Wenn es ein Fenster gibt
        If Not hwnd.Equals(IntPtr.Zero) Then

```

---

Listing 144: Erstellen der Fensterliste

---

Basics

---

Datum/  
Zeit

---

Anwen-  
dungen

---

Zeich-  
nen

---

Bild-  
bearbei-  
tung

---

Win-  
dows  
Forms

---

Controls

---

Proper-  
tyGrid

---

Datei-  
system

---

Netz-  
werk

---

System

---

Daten-  
banken

---

XML

---

Wissen-  
schaft

---

Ver-  
schie-  
denes

---

```

If Not API.GetTopWindow(hwnd).Equals(IntPtr.Zero) Then

    ' neues WindowInfo-Objekt anlegen
    Dim wi As New WindowInfo(hwnd)

    ' Wenn es einen Titel hat, der Liste hinzufügen
    If wi.Title <> "" Then
        windowList.Add(wi)
    End If
End If

' Nächstes Fenster holen
hwnd = API.GetWindow(hwnd, API.GetWindowConstants.GW_HWNDNEXT)

Loop While Not hwnd.Equals(IntPtr.Zero) ' Bis hwnd 0 ist

' Array zurückgeben
Return windowList.ToArray(GetType(WindowInfo))

End Function

```

---

Listing 144: Erstellen der Fensterliste (Forts.)

Im Hauptfenster wird das Array einer ListBox hinzugefügt:

---

```

' Fensterliste füllen
LBWindows.Items.AddRange(WindowManagement.GetInfoOfWindows _
    (Me.Handle))

```

---

Abbildung 76 zeigt ein Abbild des Hauptfensters von sich selbst, das mit dem Programm erzeugt worden ist. Nach Auswahl eines Fensters aus der Liste wird der Capture-Vorgang durch Betätigen der Schaltfläche FENSTER KOPIEREN gestartet (Listing 145). In `WindowManagement.CopyWindowToBitmap` (Listing 146) wird das Bitmap-Objekt generiert. Anschließend wird mit `CopyForm.CreateAndShow` diese Bitmap in einem neuen Fenster angezeigt.

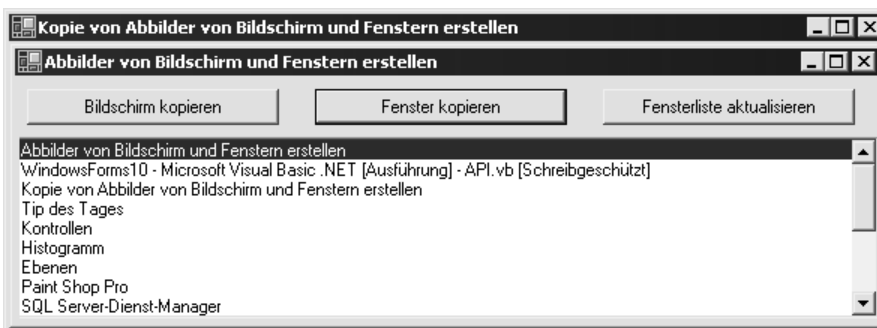


Abbildung 76: Images von Fenstern und Desktop erstellen

```

Private Sub BTNWindow_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles BTNWindow.Click

    ' Auswahl korrekt?
    If LBWindows.SelectedItem Is Nothing Then Exit Sub

    ' Cast auf WindowInfo
    Dim wi As WindowManagement.WindowInfo = DirectCast( _
        LBWindows.SelectedItem, WindowManagement.WindowInfo)

    ' Bitmap erzeugen
    Dim bmp As Bitmap = WindowManagement.CopyWindowToBitmap(wi.HWnd)

    ' Wenn kein Bitmap erzeugt wurde, abbrechen
    If bmp Is Nothing Then Exit Sub

    ' Fenster anzeigen
    CopyForm.CreateAndShow(bmp, "Kopie von " + wi.ToString())

End Sub

```

Listing 145: Event-Handler für Schaltfläche Fenster kopieren

Die Hauptaufgabe wird von der Methode `CopyWindowToBitmap` (Listing 146) erledigt. Sie ruft zunächst `SetForegroundWindow` und `ShowWindow` auf, um das Fenster im Vordergrund anzuzeigen, so dass es nicht durch andere Fenster verdeckt wird. Mit `GetWindowRect` werden Position und Größe ermittelt, mit `ScreenToClient` die Position in die Client-Koordinaten des Fensters umgerechnet. Entsprechend der ermittelten Größe wird ein neues Bitmap-Objekt angelegt. Für Bitmap und Fenster werden dann die Gerätekontexte (DC) angelegt und das Image des Fensters via `BitBlt` in das Bitmap-Objekt übertragen. Abschließend müssen die Ressourcen wieder freigegeben werden.

```

Public Shared Function CopyWindowToBitmap(ByVal hwnd As IntPtr) _
    As Bitmap

    ' Handle gültig?
    If hwnd.Equals(IntPtr.Zero) Then Return Nothing

    ' Das Fenster nach vorne holen
    API.SetForegroundWindow(hwnd)
    ' und anzeigen
    API.ShowWindow(hwnd, API.ShowWindowConstants.SW_SHOW)

    ' Größe und Position des Fensters ermitteln
    Dim rwnd As API.RECT
    API.GetWindowRect(hwnd, rwnd)
    Dim w As Integer = rwnd.Right - rwnd.Left
    Dim h As Integer = rwnd.Bottom - rwnd.Top

```

Listing 146: Kopieren eines Fensters in ein Bitmap-Objekt

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Propert-  
yGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

---

```

' Größenangaben plausibel?
If w <= 0 Or h <= 0 Then Exit Function

' Koordinatentransformation
Dim p As New API.POINTAPI
p.x = rwnd.Left
p.y = rwnd.Top
API.ScreenToClient(hwnd, p)

' Bitmap-Objekt in passender Größe anlegen
Dim bmp As New Bitmap(w, h)

' Graphics-Objekt hierzu
Dim g As Graphics = Graphics.FromImage(bmp)

' Gerätekontext hierzu
Dim hdcBmp As IntPtr = g.GetHdc()

' Gerätekontext für Fenster
Dim hdcWnd As IntPtr = API.GetDC(hwnd)

' Fensterinhalt in Gerätekontext des Bitmaps kopieren
' Geht alternativ auch mit StretchBlt
API.BitBlt(hdcBmp, 0, 0, w, h, hdcWnd, p.x, p.y, _
    API.ROPConstants.SRCCOPY)

' Aufräumen
API.ReleaseDC(hwnd, hdcWnd)
g.ReleaseHdc(hdcBmp)
g.Dispose()

' Bitmap zurückgeben
Return bmp

```

End Function

---

*Listing 146: Kopieren eines Fensters in ein Bitmap-Objekt (Forts.)*

CopyForm ist eine einfache Fensterklasse, die eine statische Methode zum Instanzieren und Anzeigen bereitstellt (CreateAndShow, siehe Listing 147). Auf dem Fenster wird eine PictureBox platziert, die die übergebene Bitmap anzeigt. Die PictureBox füllt den Client-Bereich des Fensters aus und bietet über ein Kontextmenü die Möglichkeit, das Bild zu speichern.

---

```

Public Shared Function CreateAndShow(ByVal bmp As Bitmap, _
    ByVal title As String) As CopyForm

    ' Neue Fensterinstanz anlegen
    Dim f As New CopyForm

    ' Bitmap der PictureBox zuweisen

```

---

*Listing 147: Anzeige des kopierten Bildes in einem Fenster vom Typ CopyForm*

```
f.PICWindowCopy.Image = bmp

' Fenstergröße
f.ClientSize = bmp.Size

' und Titel
f.Text = title

' Fenster nichtmodal anzeigen
f.Show()

' und Referenz zurückgeben
Return f
```

End Function

Listing 147: Anzeige des kopierten Bildes in einem Fenster vom Typ CopyForm (Forts.)

## Abbild der gesamten Bildschirmarbeitsfläche

Listing 148 zeigt den Aufruf zum Anlegen des vollständigen Screenshots. Hierfür wird mit `GetDesktopWindow` ein Handle ermittelt, das das Fenster für die gesamte Bildschirmarbeitsfläche repräsentiert. Der weitere Ablauf ist identisch mit dem Kopieren eines einfachen Fensters.

```
Public Shared Function GetScreenImage() As Bitmap
    ' Handle für gesamten Bildbereich holen
    Return CopyWindowToBitmap(API.GetDesktopWindow())
End Function
```

Listing 148: `GetDesktopWindow` liefert das Handle für den gesamten Bildbereich

Mit der gezeigten Vorgehensweise können Sie beliebige Fenster und Bildschirmausschnitte kopieren. Sie können das Beispiel erweitern und eine komfortablere Bedienoberfläche anbieten, die es dem Anwender erleichtert, Fenster und Bereiche auszuwählen.

## 104 TextViewer-Klasse

In vielen Situationen sind Hilfsklassen nützlich, die es erlauben, formatierte oder unformatierte Texte, HTML-Seiten oder Bilder in eigenständigen Fenstern anzuzeigen. Dieses Rezept beschreibt den Aufbau einer Klasse zur Ansicht von unformatierten Texten, die wahlweise als String, zeilenweise oder als `Stream` übergeben werden können. Ein Anwendungsbeispiel sehen Sie in Abbildung 77.

Als Basis dient eine gewöhnliche Windows-Form, auf der eine `TextBox` flächenfüllend platziert wird. Tabelle 15 zeigt die Werte der wichtigsten Eigenschaften der `TextBox`. Mehrere Überladungen der Methode `CreateAndShow` erlauben die Darstellung der Texte aus unterschiedlichen Quellen. Der Aufbau aller Überladungen folgt dem gleichen Muster:

1. Anlegen einer neuen Instanz der Fensterklasse
2. Zuweisung der Daten an das Steuerelement
3. Anzeigen des Fensters
4. Rückgabe der Objektreferenz zur weiteren Steuerung

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

In Listing 149 sehen Sie die Implementierung für die Anzeige eines als String übergebenen Textes. Dieser Text kann direkt der TextBox zugewiesen werden. Ähnlich verhält es sich mit der in Listing 150 gezeigten Variante, bei der ein übergebenes String-Array die anzuzeigenden Zeilen enthält. Wird dieses Array der Eigenschaft Lines der TextBox zugewiesen, dann wird jedes Array-Element in einer eigenen Zeile angezeigt.

| Eigenschaft | Wert        |
|-------------|-------------|
| Dock        | Fill        |
| Multiline   | True        |
| Scrollbars  | Both        |
| WordWrap    | False       |
| ReadOnly    | nach Bedarf |

Tabelle 15: Eigenschaften der TextBox in der TextViewer-Klasse

---

```
Public Shared Function CreateAndShow(ByVal text As String) _
    As Textviewer

    ' Fensterinstanz anlegen
    Dim f As New Textviewer

    ' Text setzen
    f.TBText.Text = text

    ' Fenster anzeigen, Referenz zurückgeben
    f.Show()
    Return f

End Function
```

---

Listing 149: Anlegen und Anzeigen eines Fensters zur Darstellung unformatierter Texte

---

```
Public Shared Function CreateAndShow(ByVal lines() As String) _
    As Textviewer

    'Fensterinstanz anlegen
    Dim f As New Textviewer

    ' String-Array als Zeilen setzen
    f.TBText.Lines = lines

    ' Fenster anzeigen, Referenz zurückgeben
    f.Show()
    Return f

End Function
```

---

Listing 150: String-Array zeilenweise anzeigen



Für die Übergabe eines Streams steht die dritte Variante (Listing 151) zur Verfügung. Sie verwendet ein `StreamReader`-Objekt zum Lesen des Textes aus dem Stream mit Hilfe der Methode `ReadToEnd`. In der letzten Variante (Listing 152) werden die Daten als allgemeine `Object`-Referenz übergeben und die zum Datentyp passende Methode aufgerufen.

```
Public Shared Function CreateAndShow(ByVal str As Stream) _
    As TextViewer

    Try
        ' Streamreader zum Lesen des Streams anlegen
        Dim sr As New StreamReader(str)

        ' Stream vollständig lesen
        Dim t As String = sr.ReadToEnd()

        sr.Close()
        str.Close()

        ' Fenster anzeigen, Referenz zurückgeben
        Return CreateAndShow(t)

    Catch ex As Exception
        MessageBox.Show("Fehler beim Öffnen des Streams: " & ex.Message)
    End Try

End Function
```

Listing 151: Anzeigen eines Textes aus einem Stream

```
Public Shared Function CreateAndShow(ByVal obj As Object) _
    As TextViewer

    ' Passenden Aufruf an Hand des Datentyps ermitteln
    If TypeOf obj Is String Then Return _
        CreateAndShow(DirectCast(obj, String))
    If TypeOf obj Is String() Then Return _
        CreateAndShow(DirectCast(obj, String()))
    If TypeOf obj Is Stream Then Return _
        CreateAndShow(DirectCast(obj, Stream))
    Return Nothing

End Function
```

Listing 152: Parameterübergabe als `Object`-Referenz

## 105 RTFTextViewer-Klasse

Zur Darstellung im Rich Text Format (RTF) formatierter Texte eignet sich das `RichTextBox`-Steuerelement. Analog zur `TextViewer`-Klasse wird ein `RichTextBox`-Steuerelement flächendeckend auf dem Fenster platziert. Wieder stehen vier Überladungen der Methode `CreateAndShow` (Listing 153) zur Verfügung, um, je nach Bedarf, den Text aus unterschiedlichen Quellen anzeigen zu können. Abbildung 78 zeigt als Beispiel die Darstellung des als formatierten Text kopierten Listings.

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

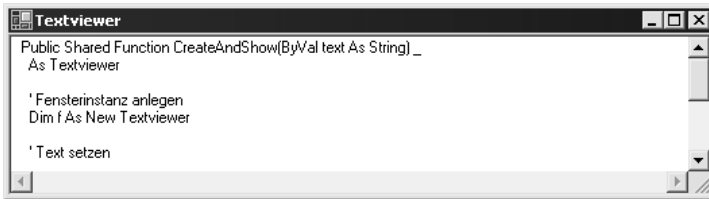


Abbildung 77: TextViewer zum Anzeigen unformatierter Texte

---

```
Public Shared Function CreateAndShow(ByVal text As String) _
    As RTFTextViewer

    Dim f As New RTFTextViewer

    ' Text zuweisen
    f.RTFText.Rtf = text

    ' Fenster anzeigen und Referenz zurückgeben
    f.Show()
    Return f
End Function
```

```
Public Shared Function CreateAndShow(ByVal lines() As String) _
    As RTFTextViewer

    Dim f As New RTFTextViewer

    ' Zeilen zuweisen
    f.RTFText.Lines = lines

    ' Fenster anzeigen und Referenz zurückgeben
    f.Show()
    Return f
End Function
```

```
Public Shared Function CreateAndShow(ByVal str As Stream) _
    As RTFTextViewer

    Try
        ' Streamreader zum Lesen des Streams anlegen
        Dim sr As New StreamReader(str)

        ' Stream vollständig lesen
        Dim t As String = sr.ReadToEnd()

        sr.Close()
        str.Close()
    End Try
End Function
```

---

Listing 153: RTFTextViewer zur Anzeige formatierter Texte

```

' Fenster anzeigen und Referenz zurückgeben
Return CreateAndShow(t)

Catch ex As Exception
    MessageBox.Show("Fehler beim Öffnen des Streams: " & ex.Message)
End Try

End Function

Public Shared Function CreateAndShow(ByVal obj As Object) _
    As RTFTextViewer

    ' Passenden Aufruf an Hand des Datentyps ermitteln
    If TypeOf obj Is String Then Return _
        CreateAndShow(DirectCast(obj, String))
    If TypeOf obj Is String() Then Return _
        CreateAndShow(DirectCast(obj, String()))
    If TypeOf obj Is Stream Then Return _
        CreateAndShow(DirectCast(obj, Stream))
    Return Nothing
End Function

```

Listing 153: RTFTextViewer zur Anzeige formatierter Texte (Forts.)

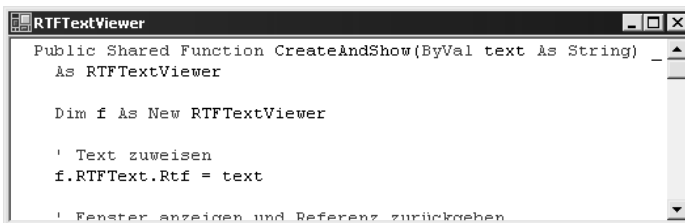


Abbildung 78: Formatierter Text im RTFTextViewer

## 106 PictureViewer-Klasse

Wie der Name erwarten lässt, dient diese Klasse zur Darstellung von Bildern. Auch sie ist ähnlich der Klasse `TextViewer` aufgebaut, verwendet die übergebenen Parameter aber etwas anders. Bei der Übergabe eines Strings (Listing 154) wird dieser als *URI* (Uniform Resource Identifier) interpretiert. Es kann sich dabei wahlweise um einen Dateipfad oder eine Internet-Adresse handeln.

An Hand der Eigenschaft `IsFile` wird unterschieden, wie das angegebene Bild zu öffnen ist. Im Falle einer Datei (lokal oder im Netzwerk) kann der Pfad direkt dem Konstruktor der `Bitmap`-Klasse übergeben werden. Handelt es sich um eine Internet-Adresse, sind einige Zwischenschritte notwendig. Über ein `WebRequest` wird zunächst die Datei angefordert und anschließend der mit `GetResponse` bzw. `GetResponseStream` zurückgegebene Stream dem Konstruktor der `Bitmap`-Klasse übergeben.

Analog zu den zuvor beschriebenen Viewer-Klassen wird das Bild auf einer `PictureBox` dargestellt, die den gesamten Client-Bereich des Fensters ausfüllt. Zusätzlich wird das Fenster so vergrößert, dass das Bild in voller Größe dargestellt wird. Abbildung 79 zeigt eine Beispielanwendung der Klasse.

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Propor-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

```

Public Shared Function CreateAndShow(ByVal filename As String) _
    As PictureBox

    ' URI-Instanz aus Dateinamen bilden
    Dim picUri As New Uri(filename)

    ' Neue Fensterinstanz
    Dim f As New PictureBox

    Try

        ' Wenn der Parameter eine lokale Datei ist, direkt
        ' als Bitmap laden
        If picUri.IsFile Then
            f.PBPicture.Image = New Bitmap(filename)
        Else
            ' Sonst Datei via WebRequest anfordern
            Dim wr As WebRequest = WebRequest.Create(filename)
            ' Auf Response warten
            Dim ws As WebResponse = wr.GetResponse()
            ' Bitmap aus Stream erstellen
            Dim s As Stream = ws.GetResponseStream()
            f.PBPicture.Image = New Bitmap(s)
        End If

    Catch ex As Exception
        MessageBox.Show(ex.Message)
    End Try

    ' Fenstergröße setzen
    f.ClientSize = f.PBPicture.Image.Size

    ' Fenster anzeigen und Referenz zurückgeben
    f.Show()
    Return f

End Function

```

---

*Listing 154: Als Datenquelle für den PictureBox kann wahlweise eine Datei oder eine Internet-Adresse angegeben werden*

Auch ein als Parameter übergebenes String-Array wird anders interpretiert. Für jedes Element wird ein eigenes Fenster geöffnet und das betreffende Bild darin angezeigt (Listing 155).

---

```

Public Shared Function CreateAndShow(ByVal files As String()) _
    As PictureBox

    Dim f As PictureBox

    ' String-Array als Liste von Bilddateien betrachten

```

---

*Listing 155: Anzeigen mehrerer Bilder*

```

For Each file As String In files
    ' Jedes Bild in eigenem Fenster öffnen
    f = CreateAndShow(file)
Next

' Referenz des letzten zurückgeben
Return f

```

End Function

Listing 155: Anzeigen mehrerer Bilder (Forts.)



Abbildung 79: PictureViewer zum Anzeigen von Bildern

Wiederum besteht die Möglichkeit, einen Stream als Parameter zu übergeben. Wie in Rezept 6.16 ((Analyseprogramm für Drag&Drop-Operationen aus anderen Anwendungen)) noch gezeigt wird, liegt oft bei Drag&Drop-Vorgängen in Verbindung mit einem Internet-Browser die Adresse eines Bildes als Stream vor. Daher wird in dieser Überladung nicht das Bild als Stream erwartet, sondern eine Internet-Adresse. Das Einlesen ist etwas aufwändiger, da zusätzliche Control-Zeichen aus dem String entfernt werden müssen. Listing 156 zeigt die vollständige Funktion.

```

Public Shared Function CreateAndShow(ByVal str As Stream) _
    As PictureViewer

    ' Stream mit StreamReader lesen
    Dim sr As New StreamReader(str)
    Dim buf(500) As Char
    sr.Read(buf, 0, 500)
    Dim t As String

    ' Control-Character suchen
    For i As Integer = 0 To buf.GetUpperBound(0)
        If Char.IsControl(buf(i)) Then
            ' String abschneiden
            t = New String(buf, 0, i)
            Exit For
        End If
    Next

```

Listing 156: Interpretation des Streams als Bildadresse

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

Next

```
sr.Close()
str.Close()
```

```
Return CreateAndShow(t)
```

```
End Function
```

---

*Listing 156: Interpretation des Streams als Bildadresse (Forts.)*

Zusätzlich zu den Überladungen, die die anderen Viewer bieten, implementiert die `PictureViewer`-Klasse noch eine Variante, die eine `Bitmap`-Referenz als Datentyp entgegennimmt und anzeigt (Listing 157).

---

```
Public Shared Function CreateAndShow(ByVal pic As Bitmap) _
    As PictureBox
```

```
    ' Neue Fensterinstanz
    Dim f As New PictureBox
```

```
    ' Bild zuweisen
    f.PBPicture.Image = pic
```

```
    ' Fenstergröße setzen
    f.ClientSize = f.PBPicture.Image.Size
```

```
    ' Fenster anzeigen und Referenz zurückgeben
    f.Show()
    Return f
```

```
End Function
```

---

*Listing 157: Direkte Anzeige eines als Bitmap vorliegenden Bildes*

Die fünfte Überladung nimmt wie in den vorangegangenen Beispielen den Parameter als `Object`-Referenz entgegen und nimmt die Verzweigung an Hand des Datentyps vor. Der Code ist nahezu identisch mit dem des `TextViewers` in Listing 157.

## 107 HTML-Viewer

Die letzte Viewer-Klasse in dieser Runde dient zur Anzeige von HTML-Inhalten. Zur Darstellung wird das `ActiveX WebBrowser`-Steuerelement eingesetzt. Zusätzlich zur Anzeige von HTML-Texten kann es auch zur weiteren Navigation über angezeigte Links benutzt werden. Es stehen nahezu dieselben Bedienfunktionen zur Verfügung, die der Internet Explorer bietet. Eine praktische Anwendung zeigt Abbildung 80.

Ein als `String` übergebener Parameter kann wahlweise eine Internet-Adresse oder der HTML-Text selbst sein (siehe Listing 158). Zur Unterscheidung wird untersucht, ob der Text das einleitende `html`-Tag enthält. In diesem Fall wird der Text in einer temporären Datei gespeichert und der Inhalt dieser Datei mittels `WebBrowser.Navigate` angezeigt. Die Datei wird beim Schließen des Fensters wieder gelöscht.

Ist das Tag nicht enthalten, wird der Text als Adresse interpretiert. Erlaubt sind alle Adressangaben, die auch der Internet Explorer interpretieren kann. Dazu gehören neben HTML-Seiten auch Bilder und FTP-Adressen (zumindest wenn der IE6 installiert ist).

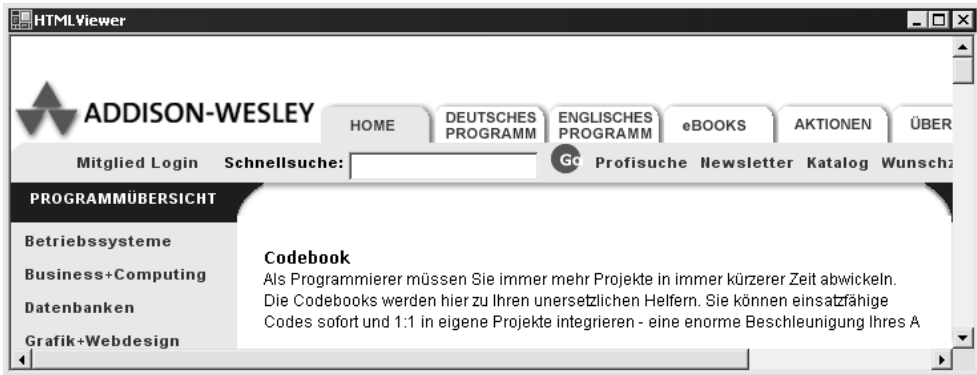


Abbildung 80: Einsatz des HTML-Viewers

```
Public Shared Function CreateAndShow(ByVal text As String) _
    As HTMLViewer

    ' Fensterinstanz erstellen
    Dim f As New HTMLViewer

    ' Enthält Text HTML-Tag?
    If text.ToLower().IndexOf("<html") < 0 Then
        ' Nein, Text als URL interpretieren
        f.WebBrowser.Navigate(text)
        ' Fenster anzeigen
        f.Show()
    Else
        ' Ja, Text in temporäre Datei kopieren
        f.FileName = Path.GetTempFileName() & ".html"
        Dim sw As New StreamWriter(f.FileName, False, _
            System.Text.Encoding.Default)
        sw.Write(text)
        sw.Close()

        ' Datei mit WebBrowser öffnen
        f.WebBrowser.Navigate(f.FileName)
        ' Fenster anzeigen
        f.Show()
    End If

    ' Referenz zurückgeben
    Return f

End Function
```

Listing 158: Wahlweise Übergabe eines HTML-Textes oder einer Adresse

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Propor-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schie-  
denes

Ein String-Array als Parameter wird wie bei der `PictureViewer`-Klasse als Liste von HTML-Quellen interpretiert (Listing 159). Für jede Quelle wird ein eigenes Fenster geöffnet.

---

```
Public Shared Function CreateAndShow(ByVal urls() As String) _
    As HTMLViewer

    Dim f As HTMLViewer
    ' String-Array mit mehreren URLs durchlaufen
    For Each url As String In urls

        ' Jeweils eine Fensterinstanz anlegen
        f = CreateAndShow(url)

    Next

    ' Referenz des letzten zurückgeben
    Return f

End Function
```

---

Listing 159: Anzeigen mehrerer HTML-Quellen

Ein Stream als Parameter wird analog zur Klasse `PictureViewer` als Adresse interpretiert (Listing 160). Wiederum müssen Kontrollzeichen entfernt werden, damit das WebBrowser-Steuerelement die Adresse richtig interpretieren kann.

---

```
Public Shared Function CreateAndShow(ByVal str As Stream) _
    As HTMLViewer

    Try
        ' URL in MemoryStream
        ' Mit Reader öffnen
        Dim sr As New StreamReader(str)

        ' max. 500 Zeichen einlesen
        Dim buf(500) As Char
        sr.Read(buf, 0, 500)
        Dim t As String
        ' Nach dem ersten Control-Zeichen suchen
        For i As Integer = 0 To buf.GetUpperBound(0)
            If Char.IsControl(buf(i)) Then
                ' Text ohne Control-Zeichen speichern
                t = New String(buf, 0, i)
                Exit For
            End If
        Next

        sr.Close()
        str.Close()
```

---

Listing 160: Adresse einer HTML-Quelle aus einem Stream entgegennehmen



```
' Neue Fensterinstanz
Dim f As New HTMLViewer

' URL im Browser öffnen und Fenster anzeigen
f.WebBrowser.Navigate(t)
f.Show()

' Referenz zurückgeben
Return f

Catch ex As Exception
    MessageBox.Show("Fehler beim Öffnen des Streams: " & ex.Message)
End Try

End Function
```

Listing 160: Adresse einer HTML-Quelle aus einem Stream entgegennehmen (Forts.)

# 108 Drag&Drop-Operationen aus anderen Anwendungen ermöglichen

Bereits die 16-Bit-Vorgänger der heutigen Windows-Versionen unterstützten den Drag&Drop-Mechanismus, der es erlaubt, auf grafischem Weg komfortabel Objekte von einem Punkt zu einem anderen zu bewegen. Die mit dem Ziehen der Objekte verbundenen Operationen sind vom Kontext abhängig und sollten intuitiv nutzbar sein. So führt beispielsweise das Ziehen einer Textdatei aus dem Explorer auf ein Textverarbeitungsprogramm zum Öffnen der Datei. Ist das Ziel der Papierkorb, wird sie gelöscht.

Selbstverständlich kann auch ein Fenster oder ein Steuerelement einer .NET-Anwendung das Ziel einer Drag&Drop-Operation sein. Die Technik basiert auf dem (ur-)alten OLE (Object Linking and Embedding)-Verfahren. Verschiedene Ereignisse werden beim Umgang mit Drag&Drop benötigt (siehe Tabelle 16). Jedes Steuerelement kann als Ziel definiert werden, indem die Eigenschaft AllowDrop auf True gesetzt wird.

| Ereignis     | Empfänger | Bedeutung                                                                                                                   |
|--------------|-----------|-----------------------------------------------------------------------------------------------------------------------------|
| DragEnter    | Ziel      | Ein Objekt wird in den Bereich eines Steuerelementes gezogen                                                                |
| DragOver     | Ziel      | Ein Objekt wird innerhalb der Begrenzungen des Steuerelementes gezogen                                                      |
| DragLeave    | Ziel      | Ein Objekt wird aus dem Bereich eines Steuerelementes herausgezogen                                                         |
| DragDrop     | Ziel      | Ein Objekt wird auf dem Steuerelement abgelegt                                                                              |
| GiveFeedback | Quelle    | ermöglicht der Quelle während des Drag&Drop-Vorgangs ein optisches Feedback, z. B. durch Änderung des Mauszeigers, zu geben |

Tabelle 16: Wichtige Ereignisse beim Drag&Drop-Vorgang

Ist ein Fenster oder ein Steuerelement ein Container für andere Steuerelemente, dann empfängt es auch die Drag&Drop-Ereignisse aller untergeordneten Steuerelemente, deren AllowDrop-Eigen-

schaft den Wert `False` aufweist. So gestaltet sich die Handhabung komplexer Fenster recht einfach, da nicht für jedes Steuerelement die entsprechenden Ereignisse berücksichtigt werden müssen.

Das erste Ereignis, das im Zielobjekt des Drag&Drop-Vorgangs ausgelöst wird, ist `DragEnter`. Dem Ereignis-Handler wird ein Parameter vom Typ `DragEventArgs` übergeben (siehe Tabelle 17). Von der Quelle wird mit der schreibgeschützten Eigenschaft `AllowedEffect` mitgeteilt, welche Operationen möglich sind. `AllowedEffect` ist eine Oder-Verknüpfung der in Tabelle 18 aufgeführten Operationen. Nur die Operationen, deren Bit gesetzt ist, können verwendet werden. In der Eigenschaft `Effect` kann das Ziel der Drag&Drop-Operation die gewünschten Effekte eintragen (siehe Beispiel in Listing 161). Von der Quelle können dann Maßnahmen zur Visualisierung (z. B. durch Ändern des Mauszeigers) vorgenommen und so dem Anwender ein optisches Feedback für seine Aktion gegeben werden.

---

```
Protected Overrides Sub OnDragEnter(ByVal drgevent As _
    System.Windows.Forms.DragEventArgs)

    ' Gewünschten Effekt festlegen
    drgevent.Effect = DragDropEffects.Copy
End Sub
```

---

Listing 161: Im `DragEnter`-Event wird definiert, welche Drag&Drop-Operationen zulässig sind

| Eigenschaft                | Bedeutung                                                             |
|----------------------------|-----------------------------------------------------------------------|
| <code>AllowedEffect</code> | gibt die möglichen Drag&Drop-Operationen an ( <code>ReadOnly</code> ) |
| <code>Effect</code>        | gibt die vom Ziel gewünschten Operationen an                          |
| <code>Data</code>          | Referenz der Daten über die <code>IDataObject</code> -Schnittstelle   |
| <code>KeyState</code>      | Zustand der Tasten                                                    |
| <code>X, Y</code>          | Position des Mauszeigers (in Bildschirmkoordinaten!)                  |

Tabelle 17: Eigenschaften der `DragEventArgs`-Parameter

| Konstante         | Wert | Bedeutung                                                                             |
|-------------------|------|---------------------------------------------------------------------------------------|
| <code>Copy</code> | 1    | Kopieren (z. B. Datei von Festplatte auf Diskette)                                    |
| <code>Move</code> | 2    | Verschieben (z. B. Datei von einem Verzeichnis in ein anderes)                        |
| <code>Link</code> | 4    | Verknüpfung (z. B. bei Office-Anwendungen oder beim Ziehen von Links aus HTML-Seiten) |
| <code>None</code> | 0    | Keine Aktion                                                                          |

Tabelle 18: Konstanten der `DragDropEffects`-Enumeration

## Daten der Drag&Drop-Operationen

Die eigentlichen Daten können von der Quelle in verschiedenen Formaten bereitgestellt werden. So kann ein Text beispielsweise als unformatierter Text, als formatierter Text oder als Verknüpfung angeboten werden. Das Ziel der Drag&Drop-Operation kann selbst wählen, welches Format es benutzen möchte. Übergeben wird stets eine Referenz eines Objektes, das die Schnittstelle `IDa-`

taObject implementiert. Vier Methoden (siehe Tabelle 19) gestatten den Zugriff auf die Daten bzw. Datenformate.

| Methode        | Bedeutung                                             |
|----------------|-------------------------------------------------------|
| GetFormats     | Gibt die Liste der verfügbaren Datenformate zurück    |
| GetDataPresent | Gibt an, ob für ein bestimmtes Format Daten vorliegen |
| GetData        | Ruft die Daten zu einem angegebenen Format ab         |

Tabelle 19: Abfrage von Daten und Datenformaten bei Drag&Drop-Operationen

GetFormats liefert ein String-Array, das die Bezeichnungen aller unterstützten Formate enthält (siehe Beispielcode in Listing 162). Werden beispielsweise Dateien aus dem Windows Explorer über Drag&Drop kopiert oder verschoben, liefert der Beispielcode die folgende Ausgabe:

```
Shell IDList Array
Shell Object Offsets
InShellDragLoop
DragContext
FileDrop
FileNameW
FileName
```

Für diese Formate kann mit GetDataPresent abgefragt werden, ob auch tatsächlich Daten vorhanden sind. Mit GetData können diese Daten abgerufen werden. GetData liefert eine allgemeine Object-Referenz der Daten, deren Datentyp zur Laufzeit analysiert werden muss. Typische .NET-Datentypen sind String, String-Array, MemoryStream oder Bitmap. Wie die Daten aufgebaut sind, ist in keiner Weise festgelegt und wird von unterschiedlichen Programmen auch verschieden gehandhabt.

```
Protected Overrides Sub OnDragDrop(ByVal drgevent As _
    System.Windows.Forms.DragEventArgs)

    For Each frmt As String In drgevent.Data.GetFormats()
        Debug.WriteLine(frmt)
    Next
End Sub
```

Listing 162: Ermitteln der von der Quelle zur Verfügung gestellten Formate

# 109 Analyseprogramm für Drag&Drop-Operationen aus anderen Anwendungen

Ein kleines Analyseprogramm erweist praktische Dienste, wenn es darum geht, die via Drag&Drop übergebenen Daten aus anderen Anwendungen zu entschlüsseln. Das Hauptfenster des Programms empfängt alle externen Drag&Drop-Ereignisse, zeigt die möglichen Effekte an und listet die übergebenen Datenformate in einem ListView-Steuerelement auf. Die Daten lassen sich mittels der in den vorangegangenen Kapiteln beschriebenen Viewer-Klassen einsehen. Abbildung 81 zeigt das Programm im praktischen Einsatz.

|                 |
|-----------------|
| Basics          |
| Datum/Zeit      |
| Anwendungen     |
| Zeichen         |
| Bildbearbeitung |
| Windows Forms   |
| Controls        |
| PropertyGrid    |
| Dateisystem     |
| Netzwerk        |
| System          |
| Datenbanken     |
| XML             |
| Wissenschaft    |
| Verschiedenes   |

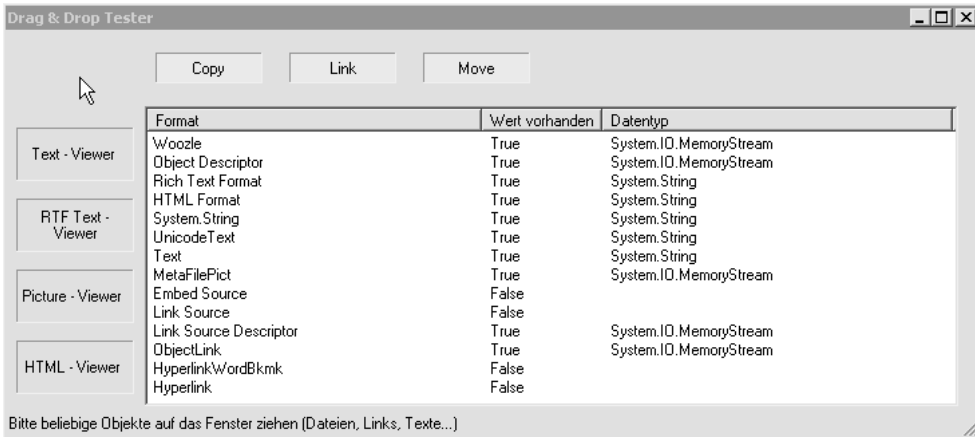


Abbildung 81: Anzeige der Datenformate und erlaubten Effekte im Analyseprogramm

Im `DragEnter`-Ereignis des Fensters werden die drei Labels zur Anzeige der verfügbaren Effekte angezeigt oder verborgen. Für Ziehvorgänge auf das Fenster wird grundsätzlich der Wert der Eigenschaft `AllowedEffect` in die Eigenschaft `Effect` übertragen. Nur Ziehvorgänge von außerhalb werden berücksichtigt. Listing 163 zeigt die Implementierung des Ereignis-Handlers.

```
Private Sub Mainwindow_DragEnter(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.DragEventArgs) _
    Handles MyBase.DragEnter

    ' Wenn der Drag-Vorgang von der ListView selbst ausgelöst worden
    ' ist, ignorieren
    Dim lvi As ListViewItem = e.Data.GetData(GetType(ListViewItem))
    If Not lvi Is Nothing Then Exit Sub

    ' Effekte setzen
    e.Effect = e.AllowedEffect

    ' Labels entsprechend der erlaubten Effekte ein-/ausschalten
    LBLCopy.Visible = _
        (e.AllowedEffect And DragDropEffects.Copy) <> 0
    LBLLink.Visible = _
        (e.AllowedEffect And DragDropEffects.Link) <> 0
    LBLMove.Visible = _
        (e.AllowedEffect And DragDropEffects.Move) <> 0

End Sub
```

Listing 163: Feedback an Quelle und Anzeigen bzw. Ausblenden der Labels

Wird ein Objekt auf dem Fenster abgelegt, dann wird im `DragDrop`-Ereignis ermittelt, welche Datenformate unterstützt werden (Listing 164). Die angegebenen Formatbezeichnungen werden im `ListView`-Steuerelement angezeigt. Für alle Formate wird mittels `GetData` versucht, die Daten abzurufen. Ob dies erfolgreich ist, wird in der zweiten Spalte dargestellt, der Datentyp der Daten, wie er zur Verfügung gestellt wird, in der dritten Spalte. In der `Tag`-Eigenschaft des jeweiligen

ListViewItem-Objektes wird die Referenz auf das Typ-Objekt der Daten für die weitere Verarbeitung gespeichert. Die Referenz auf das Daten-Objekt wird in der Member-Variablen DroppedData festgehalten.

```

Private Sub Mainwindow_DragDrop(ByVal sender As System.Object, _
    ByVal e As System.Windows.Forms.DragEventArgs) _
    Handles MyBase.DragDrop, LBLCopy.DragDrop, LBLLink.DragDrop, _
    LBLMove.DragDrop

    ' Behandlung in SetData
    SetData(e.Data)

End Sub

Protected Sub SetData(ByVal data As IDataObject)

    ' Referenz merken
    DroppedData = data

    ' Unterstützte Formate ermitteln
    Dim formats() As String = data.GetFormats()

    Dim o As Object

    LVDragDropData.Items.Clear()

    ' Für alle Formate
    For Each t As String In formats

        ' Daten lesen
        o = data.GetData(t)

        ' ListView-Eintrag erstellen
        ' 1. Format
        Dim lvi As ListViewItem = LVDragDropData.Items.Add(t)

        ' 2. Daten vorhanden
        lvi.SubItems.Add((Not o Is Nothing).ToString())
        If Not o Is Nothing Then
            ' 3. Datentyp der Daten
            lvi.SubItems.Add(o.GetType().ToString())
            lvi.Tag = o.GetType
        End If

    Next

End Sub

```

Listing 164: Reaktion auf das DragDrop-Ereignis und Anzeigen von Informationen zu den verfügbaren Datenformaten

---

Basics

---

Datum/  
Zeit

---

Anwen-  
dungen

---

Zeich-  
nen

---

Bild-  
bearbei-  
tung

---

Win-  
dows  
Forms

---

Controls

---

Proper-  
tyGrid

---

Datei-  
system

---

Netz-  
werk

---

System

---

Daten-  
banken

---

XML

---

Wissen-  
schaft

---

Ver-  
schiede-  
nes

---

Auch die drei Labels, die anzeigen, welche der Drag&Drop-Effekte erlaubt sind, können Ziel der Operation sein. Wie Ihnen vielleicht schon aufgefallen ist, wird das DragDrop-Ereignis der drei Labels ebenfalls in `Mainwindow_DragDrop` (siehe Listing 164) abgehandelt. Das `DragEnter`-Ereignis behandelt jedoch jedes Label unterschiedlich (Listing 165). Nur die einem Label zugeordnete Operation wird unterstützt. Beim Ziehen von Objekten auf die jeweiligen Labels sehen Sie daher den von der Quelle im `GiveFeedback`-Ereignis ausgewählten Maus-Cursor.

---

```
Private Sub LBLCopy_DragEnter(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.DragEventArgs) _
    Handles LBLCopy.DragEnter

    ' Ignorieren, wenn die Quelle die ListView ist
    Dim lvi As ListViewItem = e.Data.GetData(GetType(ListViewItem))
    If Not lvi Is Nothing Then Exit Sub

    ' Nur Kopieren erlauben
    e.Effect = DragDropEffects.Copy

End Sub

Private Sub LBLLink_DragEnter(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.DragEventArgs) _
    Handles LBLLink.DragEnter

    ' Nur Link erlauben
    e.Effect = DragDropEffects.Link

End Sub

Private Sub LBLMove_DragEnter(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.DragEventArgs) _
    Handles LBLMove
```

---

*Listing 165: Jedes der drei Labels unterstützt nur genau eine Drag&Drop-Operation*

## Anzeige der Daten

Links neben der Liste der verfügbaren Formate befinden sich vier Labels, denen jeweils einer der in den vorangegangenen Rezepten beschriebenen `TextViewer`, `RTFTextViewer`, `PictureViewer` bzw. `HTMLViewer` zugeordnet ist. Wiederum mittels Drag&Drop lässt sich ein Eintrag der Liste auf die Label ziehen. Dadurch wird der entsprechende `Viewer` angezeigt und versucht, die Daten darzustellen. Stellvertretend für alle vier zeigt Listing 166 die Implementierung für das `TextViewer`-Label. Um den Drag&Drop-Vorgang einzuleiten, wird im `ItemDrag`-Ereignis des `ListView-Controls` die Methode `DoDragDrop` aufgerufen (Listing 167).

---

```
Private Sub LBLTextViewer_DragEnter(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.DragEventArgs) _
    Handles LBLTextViewer.DragEnter

    ' TextViewer unterstützt String, String[] und MemoryStream
```

---

*Listing 166: DragEnter- und Drag&Drop-Ereignis-Handler für das dem TextViewer zugeordnete Label*

```

Dim lvi As ListViewItem = e.Data.GetData(GetType(ListViewItem))
If lvi Is Nothing Then Exit Sub
If lvi.Tag Is GetType(String) Or lvi.Tag Is GetType(String()) _
    Or lvi.Tag Is GetType(MemoryStream) Then
    e.Effect = DragDropEffects.Copy
End If

End Sub

Private Sub LBLTextViewer_DragDrop(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.DragEventArgs) _
    Handles LBLTextViewer.DragDrop

    ' Drag&Drop nur von der ListView akzeptieren
    Dim lvi As ListViewItem = e.Data.GetData(GetType(ListViewItem))
    If lvi Is Nothing Then Exit Sub

    ' TextViewer aufrufen
    TextViewer.CreateAndShow(DroppedData.GetData(lvi.Text))

End Sub

```

*Listing 166: DragEnter- und Drag&Drop-Ereignis-Handler für das dem TextViewer zugeordnete Label (Forts.)*

```

Private Sub LVDragDropData_ItemDrag(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.ItemDragEventArgs) _
    Handles LVDragDropData.ItemDrag

    ' Drag&Drop einleiten
    LVDragDropData.DoDragDrop(e.Item, DragDropEffects.Copy)

End Sub

```

*Listing 167: Einleitung der Drag&Drop-Operation mit DoDragDrop*

In `DragEnter` wird überprüft, ob es sich bei den Daten um ein `ListViewItem`-Objekt handelt. Nur dann wird die `Copy`-Operation zugelassen. Ferner wird überprüft, ob das in der `Tag`-Eigenschaft des `ListViewItem`-Objektes referenzierte Typ-Objekt einen vom Viewer unterstützten Datentyp repräsentiert. Abbildung 82 bis Abbildung 85 zeigen einige Beispiele für typische Drag&Drop-Operationen.

Werden aus dem Windows Explorer Dateien auf ein Zielfenster gezogen, dann kann die Liste der Dateien als `String`-Array unter dem Format `FileDrop` abgerufen werden (Abbildung 83). Beim Ziehen von Links aus dem Internet Explorer wird die Adresse als Stream unter dem Formatnamen `UniformResourceLocator` angeboten. Viele Programme stellen zahlreiche Alternativen für formatierte Daten zur Verfügung, von denen Sie sich die für Sie günstigste auswählen können. Das Analyseprogramm hilft Ihnen bei der Auswahl der Formate.

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Proper-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

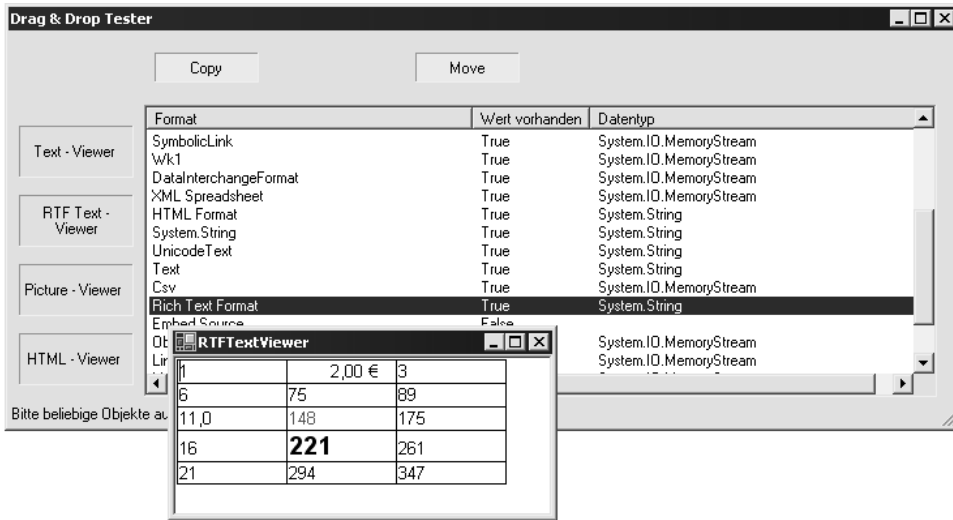


Abbildung 82: Excel 2003 unterstützt über 25 Formate, hier eine Ansicht im Rich Text Format

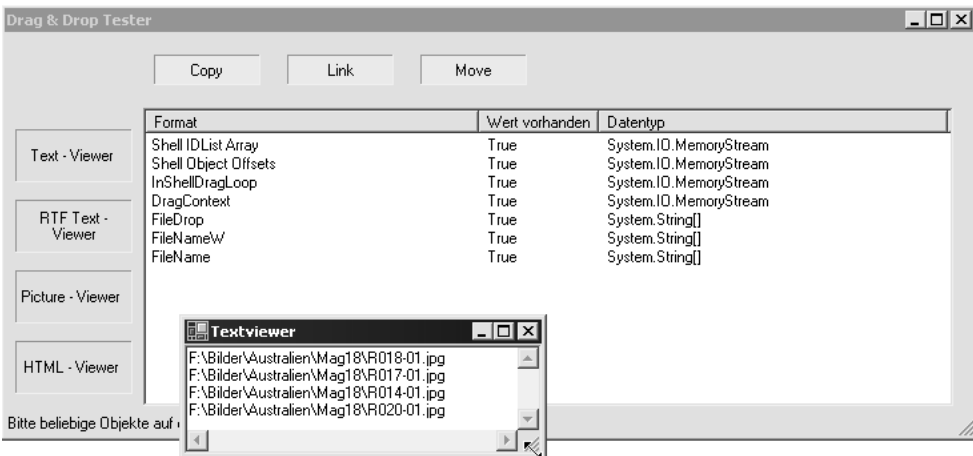


Abbildung 83: Ziehen von Dateien aus dem Explorer und Darstellung der Liste im TextViewer

## Besonderheiten

Einige Programme verhalten sich leider etwas sonderbar und verhindern, dass die Daten vom Analyseprogramm korrekt angezeigt werden können. Das Problem besteht darin, dass manche Programme, wie z.B. Microsoft Word, die Daten nur während des Ziehvorgangs zur Verfügung stellen. Nach Abschluss des DragDrop-Events liefert Word 2000 auf Anfrage nicht mehr die gezogenen Daten, sondern die aktuell in der Zwischenablage gespeicherten.

Die Zwischenspeicherung der Referenz in der Member-Variablen `DroppedData` ist daher nur für Testzwecke sinnvoll. Im konkreten Anwendungsfall müssen Sie die Daten spätestens im DragDrop-Ereignis abfragen und für die weitere Verarbeitung speichern.



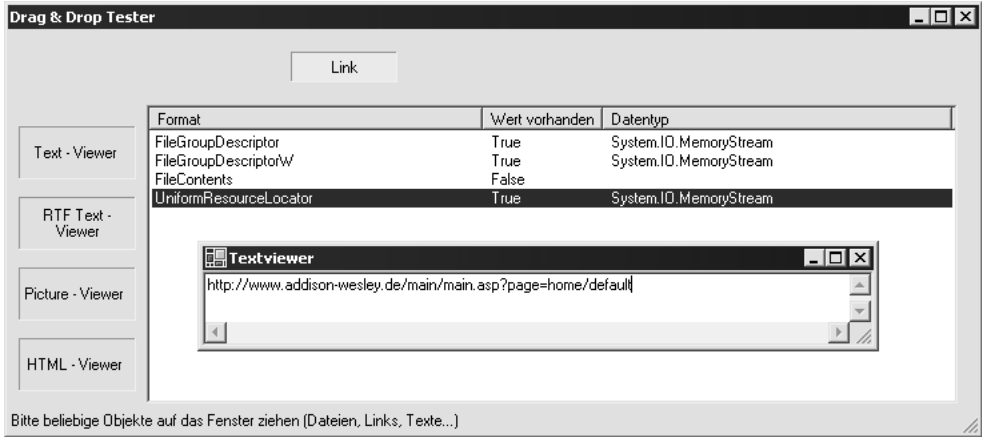


Abbildung 84: Drag&Drop von Links aus dem Internet Explorer

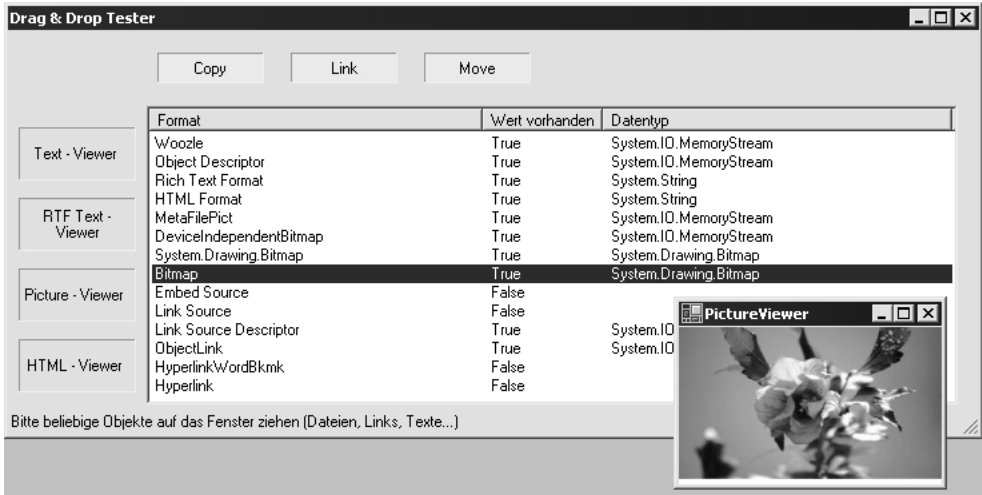


Abbildung 85: Drag&Drop von Bildern aus Word

## 110 Anzeigen von Daten aus der Zwischenablage

Nahezu identisch mit der Vorgehensweise bei Drag&Drop-Operationen ist die Bearbeitung von Daten, die in der Zwischenablage gespeichert sind. Auch sie werden auf Basis von OLE bereitgestellt. Das Analyseprogramm aus dem vorangegangenen Beispiel lässt sich leicht um eine Schaltfläche erweitern, die die verfügbaren Formate der in der Zwischenablage gespeicherten Daten anzeigt.

Für den Zugriff auf die Zwischenablage stellt die Klasse `Clipboard` die Methoden `GetDataObject` und `SetDataObject` zur Verfügung. Mit `GetDataObject` erhält man die Referenz der Daten analog zur `GetData`-Methode bei den `DragDropEventArgs`-Objekten. Listing 168 zeigt die Implementierung des Schaltflächen-Ereignisses zum Anzeigen der Clipboard-Daten. `SetData` ist dabei die in Listing 168 abgedruckte Methode. In Abbildung 86 sehen Sie ein Anwendungsbeispiel, bei dem der Text eines WordPad-Dokuments aus der Zwischenablage gelesen und dargestellt wurde.

```

Private Sub BTNClipboard_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles BTNClipboard.Click

    ' Daten der Zwischenablage abfragen
    SetData(Clipboard.GetDataObject())

End Sub

```

Listing 168: Abfrage der Daten aus der Zwischenablage und Anzeigen der Formate im Analyseprogramm

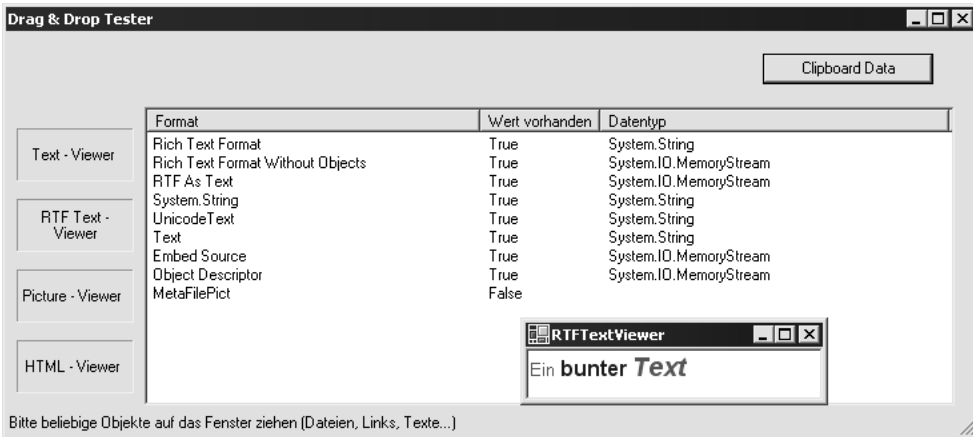


Abbildung 86: Anzeigen von Daten aus der Zwischenablage

## 111 Exportieren von Daten über die Zwischenablage

Auch die andere Richtung, nämlich das Kopieren von Daten in verschiedenen Formaten in die Zwischenablage, ist denkbar einfach zu programmieren. Der Trick besteht darin, der Methode `SetDataObject` der `Clipboard`-Klasse nicht wie im Beispiel der MSDN-Dokumentation ein beliebiges Objekt wie z.B. einen `String` zu übergeben, sondern eine Instanz der Klasse `DataObject`. Dieser Instanz lassen sich dann mit Hilfe der Methode `SetData` beliebige Formate hinzufügen (siehe Listing 169).

```

' Neue Instanz von DataObject mit einem String im Format Text
Dim ClipboardData As New DataObject("Text", "Hello World!")

' Bitmap hinzufügen
ClipboardData.SetData("Bitmap", New Bitmap(Application.StartupPath _
    & "\HibiskusKlein.jpg"))

' Zahl hinzufügen
ClipboardData.SetData("Number", 1234)

' Dateiliste hinzufügen
ClipboardData.SetData("FileDrop", New String() _
    {Application.StartupPath & "\HibiskusKlein.jpg"})

```

Listing 169: Bereitstellen von Daten in verschiedenen Formaten in der Zwischenablage

```
' Kopieren in die Zwischenablage
Clipboard.SetDataObject(ClipboardData, True)
```

Listing 169: Bereitstellen von Daten in verschiedenen Formaten in der Zwischenablage (Forts.)

Nach Abarbeitung des Codes in Listing 169 bietet Word bei Aufruf des Befehls INHALTE EINFÜGEN die in Abbildung 87 gezeigte Auswahl. Der Explorer erlaubt das Einfügen der angegebenen Datei. Eine vollständige Liste der verfügbaren Formate offenbart das Analyseprogramm (Abbildung 88). Zusätzlich zu den vier definierten Formaten werden weitere automatisch erzeugt. Entscheidend für diesen Automatismus ist der eingesetzte Formatname, nicht der tatsächliche Datentyp. Für das Format Bitmap wird automatisch das Format System.Drawing.Bitmap erzeugt, für Text automatisch System.String und Unicode Text und für FileDrop automatisch FileNameW und FileName.

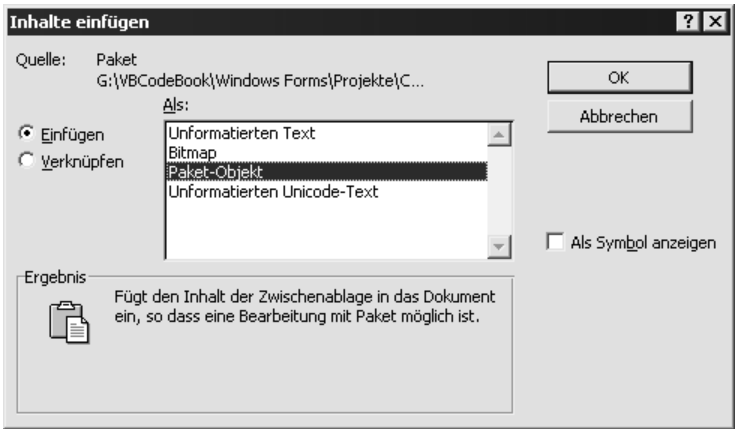


Abbildung 87: Word erkennt die bereitgestellten Datenformate

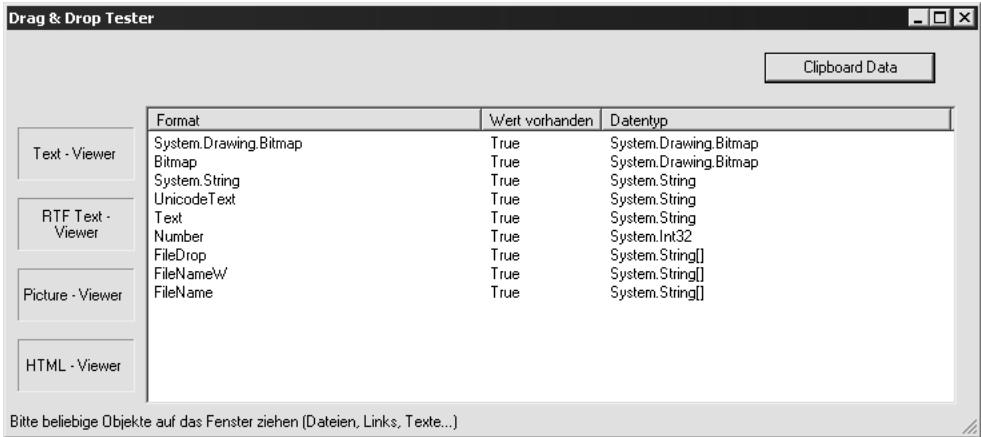


Abbildung 88: Angelegte und automatisch erzeugte Datenformate für die Zwischenablage

## 112 Exportieren von Daten über Drag&Drop

Wiederum mit den gleichen Mechanismen lassen sich Daten per Drag&Drop von .NET-Anwendungen auf Fenster anderer Anwendungen übertragen. Sie können auch hierbei die Daten in verschiedenen Formaten anbieten, so dass andere Programme flexibel reagieren können.

Eingeleitet wird der Drag&Drop-Vorgang durch Aufruf der Methode `DoDragDrop` des betreffenden Steuerelementes. Dieser Methode wird als Parameter eine Instanz von `DataObject` übergeben, die die Daten in den gewünschten Formaten enthalten muss. Die Schwierigkeit besteht hauptsächlich darin, `DoDragDrop` im passenden Moment aufzurufen. Die Steuerelemente `ListView` und `TreeView` besitzen zu diesem Zweck das Ereignis `ItemDrag`, das ausgelöst wird, wenn der Anwender beginnt, einen Eintrag mit der Maus zu ziehen. Abbildung 89 zeigt eine Beispielanwendung, bei der im oberen Teil ein `TreeView`-Steuerelement Dateien anzeigt, die via Drag&Drop auf andere Anwendungen gezogen werden können. In Listing 170 sehen Sie, wie der Vorgang eingeleitet wird.

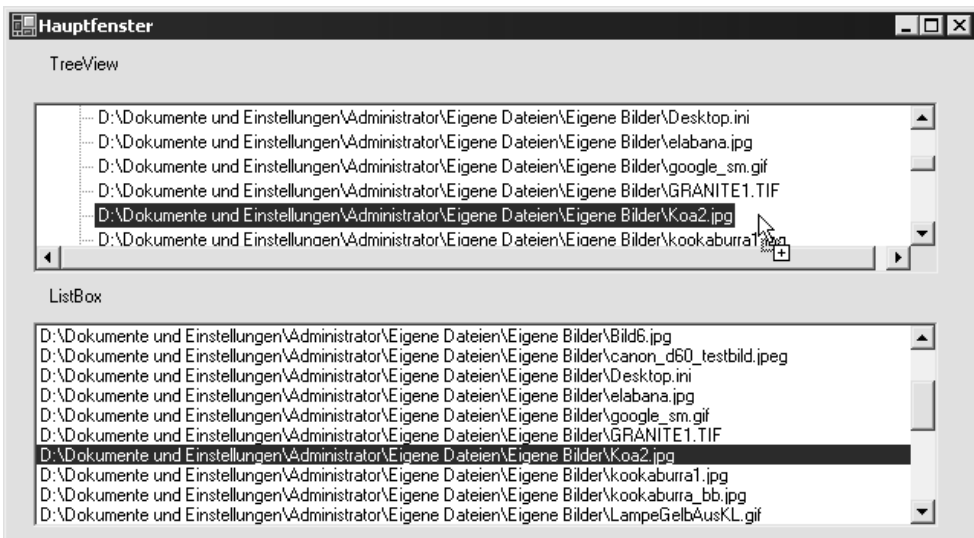


Abbildung 89: Drag&Drop aus einem TreeView und einer ListBox heraus

```
Private Sub TVLinks_ItemDrag(ByVal sender As System.Object, _
    ByVal e As System.Windows.Forms.ItemDragEventArgs) _
    Handles TVFiles.ItemDrag

    ' Knoten ermitteln
    Dim node As TreeNode = DirectCast(e.Item, TreeNode)

    ' Neue DataObject-Instanz mit Text-Format
    Dim data As New DataObject("Text", node.Text)

    ' FileDrop-Format hinzufügen
    Dim strArr() As String = New String() {node.Text}
    data.SetData("FileDrop", strArr)
```

Listing 170: Einleiten des Drag&Drop-Vorgangs bei einem TreeView-Steuerelement

---

```
' DragDrop-Vorgang beginnen
TVFiles.DoDragDrop(data, DragDropEffects.Copy Or _
    DragDropEffects.Link Or DragDropEffects.Move)
```

---

```
End Sub
```

---

*Listing 170: Einleiten des Drag&Drop-Vorgangs bei einem TreeView-Steuerelement (Forts.)*

Zu beachten ist, dass der Aufruf von `DoDragDrop` erst dann beendet ist, wenn der Vorgang abgeschlossen ist. D.h., erst wenn die Maustaste wieder losgelassen wurde, wird der nachfolgende Code ausgeführt.

Während des Drag&Drop-Vorgangs wird, wie schon beschrieben, bei der Quelle das Ereignis `GiveFeedback` ausgelöst (siehe Listing 171). Hier können Sie Aktionen (z.B. Änderung des Mauszeigers) vorsehen, um dem Anwender eine optische Rückmeldung zu geben.

---

```
Private Sub TVLinks_GiveFeedback(ByVal sender As System.Object, _
    ByVal e As System.Windows.Forms.GiveFeedbackEventArgs) _
    Handles TVFiles.GiveFeedback

    ' Zur Diagnose
    Debug.WriteLine(e.Effect)
End Sub
```

---

*Listing 171: GiveFeedback wird während des Ziehvorgangs aufgerufen*

Für Steuerelemente, die das `ItemDrag`-Ereignis nicht unterstützen, ist es etwas kniffliger, `DoDragDrop` im richtigen Moment aufzurufen. Üblicherweise überprüft man dazu im `MouseMove`-Ereignis die folgenden Bedingungen:

- ▶ Der Mauszeiger befindet sich über gültigen Daten, die gezogen werden können
- ▶ Die linke Maustaste (und nur diese) wurde gedrückt

In Listing 172 sehen Sie die Implementierung für die in Abbildung 89 gezeigte `ListBox`. Leider kollidieren Drag&Drop-Vorgänge oft mit Mauseaktionen innerhalb des Steuerelementes. Möchten Sie beispielsweise eine Mehrfachauswahl innerhalb der `ListBox` erlauben, dann lässt der Code es nicht zu, diese Auswahl mit der Maus aufzuziehen. Es gibt leider keine Möglichkeit zu unterscheiden, ob der Anwender einen bestimmten Eintrag per Drag&Drop ziehen möchte oder ob er die Auswahl erweitern will. Derartige Konfliktsituationen sollten Sie bei Ihren Anwendungen bedenken.

---

```
Private Sub LBFiles_MouseMove(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.MouseEventArgs) _
    Handles LBFiles.MouseMove

    ' Index des Eintrags an der Mausposition ermitteln
    Dim idx As Integer = LBFiles.IndexFromPoint(e.X, e.Y)

    ' Kein Drag&Drop, wenn dort kein Eintrag existiert
    If idx < 0 Then Exit Sub
```

---

*Listing 172: Drag&Drop-Vorgang für eine ListBox einleiten*

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

```

' Nur bei gedrückter linker Maustaste
If e.Button <> MouseButtons.Left Then Exit Sub

' Eintrag abrufen
Dim fn As String = LBFiles.Items.Item(idx).ToString()

' Datenobjekt anlegen und mit verschiedenen Datenformaten
' versorgen
Dim data As New DataObject("Text", fn)
Dim strArr() As String = New String() {fn}
data.SetData("FileDrop", strArr)

' DragDrop-Vorgang einleiten
LBFiles.DoDragDrop(data, DragDropEffects.Copy Or _
    DragDropEffects.Link Or DragDropEffects.Move)

End Sub

```

---

*Listing 172: Drag&Drop-Vorgang für eine ListBox einleiten (Forts.)*

## 113 Windows XP Darstellungsformate nutzen

Windows XP bietet neue elegante Stile zur Darstellung von Schaltflächen, CheckBoxen, Optionsfeldern usw., die Windows-Dialoge etwas gefälliger erscheinen lassen sollen als das triste Grau der alten Versionen (siehe Abbildung 90). Allerdings werden die Steuerelemente nicht automatisch in den neuen Darstellungsformaten angezeigt, wie in Abbildung 91 zu sehen ist. Zwar hat das Fenster die für XP üblichen abgerundeten Ecken, die Steuerelemente haben jedoch nach wie vor das alte Erscheinungsbild.

Zwei Schritte sind notwendig, um für eigene Fenster die Windows XP-Stile zu nutzen:

1. Aufruf von `Application.EnableVisualStyles`, bevor eine Fensterklasse instanziiert wird
2. Setzen der Eigenschaft `FlatStyle` auf den Wert `System`

Damit der Aufruf von `EnableVisualStyles` greift, empfiehlt es sich, diesen bereits in `Sub Main` zu platzieren, bevor überhaupt ein Fenster angelegt wird. Ein Aufruf innerhalb eines Konstruktors einer Fensterklasse kommt bereits zu spät. In Listing 173 sehen Sie eine typische Konstruktion zum Anzeigen eines Hauptfensters mit XP-Stilen. Unter älteren Windows-Versionen funktioniert der Aufruf auch, hat aber keinen sichtbaren Effekt. Es bietet sich daher an, ihn grundsätzlich für neue Programme vorzusehen, damit die Darstellungseigenschaften von XP ausgenutzt werden können.

---

```

Public Shared Sub Main()

    ' XP-Stile verwenden
    Application.EnableVisualStyles()

    ' Hauptfenster instanzieren
    Dim f As Form = New MainWindow

    ' Start der Nachrichtenschleife

```

---

*Listing 173: Verwendung von Windows XP-Stilen für Steuerelemente*

```
Application.Run(f)
```

```
End Sub
```

Listing 173: Verwendung von Windows XP-Stilen für Steuerelemente (Forts.)



Abbildung 90: Darstellung unter Windows 2000



Abbildung 91: Darstellung unter Windows XP ohne Aufruf von EnableVisualStyles

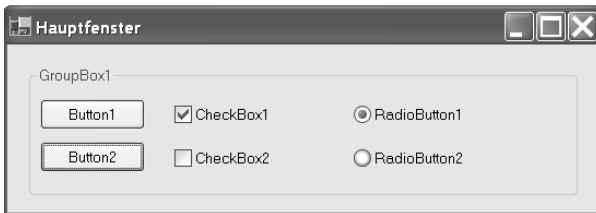


Abbildung 92: Darstellung unter Windows XP mit Aufruf von EnableVisualStyles

Haben Ihre Fenster viele Steuerelemente, die vielleicht auch noch in Containern wie GroupBoxen oder Panels geschachtelt sind, dann kann es recht mühsam sein, für alle betreffenden Steuerelemente die Eigenschaft `FlatStyle` zu setzen. Sie können dies jedoch auch mit Hilfe einer statischen Methode automatisieren (Listing 174). `SetSystemStyle` durchläuft alle Steuerelemente eines angegebenen Containers, prüft, ob sie die Eigenschaft `FlatStyle` unterstützen, und setzt sie gegebenenfalls. Ist ein betrachtetes Steuerelement selbst ein Container, dann werden auch alle untergeordneten Steuerelemente durch rekursiven Aufruf der Methode bearbeitet.

```
Public Shared Sub Main()

    ' XP-Stile verwenden
    Application.EnableVisualStyles()
```

Listing 174: Automatisiertes Setzen der Eigenschaft `FlatStyle` für alle Steuerelemente, die das unterstützen

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Propor-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

---

```

' Hauptfenster instanzieren
Dim f As Form = New MainWindow

' FlatStyle = System für alle Steuerelemente, die das unterstützen
SetSystemStyle(f)

' Start der Nachrichtenschleife
Application.Run(f)

End Sub

Public Shared Sub SetSystemStyle(ByVal ctrl As Control)

' Alle Steuerelemente dieses Containers bearbeiten
For Each c As Control In ctrl.Controls

' PropertyInfo-Objekt abrufen, sofern vorhanden
Dim pi As System.Reflection.PropertyInfo
pi = c.GetType().GetProperty("FlatStyle")

' Wenn vorhanden, dann FlatStyle auf System setzen
If Not pi Is Nothing Then pi.SetValue(c, _
    System.Windows.Forms.FlatStyle.System, Nothing)

' Wenn das Steuerelement selbst andere Steuerelemente
' beinhaltet, auch diese bearbeiten
If c.Controls.Count > 0 Then SetSystemStyle(c)
Next

End Sub

```

---

Listing 174: Automatisiertes Setzen der Eigenschaft FlatStyle für alle Steuerelemente, die das unterstützen (Forts.)

## 114 Eingabetaste in TextBox abfangen

Hat ein Dialogfenster eine OK-Schaltfläche, die als `AcceptButton` registriert ist, dann wird das Fenster geschlossen, wenn der Anwender die Eingabetaste auf der Tastatur betätigt. Für manche Benutzer ist dieses Verhalten überraschend und oft nicht erwünscht, z.B. dann, wenn sie gerade in einer `TextBox` etwas eingegeben haben und eigentlich nur diese Eingabe abschließen möchten. Oft wird daher der Wunsch geäußert, bei Betätigung der Eingabetaste (während eine `TextBox` den Fokus hat) einfach dem nächsten Steuerelement den Fokus zu geben und das Fenster nicht zu schließen.

Die Realisierung ist etwas kniffliger, als es zunächst den Anschein hat. Das Problem besteht darin, dass das entsprechende Tastatur-Ereignis weder von der `TextBox`, die den Fokus hat, noch von deren Container empfangen wird. Auch das Setzen der `KeyPreview`-Eigenschaft hilft nicht weiter. Die Windows-Botschaft wird schon vorher abgefangen und führt zum Schließen des Fensters.

In Rezept [7.12](#) ((Abfangen von Windows-Nachrichten)) werden mehrere Möglichkeiten vorgestellt, die Windows-Nachrichten vor ihrer Verarbeitung abzufangen und zu filtern. Für das beschriebene Problem lässt sich am einfachsten die `IMessageFilter`-Schnittstelle implementieren.



Eine zusätzliche Klasse ist hier nicht notwendig. Vielmehr lässt sich die Schnittstelle direkt in der Fensterklasse implementieren (siehe Listing 175). Das An- und Abschalten der Nachrichtenfilterung erfolgt in den überschriebenen Methoden `OnHandleCreated` bzw. `OnHandleDestroyed`.

Jede Nachricht, die Windows an das Fenster oder seine Steuerelemente sendet, führt bei eingeschalteter Filterung zum Aufruf der Methode `PreFilterMessage`. Hier müssen die Parameter ausgewertet werden. Zunächst wird überprüft, ob es sich um eine Tastatureingabe handelt (`WM_KEYDOWN`). Ist das der Fall, dann wird das Steuerelement ermittelt, das diese Nachricht empfangen hat. Nur, wenn es sich bei diesem Steuerelement um eine TextBox handelt, wird geprüft, welche Taste betätigt wurde. Der Tasten-Code wird im Parameter `WParam` übergeben.

Wurde die Eingabetaste betätigt, dann wird mittels `SendKeys.Send` die Betätigung der Tabulatortaste simuliert und die weitere Bearbeitung der aktuellen Nachricht abgebrochen. In allen anderen Fällen wird die Bearbeitung ohne Einflussnahme fortgesetzt.

```
Public Class Dialog
    Inherits System.Windows.Forms.Form
    Implements IMessageFilter

    ...

    Protected Overrides Sub OnHandleCreated( _
        ByVal e As System.EventArgs)

        ' Filter einschalten
        Application.AddMessageFilter(Me)

    End Sub

    Protected Overrides Sub OnHandleDestroyed( _
        ByVal e As System.EventArgs)

        ' Filter ausschalten
        Application.RemoveMessageFilter(Me)

    End Sub

    Public Function PreFilterMessage(ByRef m As _
        System.Windows.Forms.Message) As Boolean Implements _
        System.Windows.Forms.IMessageFilter.PreFilterMessage

        ' Message = WM_KEYDOWN?
        If m.Msg = &H100 Then

            ' Control-Referenz aus Handle ermitteln
            Dim c As Control = Control.FromChildHandle(m.HWnd)

            ' Ist das Steuerelement eine TextBox?
            If TypeOf c Is TextBox Then

                ' Wurde die Eingabetaste gedrückt?
```

Listing 175: Betätigung der Eingabetaste in einer TextBox abfangen und wandeln

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Propert-  
yGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schie-  
denes

---

```

    If m.WParam.ToInt32 = &HD Then

        ' Tab-Taste senden
        SendKeys.Send("{TAB}")

        ' Nachrichtenbearbeitung abbrechen
        Return True

    End If
End If
End If

' Nachrichtenbearbeitung fortsetzen
Return False

End Function

End Class

```

---

*Listing 175: Betätigung der Eingabetaste in einer TextBox abfangen und wandeln (Forts.)*

Beachten Sie bei der beschriebenen Vorgehensweise aber bitte, dass sich das Filtern der Botschaften nicht auf das Fenster beschränkt, in dem die Filtermethode definiert ist. Vielmehr werden alle Botschaften, die von dem Thread empfangen werden, der das Fenster erstellt hat, an die Methode weitergereicht. Das heißt, auch Nachrichten anderer Fenster werden bearbeitet. Möchten Sie beispielsweise aus dem Beispieldialog heraus einen weiteren modalen Dialog aufrufen, dessen Nachrichtenverarbeitung unbeeinflusst bleiben soll, dann müssen Sie vor Anzeige dieses Dialogs die Filterung abschalten.

Wenn mehrere nicht modale Fenster gleichzeitig offen sind, dann müssen Sie ggf. an Hand der Handles überprüfen, welchem Fenster das betreffende Steuerelement zuzuordnen ist, und Ihren Code darauf abstimmen.

## 115 Pfade so kürzen, dass sie in den verfügbaren Bereich passen

Immer, wenn Sie Verzeichnis- oder Dateipfade in einem Menü oder auf einem Steuerelement darstellen wollen, stellt sich das Problem, dass der Text sehr lang werden und unter Umständen nicht vollständig dargestellt werden kann. Weder in Menüs noch bei Steuerelementen wie Labels stehen Rollbalken zur Verfügung, mit denen sich der vollständige Text einsehen ließe. Abgesehen davon langt in vielen Fällen auch eine gekürzte Version. Betrachten Sie z.B. die Liste der zuletzt geöffneten Dateien im Datei-Menü von Microsoft Word oder Visual Studio. Auch hier werden aus Platzgründen die Dateinamen gekürzt, indem meist in der Mitte ein Teil durch eine Ellipse (...) ersetzt wird. Der gekürzte Pfad ist kein gültiger Pfad, der für weitere Aktionen benutzt werden kann, sondern dient lediglich der Visualisierung.

Der Algorithmus zum Kürzen der Pfade ist bereits in Windows implementiert und lässt sich über die API-Funktion `PathCompactPath` (siehe Listing 176) aufrufen. Die Benutzung dieser Methode ist allerdings ein wenig trickreich. Daher stellen wir Ihnen die Wrapper-Funktion `CompactPath` (Klasse `ApiVbNet`, Listing 177) zur Verfügung, die Sie vor einigen Fallen schützen soll.

---

```
Public Declare Auto Function PathCompactPath Lib "shlwapi.dll" _
    (ByVal hdc As IntPtr, ByVal pszPath As _
    System.Text.StringBuilder, ByVal dx As Integer) As Boolean
```

---

*Listing 176: PathCompactPath berechnet eine Kurzform eines Pfads*

Vorbereitend für den Aufruf von `PathCompactPath` wird mit dem zu kürzenden Pfad ein `StringBuilder`-Objekt initialisiert. Um die tatsächliche Ausgabegröße abschätzen zu können, benötigt die Methode Informationen über das Ausgabegerät und die verwendete Schriftart. Im Parameter `ctrl` muss daher die Referenz des Fensters bzw. Steuerelementes, auf das die Ausgabe erfolgt, übergeben werden. Für dieses `Control`-Objekt wird mittels `CreateGraphics` ein temporäres `Graphics`-Objekt angelegt, das später mit `Dispose` wieder freigegeben werden muss. Die API-Funktion arbeitet jedoch nicht mit dem `Graphics`-Objekt, sondern mit dem zugrunde liegenden Gerätekontext, dessen Handle mit `GetHdc` abgerufen und später mit `ReleaseHdc` wieder freigegeben wird. Auch für das im Parameter `usedFont` übergebene `Font`-Objekt wird ein Handle abgerufen.

Unter GDI wird eine Schriftart für die Ausgabe selektiert, indem bei einem Aufruf von `SelectObject` das Handle des Gerätekontextes sowie das Handle des `Font`-Objektes übergeben werden. Nachfolgende Ausgaben auf den Gerätekontext würden mit dieser Schriftart erfolgen. `PathCompactPath` nutzt die Informationen des ausgewählten Fonts für die internen Berechnungen und gibt den geänderten Pfad im `StringBuilder`-Objekt wieder zurück.

---

```
Public Shared Function CompactPath(ByVal path As String, _
    ByVal width As Integer, ByVal ctrl As Control, _
    ByVal usedFont As Font) As String

    ' StringBuilder für Textbearbeitung durch PathCompactPath
    Dim sb As New System.Text.StringBuilder(300)
    sb.Insert(0, path)

    Try
        ' Graphics-Objekt des Controls als Grundlage für die Berechnung
        Dim g As Graphics = ctrl.CreateGraphics()

        ' Gerätekontext abrufen
        Dim hdc As IntPtr = g.GetHdc()

        ' Handle des Font-Objektes
        Dim hfont As IntPtr = usedFont.ToHfont()

        ' Dieses Font-Objekt im Gerätekontext auswählen
        API.SelectObject(hdc, hfont)

        ' String von PathCompactPath berechnen lassen
        Dim res As Boolean = API.PathCompactPath(hdc, sb, width)

        ' Gerätekontext freigeben
        g.ReleaseHdc(hdc)

        ' Graphics-Objekt freigeben
```

---

*Listing 177: Wrapper-Funktion für die Nutzung der API-Funktion PathCompactPath*

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Propert-  
yGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

```

g.Dispose()

Catch ex As Exception
    Debug.WriteLine(ex.Message)
End Try

' Rückgabe des Strings
Return sb.ToString()

End Function

```

---

Listing 177: Wrapper-Funktion für die Nutzung der API-Funktion `PathCompactPath` (Forts.)

Ein kleines Beispiel soll die Anwendung der Methode näher erläutern. Auf einem Fenster werden sechs Labels unterschiedlicher Breite platziert. Der darzustellende Text ist die gekürzte Pfadangabe des absoluten Pfades des Ordners *Eigene Dateien*. Um die in Abbildung 93 gezeigte Ausgabe zu erzielen, wird die Methode `CreateLabels` (Listing 178) wie folgt aufgerufen:

---

```

Dim p As String
p = Environment.GetFolderPath(Environment.SpecialFolder.Personal)
CreateLabels(p)

```

---

Der Methode `CompactPath` wird dabei jeweils die Referenz des Labels sowie des zugeordneten Font-Objektes übergeben. Beachten Sie bei Label-Steuerelementen, dass der für die Schrift zur Verfügung stehende Bereich wegen der Ränder deutlich kleiner ist als der abfragbare Client-Bereich.

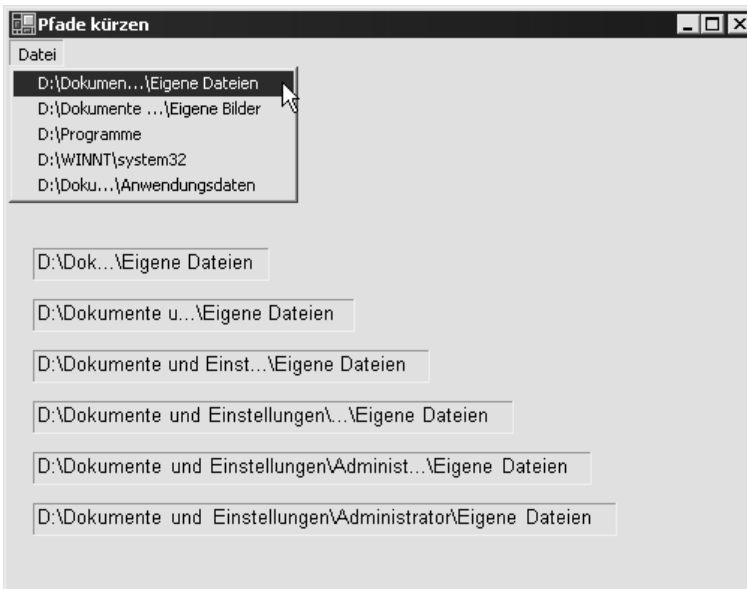


Abbildung 93: Pfade kürzen mit `PathCompactPath`

```

Public Sub CreateLabels(ByVal path As String)

    ' Hilfsvariablen
    Dim lbl As Label
    Dim y As Integer = 120
    Dim w As Integer = 150

    ' 6 Labels erstellen
    For i As Integer = 0 To 5

        ' Label-Instanz anlegen und Eigenschaften einstellen
        lbl = New Label
        lbl.BorderStyle = System.Windows.Forms.BorderStyle.Fixed3D
        lbl.Font = New System.Drawing.Font("Arial", 10.0!, _
            System.Drawing.FontStyle.Regular, _
            System.Drawing.GraphicsUnit.Point, CType(0, Byte))

        lbl.Location = New System.Drawing.Point(16, y)
        lbl.AutoSize = True

        ' Gekürzten Text ermitteln und auf Label ausgeben
        lbl.Text = ApiVNet.CompactPath(path, w, lbl, lbl.Font)

        ' Label der Controls-Auflistung hinzufügen und Variablen
        ' für Position und Breite anpassen
        Me.Controls.Add(lbl)
        w = w + 50
        y = y + 10 + 24

    Next

End Sub

```

Listing 178: Gekürzte Pfade auf Label-Steurelementen anzeigen

Die Ausgabe gekürzter Pfade in Menüs ist etwas aufwändiger. Da bei Auswahl eines Menüpunktes der zugehörige Pfad rekonstruierbar sein soll, muss er zusätzlich zum angezeigten Text abgespeichert werden. Leider bietet die Klasse `MenuItem` keine Tag-Eigenschaft oder Ähnliches, um zusätzliche Informationen an einen Menüpunkt anhängen zu können. Es bleibt somit als Lösung nur, eine neue Klasse von `MenuItem` abzuleiten. Diese Klasse (`FileMenuItem`, Listing 179) beinhaltet als Erweiterung eine `ReadOnly-Member-Variable`, die den vollständigen Pfad angibt, sowie eine zusätzliche Referenz-Variable (`Tag`), die wie bei Steuerelementen für andere Zwecke genutzt werden kann.

```

Public Class FileMenuItem
    Inherits MenuItem

    ' Der vollständige Pfad
    Public ReadOnly Path As String

```

Listing 179: Die Klasse `FileMenuItem` erweitert die Basisklasse um gekürzte Pfadangaben

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Proper-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

---

```

' Zusätzliche Referenz zur freien Verfügung
Public Tag As Object

Public Sub New(ByVal path As String, ByVal ctrl As Control, _
    ByVal width As Integer, ByVal tag As Object, _
    ByVal handler As System.EventHandler)

    ' Delegation an Konstruktor der Basisklasse
    MyBase.New(ApiVNet.CompactPath(path, width, ctrl, _
        SystemInformation.MenuFont), handler)

    ' Zusätzliche Member-Variablen setzen
    Me.Path = path
    Me.Tag = tag

End Sub

' ... weitere Konstruktoren mit Default-Parametern

End Class

```

---

*Listing 179: Die Klasse FileMenuItem erweitert die Basisklasse um gekürzte Pfadangaben (Forts.)*

Im Listing ist nur der Konstruktor abgedruckt, der alle Einstellmöglichkeiten als Parameter entgegennimmt. Konstruktoren mit gekürzter Parameterliste finden Sie im Projekt auf der CD.

Das Font-Objekt, das beim Aufruf von CompactPath übergeben werden muss, wird vom Betriebssystem vorgegeben und kann mit SystemInformation.MenuFont abgefragt werden. Im Konstruktor besteht leider keine Möglichkeit, eine Referenz des Control-Objektes zu erhalten, auf dem das Menü später dargestellt werden soll. Daher wird diese Referenz im Parameter ctrl übergeben. Mit Hilfe des folgenden Codes wird das Menü aufgebaut, wie es in Abbildung 93 zu sehen ist:

---

```

Dim p As String
p = Environment.GetFolderPath(Environment.SpecialFolder.Personal)

MNUFile.MenuItems.Add(New FileMenuItem(p, Me, 150, Nothing, _
    AddressOf MenuClick))

p = Environment.GetFolderPath( _
    Environment.SpecialFolder.MyPictures)
MNUFile.MenuItems.Add(New FileMenuItem(p, Me, 150, Nothing, _
    AddressOf MenuClick))

p = Environment.GetFolderPath( _
    Environment.SpecialFolder.ProgramFiles)
MNUFile.MenuItems.Add(New FileMenuItem(p, Me, 150, Nothing, _
    AddressOf MenuClick))

p = Environment.GetFolderPath(Environment.SpecialFolder.System)
MNUFile.MenuItems.Add(New FileMenuItem(p, Me, 150, Nothing, _
    AddressOf MenuClick))

```

---

---

```
p = Environment.GetFolderPath( _  
    Environment.SpecialFolder.ApplicationData)  
MNUFile.MenuItems.Add(New FileMenuItem(p, Me, 150, Nothing, _  
    AddressOf MenuClick))
```

---

Übergeben wird jeweils die Referenz des Forms-Objektes, das für die Generierung des Gerätekontextes herangezogen wird. Die Breite des Menütextes wird auf 150 Pixel festgelegt. Listing 180 zeigt, wie nach Auswahl eines Menüpunktes auf den vollständigen Pfad zurückgegriffen werden kann.

---

```
Private Sub MenuClick(ByVal sender As Object, ByVal e As EventArgs)  
  
    ' sender verweist auf ein FileMenuItem-Objekt  
    Dim mnu As FileMenuItem = DirectCast(sender, FileMenuItem)  
  
    ' Vollständigen Pfad ausgeben  
    MessageBox.Show(mnu.Path)  
  
End Sub
```

---

*Listing 180: Nach Auswahl eines Dateimenü-Punktes den vollständigen Pfad anzeigen*

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Propor-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes





# Windows Controls

War bereits die Flexibilität und Leistungsfähigkeit von ActiveX-Controls, die Sie mit VB6 erstellen konnten, gewaltig, dann wird sie von den Möglichkeiten, die Ihnen unter .NET zur Verfügung stehen, weit überholt und in den Schatten gestellt. Der geschickte Einsatz von Vererbungsmechanismen, oft auch Subclassing genannt, gibt Ihnen die Freiheit, vorhandene Steuerelemente mit zusätzlichen Funktionalitäten auszustatten, wie es früher nur mühsam mit C++ möglich war. Eigene Steuerelement-Kreationen können mit dem Designer aus vorhandenen Steuerelementen zusammengesetzt oder frei programmiert werden.

Anwender kommen mit Ihren Steuerelementen gewöhnlich zum ersten Mal in Berührung, wenn sie diese im Entwurfsmodus auf einem Fenster platzieren und deren Eigenschaften einstellen. Gerade hier ist es wichtig, dem Anwender möglichst einfach verständliche, aber leistungsfähige Hilfen an die Hand zu geben, um eine größtmögliche Akzeptanz zu erreichen. Auch in diesem Punkt ist .NET den Mitteln der ActiveX-Controls weit überlegen. Viele (leider nicht immer intuitiv erschließbare) Klassen und Attribute unterstützen Sie dabei, die Bedienung Ihrer Steuerelemente im Entwurfsmodus komfortabel zu gestalten. Eng verbunden hiermit ist auch das Eigenschaftsfenster (PropertyGrid), das im Kapitel 0 ((

Eigenschaftsfenster (PropertyGrid))) näher betrachtet wird.

In diesem Abschnitt finden Sie eine Reihe von Beispielen für die Programmierung eigener Steuerelemente sowie für die Erweiterung bestehender. Besonderheiten wie Transparenz, unregelmäßige Umrisse, Container-Controls werden hier ebenso erläutert wie Beispiele für die Entwicklung im Framework fehlender Steuerelemente wie FolderBrowser oder PanelGroupBox.

## 116 Ersatz für VB6-Control-Arrays

Sehr beliebt bei VB6-Programmierern waren so genannte Control-Arrays. Der VB6-Designer erlaubt es, mehreren Steuerelementen gleichen Typs denselben Namen zu geben und sie durch einen eindeutigen Index zu unterscheiden. Der Zugriff auf ein einzelnes Steuerelement erfolgt über den Namen und seinen Index, also genau wie bei einem Array-Zugriff. Alle Steuerelemente dieses Arrays benutzen gemeinsame Ereignis-Handler. Zur Unterscheidung, welches Control ein Ereignis ausgelöst hat, wird der Index als zusätzlicher Parameter übergeben.

VB.NET unterstützt diesen Mechanismus nicht mehr. Steuerelemente, die Sie mit Hilfe des Designers auf einem Fenster platzieren, müssen einen eindeutigen Namen haben. Mit Arrays kann der Designer nicht umgehen.

Dennoch stehen Ihnen viele Möglichkeiten zur Verfügung, ein ähnliches Verhalten zu implementieren. Möchten Sie lediglich, dass mehrere Steuerelemente die gleichen Event-Handler benutzen, so können Sie die `Handles`-Klausel eines bestehenden Handlers erweitern:

```
Private Sub CBScanner_CheckedChanged( _  
    ByVal sender As System.Object, _  
    ByVal e As System.EventArgs) _  
    Handles CBScanner.CheckedChanged, CBCamera.CheckedChanged, _  
    CBMouse.CheckedChanged
```

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung  
Win-  
dows  
Forms

Controls

Prop-  
ertyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

Diese Variante, bei der Sie die Ereignisse aller Objekte, für die die Methode aufgerufen werden soll, hinter dem Schlüsselwort `Handles` aufführen, lässt sich nur zur Entwurfszeit benutzen. Möchten Sie die Ereignis-Handler dynamisch zur Laufzeit hinzufügen, dann müssen Sie das über `AddHandler` realisieren (s.u.).

Soll der Zugriff auf ein Steuerelement über ein Array erfolgen, dann definieren Sie es selbst und füllen es mit den Referenzen der gewünschten Steuerelemente. In Listing 181 sehen Sie ein Beispiel, bei dem die Referenzen von vier `CheckBox`s in einem zusätzlichen Array (`EquipmentSelections`) gespeichert werden.

Achtung

Ändern Sie nicht den vom Designer angelegten Code in `InitializeComponent`! Bei Zuwiderhandlungen versagt der Designer seinen Dienst und Sie können das Formular nicht weiter bearbeiten ;-).

Der Designer kann mit Arrays von Steuerelementen nichts anfangen und wird sie im günstigsten Fall aus `InitializeComponent` wieder entfernen.

---

```
' Array als Member-Variable der Form-Klasse
Protected EquipmentSelections() As CheckBox

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

    ' Array anlegen und vorhandene CheckBoxen zuweisen
    EquipmentSelections = New CheckBox() _
        {CBPrinter, CBScanner, CBCamera, CBMouse}

    ' Array durchlaufen und jede CheckBox an gemeinsamen
    ' Click-Event-Handler binden
    For Each cb As CheckBox In EquipmentSelections
        AddHandler cb.Click, _
            New EventHandler(AddressOf SelectionChanged)
    Next

End Sub
```

---

*Listing 181: Definieren eines Arrays für vier CheckBoxen*

Das Array wird im `Load`-Ereignis angelegt. Im Beispiel werden die `CheckBox`-Referenzen direkt bei der Array-Definition zugewiesen. Anschließend wird das Array in einer `For Each`-Schleife durchlaufen und die Methode `SelectionChanged` an das `Click`-Ereignis jeder `CheckBox` gebunden.

Diese Methode könnte z.B. so wie in Listing 182 aussehen. In einer `For Each`-Schleife wird das Array durchlaufen und für jede `CheckBox`, deren Häkchen gesetzt ist, ein Eintrag bestehend aus `Name` und `Text` in der `ListBox` vorgenommen.

---

```
Private Sub SelectionChanged(ByVal sender As Object, _
    ByVal e As System.EventArgs)
```

---

*Listing 182: Mit `AddHandler` gebundene Ereignis-Methode*

```

' Liste löschen
LBSelection.Items.Clear()

' Array durchlaufen und Name und Text jeder ausgewählten CheckBox
' in die Liste aufnehmen
For Each cb As CheckBox In EquipmentSelections
    If cb.Checked Then LBSelection.Items.Add( _
        cb.Name & " / " & cb.Text)
Next

End Sub

```

Listing 182: Mit AddHandler gebundene Ereignis-Methode (Forts.)

Wenn Sie innerhalb des Handlers wissen möchten, welches Steuerelement das Ereignis ausgelöst hat, benutzen Sie den Parameter `sender`. Über einen `TypeCast` (mit `DirectCast` oder `CType`) können Sie direkt auf dessen Eigenschaften zugreifen:

```

Debug.WriteLine("Auslösende CheckBox: " & _
    DirectCast(sender, CheckBox).Name)

```

Ausgabe z.B.:

```
Auslösende CheckBox: CBCamera
```

Letztlich steht Ihnen die gesamte Bandbreite der Objektorientierten Programmierung zur Verfügung. Statt eines Arrays können Sie die Steuerelement-Referenzen auch in einer Liste speichern. Eine Hashtable bietet Ihnen z.B. zusätzlich die Möglichkeit, jedes Steuerelement über einen Schlüssel (beispielsweise seinen Namen) anzusprechen.

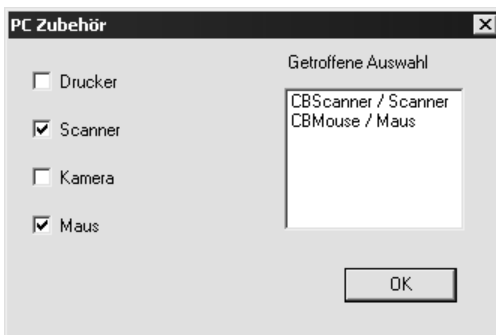


Abbildung 94: Gemeinsamer Ereignis-Handler für vier CheckBoxen

Wenn Sie alle gleichartigen Steuerelemente, die in einem Container (Form, Panel, GroupBox usw.) liegen, gleichermaßen bearbeiten, dann können Sie das `ControlAdded`-Event auswerten. Für jedes Steuerelement, das dem betreffenden Container hinzugefügt wird, löst dieser das Ereignis aus. Im Beispiel in Listing 183 wird überprüft, ob es sich bei dem Steuerelement um eine CheckBox handelt. Wenn ja, dann werden für diese CheckBox die Vorder- und Hintergrundfarben eingestellt.

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung  
Win-  
dows  
Forms

Controls

Prop-  
ertyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

```

Private Sub MainWindow_ControlAdded(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.ControlEventArgs) _
    Handles MyBase.ControlAdded

    ' Ist das neue Steuerelement eine CheckBox?
    If TypeOf e.Control Is CheckBox Then
        ' Ja, Eigenschaften einstellen
        Dim cb As CheckBox = DirectCast(e.Control, CheckBox)
        cb.BackColor = Color.Black
        cb.ForeColor = Color.Yellow
    End If

End Sub

```

---

*Listing 183: Für jedes neue Steuerelement wird ControlAdded aufgerufen*

In diesem Ereignis könnten Sie z.B. auch die Steuerelement-Referenzen in eine Liste mit aufnehmen. So erhalten Sie eine generische Lösung, bei der Sie die Steuerelemente nicht einzeln auflisten müssen. Denn bei nachträglichen Änderungen im Designer wird schnell einmal vergessen, ein Array wie das aus Listing 181 auch entsprechend anzupassen.

## 117 ListBox-Items selber zeichnen

Reicht Ihnen die Darstellung einer gewöhnlichen ListBox nicht aus, weil Sie beispielsweise Symbole oder unterschiedliche Schriftarten anzeigen möchten, dann können Sie das Zeichnen der Listenelemente auch selbst in die Hand nehmen. Die ListBox gehört zu den wenigen Steuerelementen, die durch geeignete Eigenschaften und Ereignisse eine benutzerdefinierte Anzeige wesentlich vereinfachen.

Über die Eigenschaft `DrawMode` steuern Sie das Verhalten der ListBox. Drei Werte sind erlaubt:

### 1. Normal

Die ListBox übernimmt das Zeichnen selbst. Alle Einträge haben die gleiche Höhe und benutzen den gleichen Font. Die Texte werden durch Aufruf der Methode `ToString` beim Hinzufügen neuer Elemente ermittelt.

### 2. OwnerDrawFixed

Alle Einträge haben die in `ItemHeight` eingestellte Höhe. Zum Zeichnen löst die ListBox für jedes Listenelement das Ereignis `DrawItem` aus, in dem Sie selbst die Grafikausgabe vornehmen können.

### 3. OwnerDrawVariable

Jeder Eintrag kann individuelle Abmessungen besitzen. Bevor die Liste gezeichnet wird, löst die ListBox für jedes Element das Ereignis `MeasureItem` aus, in welchem Sie festlegen müssen, wie viel Platz der jeweilige Eintrag benötigt. Das Zeichnen der Elemente erfolgt wie bei `OwnerDrawFixed` über das Ereignis `DrawItem`.

Die Vorgehensweise soll an einem kleinen Beispiel erläutert werden. In einem Anwendungsprogramm sollen in einer ListBox ausgewählte Schriftarten dargestellt werden. Jede Schriftart soll mit ihrem eigenen Font angezeigt werden (siehe Abbildung 95). Die Liste kann zur Laufzeit durch Hinzufügen neuer Elemente oder Entfernen vorhandener Elemente verändert werden.



Abbildung 95: Ownerdrawn ListBox mit unterschiedlichen Schriftarten

Im Load-Ereignis des Fensters werden zunächst einige Font-Objekte angelegt und der ListBox hinzugefügt (Listing 184). Es genügt, die Referenzen der Font-Instanzen der Add-Methode zu übergeben, da die ListBox die anzuzeigenden Texte nicht selbst ermitteln muss, sondern hierzu DrawItem aufruft (s.u.).

```
Private Sub Form1_Load(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles MyBase.Load

    ' Zu Beginn einige Schrifarten in die Liste aufnehmen
    LBFonts.Items.Add(New Font("Arial", 10))
    LBFonts.Items.Add(New Font("Arial", 20))
    LBFonts.Items.Add(New Font("Arial", 30))
    LBFonts.Items.Add(New Font("Times", 10))
    LBFonts.Items.Add(New Font("Times", 20))
    LBFonts.Items.Add(New Font("Times", 30))

End Sub
```

Listing 184: Vorbelegen der ListBox mit einigen Font-Objekten

Damit die ListBox den Platz für die Elemente einplanen kann, fragt sie deren Größe über das MeasureItem-Ereignis ab. Listing 185 zeigt die Beispielimplementierung. Übergeben wird der Parameter e vom Typ MeasureItemEventArgs. Die Eigenschaft e.Index gibt an, für welches Item die Größe berechnet werden soll, e.Graphics liefert die Referenz eines Graphics-Objektes, das für die Berechnung herangezogen werden kann.

Über den Index erhält man die Referenz des anzuzeigenden Objektes, die via TypeCast einer lokalen Variablen vom Typ Font zugewiesen wird. Der Text wird zusammengesetzt aus dem Namen

und der Größe der Schriftart. Zur Berechnung der Größe wird die Methode `MeasureString` aufgerufen. Sie liefert die Abmessungen in Form einer `SizeF`-Struktur. Durch Setzen der Eigenschaften `e.ItemHeight` und `e.ItemWidth` wird der `ListBox` die benötigte Größe mitgeteilt.

---

```
Private Sub LBFonts_MeasureItem(ByVal sender As System.Object, _
    ByVal e As System.Windows.Forms.MeasureItemEventArgs) _
    Handles LBFonts.MeasureItem

    ' Anzuzeigendes Objekt
    Dim font As Font = DirectCast(LBFonts.Items(e.Index), Font)

    ' Darstellungsgröße berechnen
    Dim sz As SizeF = e.Graphics.MeasureString(font.Name & " " & _
        font.SizeInPoints, font)

    ' Information an ListBox weitergeben
    e.ItemHeight = CInt(sz.Height)
    e.ItemWidth = CInt(sz.Width)

End Sub
```

---

*Listing 185: Ermitteln der Größe eines ListBox-Eintrags im Ereignis MeasureItem*

Ganz ähnlich erfolgt auch das eigentliche Zeichnen der Elemente. Hierzu wird für jedes Listenelement der in Listing 186 gezeigte Ereignis-Handler `DrawItem` aufgerufen. Hier liefert ein Parameter vom Typ `DrawItemEventArgs` die benötigten Informationen. Die wichtigsten Eigenschaften und Methoden sind in Tabelle 20 aufgeführt.

Wie in `MeasureItem` wird auch hier zunächst das `Font`-Objekt separiert. Der Aufruf von `DrawBackground` erspart das eigenhändige Zeichnen des Hintergrundes. Danach muss der Text mittels `DrawString` ausgegeben werden.

Üblicherweise müsste man an dieser Stelle entscheiden, mit welcher Vordergrundfarbe gezeichnet werden soll. Ist das zu zeichnende Element selektiert, dann wird es in der Regel einen dunklen Hintergrund besitzen und die eingestellte Vordergrundfarbe (meistens schwarz) bietet evtl. zu wenig Kontrast. Aber auch hier wurde mitgedacht J: Die Eigenschaft `e.ForeColor` berücksichtigt bereits, dass bei selektierten Elementen eine Vordergrundfarbe benötigt wird. So können Sie über diese Eigenschaft direkt das notwendige `Brush`-Objekt erzeugen. Abschließend vergessen Sie bitte nicht, das angelegte `Brush`-Objekt wieder mit `Dispose` freizugeben.

---

```
Private Sub LBFonts_DrawItem(ByVal sender As System.Object, _
    ByVal e As System.Windows.Forms.DrawItemEventArgs) _
    Handles LBFonts.DrawItem

    ' Anzuzeigendes Objekt
    Dim font As Font = DirectCast(LBFonts.Items(e.Index), Font)

    ' Brush für Darstellung der Schrift
    Dim fbr As Brush = New SolidBrush(e.ForeColor)
```

---

*Listing 186: Zeichnen eines Listenelementes in DrawItem*

```
' Hintergrund zeichnen
e.DrawBackground()

' Textausgabe
e.Graphics.DrawString(font.Name & " " & font.SizeInPoints, _
    font, fbr, e.Bounds.X, e.Bounds.Y)

' Ressourcen freigeben
fbr.Dispose()

End Sub
```

Listing 186: Zeichnen eines Listenelementes in DrawItem (Forts.)

| Eigenschaft / Methode | Bedeutung                                                      |
|-----------------------|----------------------------------------------------------------|
| Index                 | Index des zu zeichnenden Listenelementes                       |
| State                 | Bitkombination, die den Zustand angibt (z.B. Selected)         |
| Bounds                | Rechteck, das den Bereich für die Zeichnung vorgibt            |
| Graphics              | Referenz des Graphics-Objektes, auf das gezeichnet werden soll |
| Font                  | Eingestellte Schriftart                                        |
| ForeColor             | Farbe, in der die Schrift gezeichnet werden soll               |
| BackColor             | Farbe, in der der Hintergrund gezeichnet werden soll           |
| DrawBackground        | Methode, die automatisch den Hintergrund zeichnet              |

Tabelle 20: Wichtige Eigenschaften und Methoden der Klasse DrawItemEventArgs

Nach dem gezeigten Verfahren können Sie Ihre ListBoxen nach eigenen Vorstellungen gestalten. Das Zeichnen des Textes mit verschiedenen Schriftarten ist nur ein Anwendungsfall. Sie können beliebige Graphiken vorsehen und auch Bilder einfügen.

Bei komplexeren Konstruktionen empfiehlt es sich allerdings, eine neue Steuerelementklasse von ListBox abzuleiten und die Zeichenoperationen in dieser Klasse zu kapseln. In dieser Klasse können Sie zusätzliche Eigenschaften bereitstellen, mit denen der Anwender das Aussehen der ListBox parametrieren kann.

## 118 Mehrspaltige DropDown-Liste (ComboBox)

Besonders von Access-Programmierern wird oft die Frage gestellt, wie man die Anzeigen in einer ComboBox in mehrere Spalten aufteilen kann. Während Access ein eigenes Steuerelement dafür bereithält, unterstützt die ComboBox des .NET Frameworks diese Möglichkeit nicht auf direktem Weg. Sie können aber, genau wie im vorangegangenen Beispiel, auch bei einer ComboBox die Elemente durch eigenen Code selber zeichnen. Auch hierzu eine Demonstration zur Vorgehensweise:

Im Beispiel (Abbildung 96) sollen Kundendaten als DropDown-Auswahlliste angezeigt werden. Der Anwender kann nur aus der Liste wählen, aber keine neuen Texte eingeben. Um das Beispiel auf das Wesentliche zu reduzieren, wird die Spaltenbreite fest eingestellt. Listing 187 zeigt die zugrunde liegende Datenklasse, Listing 188 das Hinzufügen einiger Instanzen im Load-Ereignis.

|                           |
|---------------------------|
| Basics                    |
| Datum/<br>Zeit            |
| Anwen-<br>dungen          |
| Zeich-<br>nen             |
| Bild-<br>bearbei-<br>tung |
| Win-<br>dows<br>Forms     |
| Controls                  |
| Propert-<br>yGrid         |
| Datei-<br>system          |
| Netz-<br>werk             |
| System                    |
| Daten-<br>banken          |
| XML                       |
| Wissen-<br>schaft         |
| Ver-<br>schiede-<br>nes   |

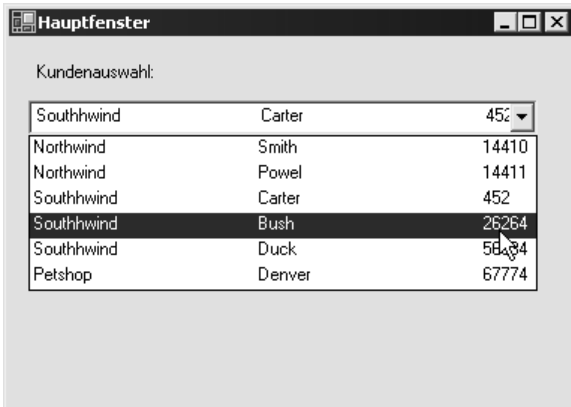


Abbildung 96: Darstellen der Informationen in einer mehrspaltigen ComboBox

---

```
Public Class Customer

    ' Datenfelder
    Public Company As String
    Public Name As String
    Public Phone As String

    ' Konstruktor
    Public Sub New(ByVal company As String, ByVal name As String, _
        ByVal phone As String)

        Me.Company = company
        Me.Name = name
        Me.Phone = phone
    End Sub

End Class
```

---

Listing 187: Klasse für die anzuzeigenden Informationen

---

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

    ' Einige Objekte der ComboBox hinzufügen
    CBCustomers.Items.Add(New Customer("Northwind", "Smith", "14410"))
    CBCustomers.Items.Add(New Customer("Northwind", "Powel", "14411"))
    CBCustomers.Items.Add(New Customer("Southwind", "Carter", "452"))
    CBCustomers.Items.Add(New Customer("Southwind", "Bush", "26264"))
    CBCustomers.Items.Add(New Customer("Southwind", "Duck", "56634"))
    CBCustomers.Items.Add(New Customer("Petshop", "Denver", "67774"))

End Sub
```

---

Listing 188: Füllen der Liste mit einigen Instanzen



Da die Abmessungen aller Listenelemente gleich sein soll, wird die Eigenschaft `DrawMode` auf `OwnerDrawFixed` gesetzt. Ein Wert von `DropDownList` für die Eigenschaft `DropDownStyle` beschränkt die Bedienung der Liste auf eine einfache Auswahl. In Listing 189 sehen Sie die Implementierung des `DrawItem`-Ereignis-Handlers, der für das Zeichnen zuständig ist.

Wie im Beispiel der `ListBox` werden zunächst die Daten ermittelt, der Hintergrund gezeichnet und die Farbe für die Textanzeige gewählt. Danach werden die einzelnen Text-Fragmente mit `DrawString` ausgegeben. Beachten Sie hier den vorletzten Parameter. Er gibt die X-Koordinate für den Text an und legt so die Position der jeweiligen Spalte fest.

```
Private Sub CBCustomers_DrawItem(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.DrawItemEventArgs) _
    Handles CBCustomers.DrawItem

    If e.Index < 0 Then Exit Sub

    ' Customer-Objekt ermitteln
    Dim cust As Customer = DirectCast(CBCustomers.Items(e.Index), _
        Customer)

    ' Brush zum Zeichnen des Textes
    Dim br As New SolidBrush(e.ForeColor)

    ' Hintergrund zeichnen
    e.DrawBackground()

    ' Text für Eigenschaft Company ausgeben
    e.Graphics.DrawString(cust.Company, CBCustomers.Font, br, _
        e.Bounds.X, e.Bounds.Y)

    ' Text für Eigenschaft Name ausgeben
    e.Graphics.DrawString(cust.Name, CBCustomers.Font, br, _
        e.Bounds.X + 150, e.Bounds.Y)

    ' Text für Eigenschaft Phone ausgeben
    e.Graphics.DrawString(cust.Phone, CBCustomers.Font, br, _
        e.Bounds.X + 300, e.Bounds.Y)

    ' Ressourcen freigeben
    br.Dispose()
End Sub
```

Listing 189: Zeichnen der Texte in mehreren Spalten

Dieses kleine Beispiel soll Ihnen lediglich demonstrieren, wie Sie auch in einer `ComboBox` die angezeigten Texte formatieren können. Für eine gebrauchsfertige Implementierung empfiehlt es sich auch hier, eine eigene `ComboBox`-Klasse abzuleiten und z.B. Eigenschaften für die Anzahl der Spalten und deren Breiten vorzusehen. Bei allen Möglichkeiten, die Ihnen `ListBox` und `ComboBox` anbieten, sollten Sie jedoch nicht die anderen Steuerelemente aus dem Blick verlieren. Statt einer komplizierten Anpassung einer List- oder `ComboBox` ist vielleicht der Einsatz einer `List-View` eleganter und passender.

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung  
Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schie-  
des

## 119 Basisklassen für selbst definierte Steuerelemente

Wenn Sie ein neues Steuerelement programmieren möchten, dann müssen Sie sich entscheiden, von welcher Basisklasse Sie es ableiten möchten. Abhängig davon, welches Ziel Sie erreichen wollen, haben Sie die Wahl zwischen vielen Klassen, die von `System.ComponentModel.Component` abgeleitet sind, sowie der Klasse `Component` selbst. Abbildung 97 zeigt nur einen kleinen Teil der Klassenhierarchie der Steuerelementklassen, die Sie als Basisklasse einsetzen können.

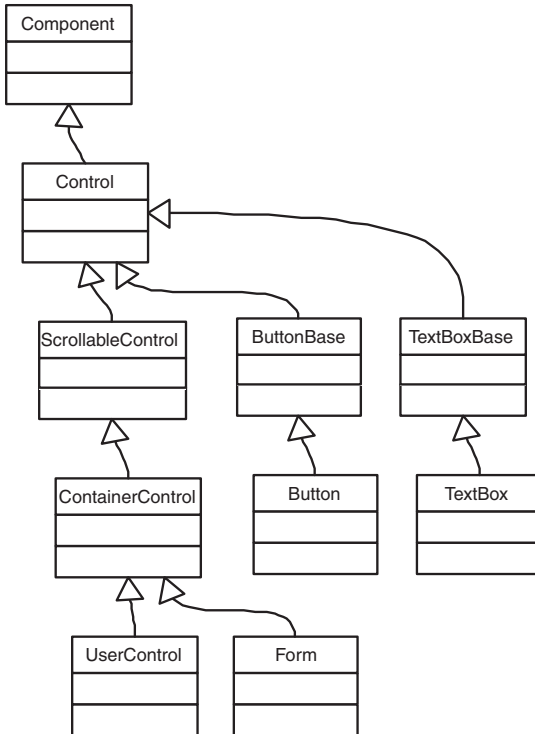


Abbildung 97: Ausschnitt aus der Klassenhierarchie der Steuerelementklassen

`UserControl` ist die Standardklasse, die Visual Studio einsetzt, wenn Sie in einem Projekt ein neues Control mit PROJEKT / HINZUFÜGEN / BENUTZERSTEUERELEMENT HINZUFÜGEN anlegen. Sie ist die einzige Klasse, für die der Designer Ihnen die Möglichkeit bietet, visuell ein neues Steuerelement aus bestehenden Steuerelementen zusammenzusetzen. Im Entwurfsmodus können Sie beliebige Controls aus der Toolbox auf das Fenster des `UserControls` ziehen und anordnen. Viele Basisfunktionalitäten wie z.B. Scroll-Balken oder oft benötigte Eigenschaften sind bereits vorbereitet.

Benötigen Sie lediglich rudimentäre Basisfunktionen und wollen den Ballast spezialisierter Controls nicht mitführen, dann kommt eine Ableitung von den Klassen `Control`, `ScrollableControl` oder `ContainerControl` in Frage. Sie erhalten hierbei allerdings keine visuelle Unterstützung der Entwicklungsumgebung und müssen das Steuerelement programmatisch erstellen.

Gibt es bereits ein Steuerelement, das weitestgehend Ihre Wünsche erfüllt, das Sie aber in seinem Funktionsumfang erweitern möchten, dann können Sie auch Ihre `Control`-Klasse direkt von diesem ableiten. So können Sie beispielsweise eine `TextBox` erweitern, indem Sie die Klasse `TextBox` als Basisklasse für Ihre Steuerelementklasse verwenden. In der MSDN-Dokumentation finden Sie

neben den öffentlichen Mitgliedern einer Klasse auch die geschützten, die Sie in Ihrer abgeleiteten Klasse ebenfalls verwenden können bzw. müssen.

Soll das Steuerelement lediglich für den einfacheren Umgang mit einer Klasse dienen, für die der Anwender bereits zur Entwurfszeit eine Instanz vorsehen und deren Eigenschaften ändern kann, dann können Sie als Basisklasse die Klasse `Component` einsetzen. Derartige Steuerelemente besitzen keine visuelle Oberfläche und sind zur Laufzeit nicht sichtbar. Beispiele hierfür sind die Klassen `System.Windows.Forms.Timer`, `FileSystemWatcher`, `Process` etc., die Sie im Abschnitt KOMPONENTEN der Toolbox finden.

Hinweis

Der Designer der Entwicklungsumgebung muss, wenn Sie ein Steuerelement auf einem Fenster oder einem anderen Steuerelement einsetzen, die betreffende Klasse instanzieren können. Das bedeutet, dass die Klasse nicht abstrakt (`MustInherit`) sein darf und dass ein öffentlicher Standard-Konstruktor (ohne Parameter) existieren muss. Treten bei der Instanzierung Fehler auf (z.B. weil Dateipfade nicht stimmen, Objekte nicht existieren o.Ä.), dann verweigert der Designer das Erstellen des Controls. Wenn Sie die Ursache für ein solches Verhalten nicht aus dem Quelltext ermitteln können, dann sollten Sie mit dem im Anhang beschriebenen Verfahren das Steuerelement zur Entwurfszeit debuggen.

## 120 ComboBox mit Auto-Vervollständigen-Funktion

Eine `ComboBox`, deren Eigenschaft `DropDownStyle` nicht auf `DropDownList` gestellt wird, bietet keine automatische Auswahl eines der Listeneinträge während der Eingabe. Diese lässt sich aber leicht nachrüsten.

Die Idee ist, bei jedem Tastendruck zu überprüfen, ob in der `Items`-Auflistung der `ComboBox` ein Text enthalten ist, der mit dem eingegebenen Text beginnt. Ist das der Fall, dann wird der als erster gefundene Text ausgewählt. Damit der Anwender sich nicht bevormundet fühlt, wird der automatisch erzeugte Teil des Textes markiert. Tippt der Anwender weiter, dann wird der markierte Bereich gelöscht und die Suche beginnt von neuem.

Zur Lösung der Aufgabe wird eine neue Klasse (`AutocompleteComboBox`) von der Klasse `ComboBox` abgeleitet und der beschriebene Algorithmus im `KeyUp`-Ereignis implementiert (Listing 190). Die Methode `FindString` sucht nach dem ersten Text der `Items`-Auflistung, der mit dem angegebenen Text beginnt, und gibt dessen Index zurück. Wird kein Text gefunden, ist der Rückgabewert negativ. Abbildung 98 zeigt ein Beispiel, bei dem die Namen verschiedener Banken in der `Items`-Liste der `ComboBox` hinterlegt wurden.

```
Public Class AutocompleteComboBox
    Inherits ComboBox

    Private Sub AutocompleteComboBox_KeyUp(ByVal sender As Object, _
        ByVal e As System.Windows.Forms.KeyEventArgs) _
        Handles MyBase.KeyUp

        Dim typedText As String = Me.Text
        Dim index As Integer

        ' Behandlung der Pfeiltasten etc. zulassen
```

Listing 190: Erweiterung der `ComboBox`-Klasse mit automatischer Textvervollständigung

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Propor-  
tGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

```

Select Case e.KeyCode
    Case Keys.Left, Keys.Right, Keys.Up, Keys.Down, _
        Keys.Delete, Keys.Back
    Return
End Select

' Text in der Items-Liste suchen
index = Me.FindString(typedText)

' Text vorhanden?
If index >= 0 Then
    ' Ja, Item auswählen und hinzugefügten Teil markieren
    Me.SelectedIndex = index
    Me.Select(typedText.Length, Me.Text.Length)
End If

End Sub

End Class

```

*Listing 190: Erweiterung der ComboBox-Klasse mit automatischer Textvervollständigung (Forts.)*



*Abbildung 98: Auto-Vervollständigen des Textes in einer ComboBox*

## 121 Benutzersteuerelement als Container für andere Steuerelemente

Ein Benutzersteuerelement (`UserControl`) ist zunächst kein Container-Control, auf das Sie im Designer andere Steuerelemente platzieren können, so, wie Sie es von Steuerelementen wie der `GroupBox` oder dem `Panel` kennen. Auch wenn es grundsätzlich von der Klasse `ContainerControl` abgeleitet ist und auch zur Laufzeit andere Steuerelemente aufnehmen kann, lässt der Designer die Ablage anderer Steuerelemente nicht zu. Um dennoch zur Entwurfszeit zu ermöglichen, dass andere Steuerelemente auf dem Benutzersteuerelement liegen und auf ihm verankert werden können, ist ein zusätzlicher Handgriff notwendig: Die Klasse muss mit dem Attribut `DesignerAttribute` gekennzeichnet werden. Der Aufruf hierfür sieht in etwa so aus:

```

Imports System.Windows.Forms
Imports System.ComponentModel
Imports System.Windows.Forms.Design

<Designer(GetType(ParentControlDesigner))> _
Public Class NameDesSteuerelementes
    Inherits System.Windows.Forms.UserControl

```

Nach der Zuweisung dieses Attributs können Sie das Steuerelement für beliebige Verschachtelungen im Designer verwenden.

Alternativ können Sie das Benutzersteuerelement statt von der Klasse `UserControl` von der Klasse `ContainerControl` ableiten:

```
Public Class UserControl1
    Inherits System.Windows.Forms.ContainerControl
    ...
End Class
```

Auch so erreichen Sie, dass ein Anwender Ihres Steuerelementes im Designer dieses als Container für andere Steuerelemente nutzen kann. Allerdings verlieren Sie dann die Option, das Steuerelement selbst im Designer bearbeiten zu können.

## 122 Scroll-Balken eines Benutzersteuerelementes im Design-Mode verfügbar machen

Wenn Sie die Scroll-Balken eines Benutzersteuerelementes sichtbar machen, indem Sie die Eigenschaften `AutoScroll` auf `True` und `AutoScrollMinSize` auf passende Werte setzen, dann werden sie im Designer zwar angezeigt, lassen sich aber nicht verändern. Erst zur Laufzeit können sie bewegt werden.

Wollen Sie erreichen, dass die Scroll-Balken Ihres Steuerelementes, wenn es im Designer auf einer Form oder einem Fenster platziert wird, bereits zur Design-Zeit betätigt werden können, dann brauchen Sie lediglich, wie im vorangegangenen Beispiel, der Klasse ein Attribut zuzuweisen:

```
Imports System.Windows.Forms
Imports System.ComponentModel
Imports System.Windows.Forms.Design

<Designer(GetType(ScrollableControlDesigner))> _
Public Class NameDesSteuerelementes
    Inherits System.Windows.Forms.UserControl
```

Die Klasse `ScrollableControlDesigner` ist von `ParentControlDesigner` abgeleitet, so dass das Steuerelement automatisch auch zum Container wird.

Auch hier können Sie wieder alternativ die Klasse ableiten von der Klasse `ContainerControl` oder deren Basisklasse `ScrollableControl`.

## 123 Benutzersteuerelemente und die Text-Eigenschaft

Viele Steuerelemente haben eine Eigenschaft zum Einstellen angezeigter Texte. Gab es hierfür früher noch viele verschiedene Namen wie `Caption`, `Text`, `Value` etc., wird in .NET hauptsächlich die Eigenschaft `Text` zu diesem Zweck verwendet. Im Gegensatz zu anderen Eigenschaften gibt es hier aber eine Namenskollision mit der Eigenschaft `Text`, die bereits in der Klasse `Control`, von der die Klassen `UserControl` und somit auch Ihr Steuerelement abgeleitet sind, definiert ist. Der erste Schritt besteht daher darin, die Eigenschaft als Überschreibung zu deklarieren. Im Designer wird sie dann aber trotzdem noch nicht angezeigt, da das `Browsable`-Attribut in der Voreinstellung auf `False` steht. Auch dieses müssen Sie ändern.

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Propert-  
yGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

```

<Browsable(True), _
DesignerSerializationVisibility( _
DesignerSerializationVisibility.Visible)> _
Public Overrides Property Text() As String
    Get
        Return MyBase.Text
    End Get
    Set(ByVal Value As String)
        MyBase.Text = Value
        Refresh()
    End Set
End Property

```

---

*Listing 191: Text-Property für eigenes Benutzersteuerelement definieren*

Eine weitere Hürde stellt der Designer auf, indem er nicht automatisch wie bei anderen Eigenschaften den Code für die Wertzuweisung in `InitializeComponent` einträgt. Die Folge ist, dass beim Programmstart der zugewiesene Text verloren geht. Erst durch ein weiteres Attribut (`DesignerSerializationVisibility`) wird er dazu überredet, auch den Code für die Zuweisung an die Text-Eigenschaft vorzunehmen.

## 124 PanelGroupBoxPicturebox – ein Steuerelement für alle Fälle

Vielleicht kennen Sie das Problem: Sie benötigen ein Steuerelement, auf das Sie andere Steuerelemente platzieren können (also ein Container-Control), das zusätzlich eine skalierte Grafik im Hintergrund anzeigt, und Sie möchten den Rahmen individuell einstellen können. Bei Bedarf sollte, wie bei der `GroupBox`, eine Überschrift angezeigt werden. Im Framework werden Ihnen drei Steuerelemente angeboten, die von jedem etwas implementieren: `GroupBox`, `Panel` und `PictureBox`. Ein Steuerelement, bei dem Sie den Rahmen einstellen können, wie es früher in VB6 möglich war, gibt es leider nicht.

Die Lösung heißt daher: selber machen! Abbildung 99 zeigt eine mögliche Variante des im Folgenden beschriebenen Steuerelementes. Andere Steuerelemente, wie Labels, Schaltflächen, Check-Boxen etc., lassen sich auf diesem Control ablegen. Definiert man die Hintergrundfarbe der Controls als `Transparent`, dann ergeben sich Effekte wie die in Abbildung 99 gezeigten durchsichtigen Schaltflächen und Labels.

Benötigt wird zunächst einmal ein Benutzersteuerelement, das hier den Namen `PanelGroupBoxPicturebox` erhalten soll. Aufbauend auf den vorausgegangenen Rezepten wird es wie folgt definiert:

---

```

<Designer(GetType(ScrollableControlDesigner)), _
ToolboxBitmapAttribute(GetType(PanelGroupBoxPicturebox))> _
Public Class PanelGroupBoxPicturebox
    Inherits System.Windows.Forms.UserControl

```

---

Im Konstruktor, nach `InitializeComponent`, wird der Stil `ResizeRedraw` auf `True` gesetzt, damit das Steuerelement automatisch neu gezeichnet wird, wenn sich seine Größe geändert hat.

---

```

SetStyle(ControlStyles.ResizeRedraw, True)

```

---

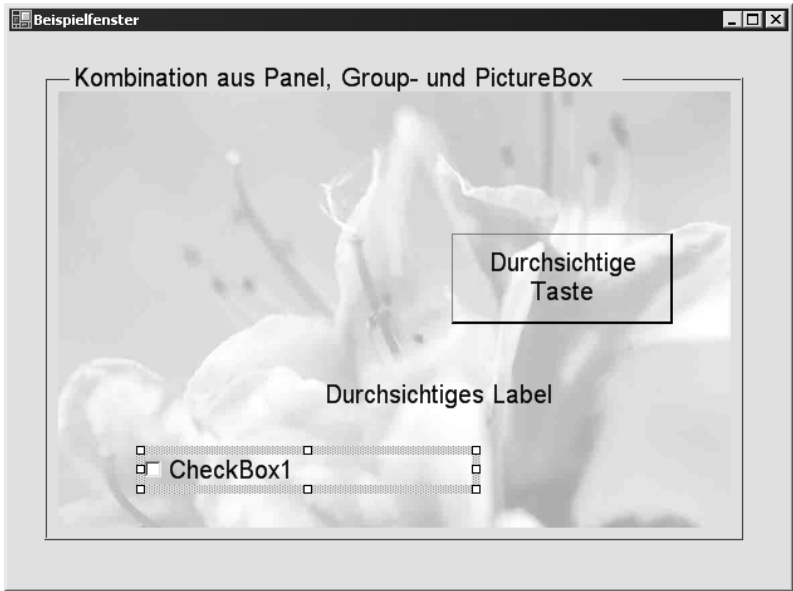


Abbildung 99: Die Eigenbau-Kombination erlaubt 3D-Rahmen, Titel und Hintergrundbild sowie die Ablage anderer Steuerelemente

Zur individuellen Parametrierung des Steuerelementes werden die in Tabelle 21 beschriebenen öffentlichen Eigenschaften definiert. Listing 192 zeigt die Implementierung der Property-Methoden, Listing 193 die geschützten Member-Variablen sowie die Enumeration für den Modus der Bilddarstellung.

| Name             | Bedeutung                                                                                  | Member-Variable       |
|------------------|--------------------------------------------------------------------------------------------|-----------------------|
| Text             | String, der als Titel gezeichnet werden soll                                               | MyBase.Text           |
| Bordersize       | Breite des Rahmens                                                                         | PanelBorderSize       |
| BorderStyleOuter | Rahmenstil des äußeren Rahmens                                                             | PanelBorderStyleOuter |
| BorderStyleInner | Rahmenstil des inneren Rahmens                                                             | PanelBorderStyleInner |
| Image            | Das im inneren Rahmen darzustellende Bild. Nothing, wenn kein Bild dargestellt werden soll | Panel Image           |
| ImageMode        | Modus zum Zeichnen des Bildes                                                              | Panel ImageMode       |

Tabelle 21: Zusätzliche Eigenschaften des PanelGroupPictureBox-Controls

```
' Die Text-Eigenschaft
<Browsable(True), _
DesignerSerializationVisibility(_
DesignerSerializationVisibility.Visible)> _
Public Overrides Property Text() As String
```

Listing 192: Implementierung der Eigenschaften

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Propert-  
yGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

```

    Get
        Return MyBase.Text
    End Get
    Set(ByVal Value As String)
        MyBase.Text = Value
        Refresh()
    End Set
End Property

<Description("Breite des Rahmens")> _
Public Property Bordersize() As Integer
    Get
        Return PanelBordersize
    End Get
    Set(ByVal Value As Integer)
        If Value < 1 Then Throw New ArgumentException( _
            "Wert muss größer sein als 0")
        If Value > Width \ 2 Then Throw New ArgumentException( _
            "Wert darf nicht größer sein als Width/2")
        PanelBordersize = Value
        Refresh()
    End Set
End Property

<Description("Darstellung der inneren Rahmenlinien")> _
Public Property BorderstyleInner() As Border3DStyle
    Get
        Return PanelBorderStyleInner
    End Get
    Set(ByVal Value As Border3DStyle)
        PanelBorderStyleInner = Value
        Refresh()
    End Set
End Property

<Description("Darstellung der äußeren Rahmenlinien")> _
Public Property BorderstyleOuter() As Border3DStyle
    Get
        Return PanelBorderStyleOuter
    End Get
    Set(ByVal Value As Border3DStyle)
        PanelBorderStyleOuter = Value
        Refresh()
    End Set
End Property

<System.ComponentModel.DefaultValue(GetType(Object), Nothing), _
Description("Bild, das im inneren Rahmen dargestellt werden soll")> _
Public Property Image() As Image
    Get
        Return PanelImage
    End Get
    Set(ByVal Value As Image)
        PanelImage = Value
        Refresh()
    End Set
End Property

```

---



```

End Get
Set(ByVal Value As Image)
    If Not PanelImage Is Nothing Then PanelImage.Dispose()
    PanelImage = Value
    Refresh()
End Set
End Property

<System.ComponentModel.DefaultValue(ImageMode.Stretch), _
Description("Darstellungsmodus des Bildes")> _
Public Property ImageStretchMode() As ImageMode
    Get
        Return PanelImageMode
    End Get
    Set(ByVal Value As ImageMode)
        PanelImageMode = Value
        Refresh()
    End Set
End Property

```

Listing 192: Implementierung der Eigenschaften (Forts.)

```

' Modi für Bildskalierungen
Public Enum ImageMode
    NoScale
    Stretch
    BestFit
End Enum

' Geschützte Variablen für die Einstellungen
Protected PanelBorderSize As Integer = 2
Protected PanelBorderStyleOuter As Border3DStyle = _
    Border3DStyle.SunkenOuter
Protected PanelBorderStyleInner As Border3DStyle = _
    Border3DStyle.RaisedInner
Protected PanelImage As Image
Protected PanelImageMode As ImageMode = ImageMode.Stretch

```

Listing 193: Membervariablen und Enumeration für Bilddarstellung

In Rezept [Z.8](#) ((Benutzersteuerelemente und die Text-Eigenschaft)) wurde bereits der Aufbau der Text-Eigenschaft beschrieben. In der Property-Set-Methode von `BorderSize` wird der Wert auf Plausibilität geprüft und im Fehlerfall eine `Exception` ausgelöst. Gibt ein Anwender im Eigenschaftsfenster des Designers einen ungültigen Wert ein, wird eine Fehlermeldung mit dem Text der `Exception` angezeigt und der Wert abgewiesen. Mit `Refresh` wird das Neuzeichnen des Steuerelementes erzwungen, so dass die Änderung auch im Designer sofort sichtbar wird. `BorderSize` bestimmt den Abstand zwischen dem äußeren und dem inneren Rahmen.

`BorderStyleInner` und `BoderstyleOuter` nehmen alle Werte der Enumeration `System.Windows.Forms.Border3DStyle` an. Diese beinhaltet alle Konstanten, die zum Zeichnen von 3D-Steuerelementen mittels `ControlPaint` verwendet werden können. Somit stehen alle Variationen zur Verfügung, um Anhebungen, Absenkungen usw. darzustellen. Soll ein Rahmen, z.B. der

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Propert-  
yGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

innere Rahmen, nicht dargestellt werden, dann muss der entsprechenden Eigenschaft `Border3DStyle.Adjust` zugewiesen werden. Abbildung 100 zeigt einige Variationen der Rahmenkombinationen.

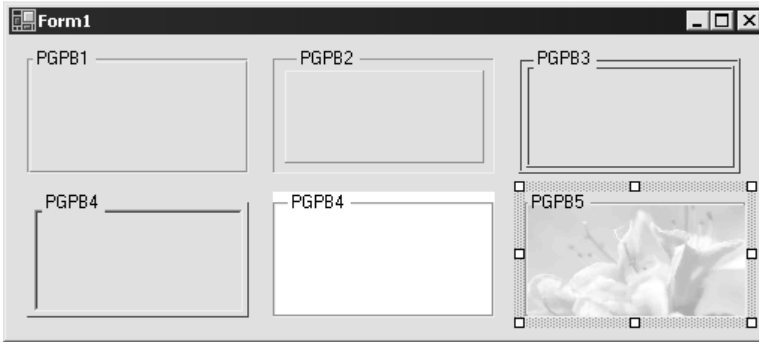


Abbildung 100: Eine Auswahl an Variationen mit verschiedenen Rahmen und Hintergründen

Mit der Eigenschaft `Image` wird die Möglichkeit bereitgestellt, bereits im Designer ein Bild auszuwählen und zuzuweisen. Der Designer zeigt zu diesem Zweck den Standarddialog zum Öffnen von Dateien an und lässt nur Bilddateien zu. Damit der Anwender auch im Eigenschaftsfenster das Bild durch Löschen des Textes wieder entfernen kann, muss der Property-Definition das Attribut `DefaultValue` zugewiesen werden. Der Wert des Attributes, hier `Nothing`, wird der Eigenschaft zugewiesen, wenn der Eintrag im Eigenschaftsfenster gelöscht wird.

`ImageStretchMode` nimmt die Werte der Enumeration `ImageMode` an. Diese können sein:

► **NoScale**

Das Bild wird nicht skaliert. Es wird innerhalb des inneren Rahmens gezeichnet. Bildteile, die über den Rahmen hinausragen, werden abgeschnitten.

► **Stretch**

Das Bild wird so skaliert, dass es vollständig die Fläche innerhalb des inneren Rahmens ausfüllt. Dabei wird es ggf. verzerrt dargestellt.

► **BestFit**

Das Bild wird unter Beibehaltung der Seitenverhältnisse vergrößert oder verkleinert, so dass es die Fläche des inneren Rahmens optimal ausfüllt. Siehe hierzu auch Rezept 5.9 ((Bilder verzerrungsfrei maximieren)).

Gezeichnet werden die Bestandteile des Steuerelementes in der Überschrift von `OnPaintBackground` (siehe Listing 194) in der folgenden Reihenfolge:

1. Hintergrundfüllung
2. Äußerer Rahmen
3. Innerer Rahmen
4. Hintergrund des Textfeldes
5. Falls zugewiesen, das Bild
6. Der Titeltext

Für Hintergrund und Text werden die Eigenschaften `ForeColor` und `BackColor` herangezogen. Der Text wird mit dem eingestellten Font gezeichnet. So können die von der Basisklasse `UserControl` bereitgestellten Eigenschaften vorteilhaft und ohne zusätzlichen Aufwand genutzt werden.

Das Zeichnen des Bildes erfolgt auf der inneren Fläche des inneren Rahmens (im Listing Variable `r3`). Bei der Einstellung `NoScale` wird zusätzlich ein Clipping-Bereich vorgegeben, außerhalb dessen das Bild abgeschnitten wird. Die maximierte Darstellung `BestFit` erfolgt in der Methode `DrawMaximizePicture`, die in Rezept [5.9](#) ((Bilder verzerrungsfrei maximieren)) näher beschrieben wird.

```
' Zeichnen des Hintergrundes
Protected Overrides Sub OnPaintBackground(ByVal e As _
    System.Windows.Forms.PaintEventArgs)

    Dim g As Graphics = e.Graphics

    ' Koordinaten für Rahmen und Füllung
    Dim ts As SizeF = g.MeasureString(Text, Me.Font)
    Dim x As Integer = PanelBordersize * 2
    Dim y As Integer = CInt(ts.Height) \ 2

    ' Der äußere Rahmen
    Dim r1 As New Rectangle(0, y, Width, Height - y)

    ' Der innere Rahmen
    Dim r2 As New Rectangle(PanelBordersize, PanelBordersize + y, _
        Width - PanelBordersize * 2 + 1, _
        Height - y - 2 * PanelBordersize + 1)

    ' Der Bildbereich
    Dim r3 As New Rectangle(PanelBordersize + 1, _
        PanelBordersize + y + 1, Width - 2 * PanelBordersize - 1, _
        Height - 2 * PanelBordersize - 1 - y)

    ' Füllen mit Hintergrundfarbe
    g.Clear(Me.BackColor)

    ' Äußeren Rahmen zeichnen
    ControlPaint.DrawBorder3D(g, r1, PanelBorderStyleOuter)

    ' Inneren Rahmen zeichnen
    ControlPaint.DrawBorder3D(g, r2, PanelBorderStyleInner)

    ' Brush-Objekte für Vorder- und Hintergrundfarbe
    Dim brb As Brush = New SolidBrush(Me.BackColor)
    Dim brf As Brush = New SolidBrush(Me.ForeColor)

    ' Textbereich füllen mit Hintergrundfarbe
    e.Graphics.FillRectangle(brb, PanelBordersize * 2, 0, _
        ts.Width, ts.Height)
```

Listing 194: Zeichnen der Bestandteile des Steuerelementes in `OnPaintBackground`

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Propor-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

```

' Bild darstellen, wenn vorhanden
If Not PanelImage Is Nothing Then
    Select Case PanelImageMode
        ' Bild auf inneren Bereich ausdehnen
        Case ImageMode.Stretch
            g.DrawImage(PanelImage, r3)

        ' Keine Skalierung, Bild abschneiden
        Case ImageMode.NoScale
            g.SetClip(r3)
            g.DrawImage(PanelImage, r3.Left, r3.Top, _
                PanelImage.Width, PanelImage.Height)
            g.ResetClip()

        ' Maximale Größe, zentrierte Darstellung
        Case ImageMode.BestFit
            DrawMaximizedPicture(g, PanelImage, r3)

    Case Else
        Throw New ArgumentException( _
            "Unbekannter Wert für ImageStretchMode")
    End Select
End If

' Text zeichnen
e.Graphics.DrawString(Text, Me.Font, brf, _
    PanelBordersize * 2, 0)

' Brushes entsorgen
brb.Dispose()
brf.Dispose()

```

End Sub

---

*Listing 194: Zeichnen der Bestandteile des Steuerelementes in OnPaintBackground (Forts.)*

Mit dem vorliegenden Sourcecode steht Ihnen die Möglichkeit offen, weitere Features zu implementieren oder vorhandene Ihren Wünschen anzupassen.

## 125 Einem Benutzersteuerelement ein Icon zuweisen

Um einem selbst definierten Benutzersteuerelement ein eigenes Icon zuzuweisen, das in der Toolbox dargestellt werden kann, gehen Sie folgendermaßen vor:

1. Erstellen Sie das Bild als Bitmap, vorzugsweise 16 x 16 Pixel
2. Geben Sie der Bilddatei den gleichen Namen wie dem Steuerelement und fügen Sie es dem Projekt hinzu
3. Stellen Sie in den Eigenschaften der Bilddatei unter BUILDAKTION EINGEBETTETE RESSOURCE ein (Abbildung 101)
4. Weisen Sie der Klasse des Steuerelementes das Attribut `ToolboxBitmapAttribute` zu.

So wird ein Icon über ein Attribut zugewiesen:

```
<ToolboxBitmapAttribute(GetType(PanelGroupPictureBox))> _
Public Class PanelGroupPictureBox
    Inherits System.Windows.Forms.UserControl
```



Abbildung 101: Icon für Benutzersteuerelement als eingebettete Ressource

## 126 Transparenz eines Steuerelementes im Designer einstellen

Immer wieder wird die Frage gestellt, wie man denn im Designer beispielsweise für ein Label eine transparente Hintergrundfarbe einstellen kann. Da die Lösung nicht besonders intuitiv ist, soll sie kurz erläutert werden.

Die Transparenz einer Farbe wird über ihren Alpha-Wert gesteuert. Alpha = 0 bedeutet hierbei eine völlige Transparenz. Im Designer werden die Farbwerte aber zunächst als RGB-Wert und nicht als ARGB-Wert angezeigt und die Auswahl an System-, Web- und benutzerdefinierten Farben helfen auch nicht weiter. Es hält Sie aber niemand davon ab, den Farbcode z.B. als Hex-Zahl einzutragen. Möchten Sie einen völlig transparenten Hintergrund, also mit Alpha-Wert = 0, dann spielt die Farbe ohnehin keine Rolle. Markieren Sie mit einem Doppelklick auf die `BackColor`-Eigenschaft den kompletten Zahlenwert, geben Sie »0« ein und betätigen Sie die Eingabetaste. Ein ARGB-Wert von 0;0;0;0 wird eingetragen (siehe Abbildung 102) und der Hintergrund des Steuerelementes wird transparent.

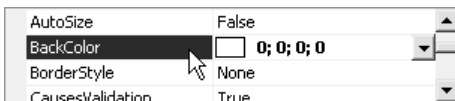


Abbildung 102: So stellen Sie einen transparenten Hintergrund für ein Steuerelement ein

## 127 Abfangen von Windows-Nachrichten

Manchmal reichen die vom Framework zur Verfügung gestellten Ereignisse eines Steuerelementes nicht aus, da sie meist nicht alle Windows-Nachrichten zur Verfügung stellen. Benötigt man beispielsweise das `Paint`-Ereignis einer `ListView` oder einer `TreeView`, dann steht das Ereignis selbst nicht zur Verfügung. Zwar können Sie die `OnPaint`-Methode überschreiben, sie wird aber in der Regel nicht aufgerufen.

Verschiedene Wege erlauben Ihnen, direkt die von Windows an das Steuerelement (oder das Fenster) gesendeten Nachrichten abzufangen.

1. Setzen des Stils `EnableNotifyMessage` mit `SetStyle`
2. Überschreiben der `WndProc`-Methode
3. Bindung eines `NativeWindow`-Objektes an das Steuerelement
4. Eingriff in die Nachrichtenschleife des ausführenden Threads

## EnableNotifyMessage

Durch Setzen des Fensterstils `EnableNotifyMessage` auf `True` erreichen Sie, dass für alle Windows-Botschaften automatisch die Methode `OnNotifyMessage` aufgerufen wird (Listing 195). Indem Sie die Methode `OnNotifyMessage` überschreiben, können Sie auf jede Nachricht reagieren, die das Steuerelement erhalten hat. Die Ausgabe des im Listing abgebildeten Codes könnte z.B. so aussehen:

---

```
msg=0x84 (WM_NCHITTEST) hwnd=0xa079a wparam=0x0 lparam=0x12d01fe
msg=0x200 (WM_MOUSEMOVE) hwnd=0xa079a wparam=0x0 lparam=0x270061
msg=0x84 (WM_NCHITTEST) hwnd=0xa079a wparam=0x0 lparam=0x1270211
msg=0x84 (WM_NCHITTEST) hwnd=0xa079a wparam=0x0 lparam=0x1270211
msg=0x200 (WM_MOUSEMOVE) hwnd=0xa079a wparam=0x0 lparam=0x210074
msg=0x2a3 (WM_MOUSELEAVE) hwnd=0xa079a wparam=0x0 lparam=0x0
msg=0x85 (WM_NCPAINT) hwnd=0xa079a wparam=0x1 lparam=0x0
msg=0x14 (WM_ERASEBKGD) hwnd=0xa079a wparam=0x70104eb lparam=0x0
msg=0x7 (WM_SETFOCUS) hwnd=0xa079a wparam=0x0 lparam=0x0
msg=0xf (WM_PAINT) hwnd=0xa079a wparam=0x0 lparam=0x0
msg=0x4e (WM_NOTIFY) hwnd=0xa079a wparam=0x0 lparam=0x12d8e4
msg=0x100c hwnd=0xa079a wparam=0xffffffffffffffff lparam=0x1
msg=0x84 (WM_NCHITTEST) hwnd=0xa079a wparam=0x0 lparam=0x1300219
```

---

Der Vorteil dieser Vorgehensweise ist, dass Sie nicht die Methode der Basisklasse aufrufen müssen. Die Implementierung von `OnNotifyMessage`, die vom Framework vorgesehen wird, ist leer. Sie können somit bedenkenlos diese Methode für weitere Ableitungen Ihres Steuerelementes für spätere Überschreibungen zur Verfügung stellen.

Eine Filterung, um bestimmte Nachrichten dem Steuerelement vorzuenthalten, ist nicht möglich, da die `OnNotifyMessage`-Methode lediglich mithört, aber nicht in die Nachrichtenverarbeitung eingreifen kann. Ist das erforderlich, müssen Sie auf eine der anderen Varianten zurückgreifen.

---

```
Public Class SubclassingA
    Inherits ListView

    Public Sub New()
        MyBase.New()
        SetStyle(ControlStyles.EnableNotifyMessage, True)
    End Sub

    Protected Overrides Sub OnNotifyMessage( _
        ByVal m As System.Windows.Forms.Message)
```

---

---

```
' Nachricht bearbeiten
Debug.WriteLine(m.ToString)
```

```
End Sub
```

```
End Class
```

---

*Listing 195: Abfangen der Windows-Nachrichten in OnNotifyMessage (Forts.)*

## Überschreiben der WndProc-Methode

Statt den `NotifyMessage`-Stil zu setzen, können Sie auch direkt die Methode `WndProc` überschreiben (Listing 196). Das Ergebnis ist das gleiche, allerdings ist es hierbei zwingend erforderlich, die `WndProc`-Methode der Basisklasse aufzurufen. Wird das vergessen, dann sind Laufzeitfehler vorprogrammiert, da die interne Nachrichtenverarbeitung in dieser Methode vorgenommen wird, die Sie ja mit der Überschreibung außer Kraft setzen.

Sie können aber gezielt einzelne Nachrichten abfangen und nicht an das Steuerelement weiterleiten und so eine aktive Filterung programmieren. Auch können Sie die Nachrichten selbst verändern, bevor Sie sie weiterleiten.

---

```
Public Class SubclassingB
    Inherits ListView

    Protected Overrides Sub WndProc( _
        ByRef m As System.Windows.Forms.Message)

        ' Wichtig! Aufruf der Methode der Basisklasse
        MyBase.WndProc(m)

        ' Bearbeitung der Message
        Debug.WriteLine(m.ToString)

    End Sub

End Class
```

---

*Listing 196: Abfangen der Windows-Nachrichten durch Überschreibung der WndProc-Methode*

## Abfangen der Nachrichten in einem NativeWindow-Objekt

Besteht keine Möglichkeit, durch Subclassing eine Methode zu überschreiben, dann lässt sich alternativ mit Hilfe der Klasse `NativeWindow` die Nachrichtenbearbeitung eines Steuerelementes bzw. Fensters umleiten. Durch Aufruf der Methode `AssignHandle` binden Sie die Nachrichtenschleife eines Steuerelementes an eine Instanz von `NativeWindow`. Die Methode lenkt den Nachrichtenstrom an die `WndProc`-Methode der `NativeWindow`-Instanz weiter. Diese wiederum ruft die `WndProc`-Methode des Steuerelementes auf.

Um an die Nachrichten zu gelangen, müssen Sie also lediglich eine eigene Klasse von `NativeWindow` ableiten und die `WndProc`-Methode überschreiben. Im Beispiel (siehe Listing 197) wird ein Benutzersteuerelement von `UserControl` abgeleitet. Es enthält als untergeordnetes Steuerelement eine `ListView`-Instanz; `ListView1` verweist auf dieses Control. `NativeWindowLV` ist eine von `NativeWindow` abgeleitete Klasse, `NativeWindowForListView` verweist auf eine Instanz dieser Klasse. In ihr

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Propert-  
yGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

wird die `WndProc`-Methode überschrieben. Da hierdurch der Nachrichtenfluss zum Steuerelement unterbrochen wird, muss auch hier unbedingt die `WndProc`-Methode der Basisklasse aufgerufen werden. Danach oder auch davor können Sie die Bearbeitung der Nachrichten vorsehen.

Die Bindung der `NativeWindowLV`-Instanz kann erst erfolgen, wenn das Steuerelement existiert und auch tatsächlich ein Handle besitzt. Ein günstiges Ereignis hierfür ist das `HandleCreated`-Event des zu bindenden Steuerelementes. Hier kann die Methode `AssignHandle` aufgerufen und das Handle des Steuerelementes übergeben werden.

Leider erfolgt die Freigabe nicht automatisch. Wird das Steuerelement zerstört, dann muss auch das Handle wieder freigegeben werden. Analog zum Anlegen erfolgt die Freigabe üblicherweise im Ereignis `HandleDestroyed` durch Aufruf der Methode `ReleaseHandle`.

---

```
Public Class NativeWindowMessages
    Inherits System.Windows.Forms.UserControl

    ' Referenz der NativeWindow-Instanz
    Private NativeWindowForListView As New NativeWindowLV

    Private Sub ListView1_HandleCreated(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles ListView1.HandleCreated

        ' Bindung der NativeWindow-Instanz an das Handle der ListView
        NativeWindowForListView.AssignHandle(ListView1.Handle)

    End Sub

    Private Sub ListView1_HandleDestroyed(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles ListView1.HandleDestroyed

        ' Handle wieder freigeben
        NativeWindowForListView.ReleaseHandle()

    End Sub

    ' Ableitung von NativeWindow zur Überschreibung von WndProc
    Private Class NativeWindowLV
        Inherits NativeWindow

        ' Überschreibung der WndProc-Methode
        Protected Overrides Sub WndProc( _
            ByRef m As System.Windows.Forms.Message)

            ' Wichtig! Delegation an Basisklasse
            MyBase.WndProc(m)

            ' Nachricht bearbeiten
            Debug.WriteLine(m)

        End Sub

    End Class
```

---



---

End Class

---

*Listing 197: Abfangen der Windows-Nachrichten mittels NativeWindow (Forts.)*

Diese Variante bietet Ihnen die gleichen Möglichkeiten wie die vorige (Überschreiben der WndProc-Methode des abgeleiteten Steuerelementes). Auch hier können Sie Nachrichten herausfiltern oder verändern.

Hinweis

Gehen Sie äußerst umsichtig vor, wenn Sie die WndProc-Methode überschreiben. Bedenken Sie, dass der Designer eine Instanz des Steuerelementes anlegt, wenn das Steuerelement auf einem Fenster platziert wird. Die Instanziierung erfolgt im Prozess der Entwicklungsumgebung. Fehlerhafte WndProc-Überschreibungen führen daher schnell zum Absturz derselben, so dass Sie einmal mehr den Task-Manager bemühen müssen ;-).

### Eingriff in die Nachrichtenschleife des ausführenden Threads

Diese Variante setzt ein paar Ebenen weiter oben an. Statt gezielt die Nachrichten, die an ein spezielles Steuerelement gesendet werden, abzufangen, werden alle Nachrichten, die die Message-Loop eines Threads empfängt, umgeleitet. Die Application-Klasse stellt hierfür die Methoden AddMessageFilter und RemoveMessageFilter bereit.

Für die Umleitung muss eine Referenz vom Typ IMessageFilter angegeben werden. Benötigt wird also eine Klasse, die diese Schnittstelle implementiert. Als einzige Methode muss PreFilterMessage implementiert werden. Diese Methode wird für jede Nachricht, die an den Thread gesendet wird, aufgerufen. Neben der Message-ID müssen Sie hier zusätzlich das Handle überprüfen, wenn Sie erfahren wollen, welchem Steuerelement oder Fenster die Nachrichten zugeordnet sind.

Der Rückgabewert der Methode PreFilterMessage ist vom Typ Boolean und gibt an, ob die Nachricht weiterverarbeitet (False) oder ob sie herausgefiltert werden soll (True). Listing 198 zeigt die Klasse ThreadMessageFilter, die mit Hilfe dieser Schnittstelle die Nachrichten verarbeiten kann.

---

```
Public Class ThreadMessageFilter
    Implements IMessageFilter

    ' Einzige Methode der Schnittstelle IMessageFilter
    Public Function PreFilterMessage( _
        ByRef m As System.Windows.Forms.Message) As Boolean _
        Implements System.Windows.Forms.IMessageFilter.PreFilterMessage

        ' Nachricht bearbeiten
        Debug.WriteLine(m)

        ' Rückgabe False, wenn die Nachricht weiterverarbeitet werden
        ' soll
        Return False
    End Function
End Class
```

---

*Listing 198: Implementierung der Schnittstelle IMessageFilter*

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Proper-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

Ein Beispielfenster mit zwei Schaltflächen (BTNTThreadMESSAGESTART und BTNTThreadMESSAGEFILTERSTOP) ermöglicht, die Umleitung der Nachrichten ein- und auszuschalten (Listing 199). Hierzu wird eine Instanz der Klasse `ThreadMessageFilter` angelegt und deren Referenz an die Methoden `AddMessageFilter` respektive `RemoveMessageFilter` übergeben.

---

```
Public Class MainWindow
    Inherits System.Windows.Forms.Form

    ...

    ' Referenz des Filter-Objektes
    Private MessageFilter As New ThreadMessageFilter

    Private Sub BTNTThreadMessageStart_Click( _
        ByVal sender As System.Object, ByVal e As System.EventArgs) _
        Handles BTNTThreadMessageStart.Click

        ' Filter-Objekt hinzufügen
        Application.AddMessageFilter(MessageFilter)

    End Sub

    Private Sub BTNTThreadMessageFilterStop_Click( _
        ByVal sender As System.Object, ByVal e As System.EventArgs) _
        Handles BTNTThreadMessageFilterStop.Click

        ' Filter-Objekt wieder entfernen
        Application.RemoveMessageFilter(MessageFilter)

    End Sub
End Class
```

---

Listing 199: Ein- und Ausschalten der Nachrichtenumleitung an das `MessageFilter`-Objekt

Dieser Eingriff in die Nachrichtenschleife des Threads benötigt mit Abstand die meiste Prozessorzeit, da die Filter-Methode für jede Nachricht, egal für welches Steuerelement oder Fenster, aufgerufen wird. Sie sollten sie deswegen nur einsetzen, wenn die anderen Lösungen nicht geeignet sind.

## Bearbeitung der Nachrichten

Um auf spezielle Nachrichten zu reagieren wird gewöhnlich in einer `Select Case`-Anweisung auf Basis der Nachrichtennummern, hier `m.Msg`, eine Verzweigung aufgebaut. Die in Frage kommenden Konstanten und ihre Bedeutungen finden Sie in der MSDN-Dokumentation, die zugehörigen Zahlenwerte leider nicht. Diese finden Sie nur in der C-Header-Datei, die in der Dokumentation genannt wird. Beachten Sie hierbei, dass die meisten Werte hexadezimal angegeben sind.

Nachrichten werden unter Windows mit den in Tabelle 22 aufgeführten Werten verschickt. Der Parameter der `WndProc`-bzw. `OnNotifyMessage`-Aufrufe ist vom Typ `Message`, einer Klasse, die lediglich diese Werte kapselt und unter .NET zur Verfügung stellt. Beispiele für den Einsatz der vorgestellten Nachrichtenfilter finden Sie in den nachfolgenden Rezepten.

| Eigenschaft | Bedeutung                                |
|-------------|------------------------------------------|
| HWND        | Handle des Fensters bzw. Steuerelementes |
| Msg         | ID-Nummer der Nachricht                  |
| Lparam      | 1. Parameter der Nachricht               |
| Wparam      | 2. Parameter der Nachricht               |
| Result      | Rückgabewert an Windows                  |

Tabelle 22: Eigenschaften der Klasse Message

## 128 Steuerelement für Verzeichnisauswahl

Windows stellt eine Reihe von Standarddialogen für verschiedene Aufgaben bereit. Dazu gehören beispielsweise Dialoge zum Öffnen und Speichern von Dateien. Auch ein Dialog für die Auswahl von Verzeichnissen existiert, ist aber etwas verborgen bzw. nur eingeschränkt nutzbar. Der `FolderBrowserDialog` wird vom Framework Version 1.1 angeboten, ist allerdings in der Ursprungsversion derart fehlerhaft, dass die meisten Entwickler auf die API-Funktion `SHBrowseForFolder` zurückgreifen. Diesen Dialog zu erweitern oder in einen eigenen Dialog zu integrieren, ist leider kaum möglich.

Abhilfe schafft die Programmierung eines benutzerdefinierten Steuerelementes, das die Verzeichnisstruktur in einem `TreeView`-Control abbildet. Der Aufwand für ein solches Steuerelement mit Basisfunktionalitäten (Anzeige der lokalen Laufwerke, keine Aktualisierung bei Änderungen des Dateisystems) ist vergleichsweise gering und wird im Folgenden beschrieben. Ziel ist ein Steuerelement wie in Abbildung 103, das im Designer verwendet und eingestellt werden kann.

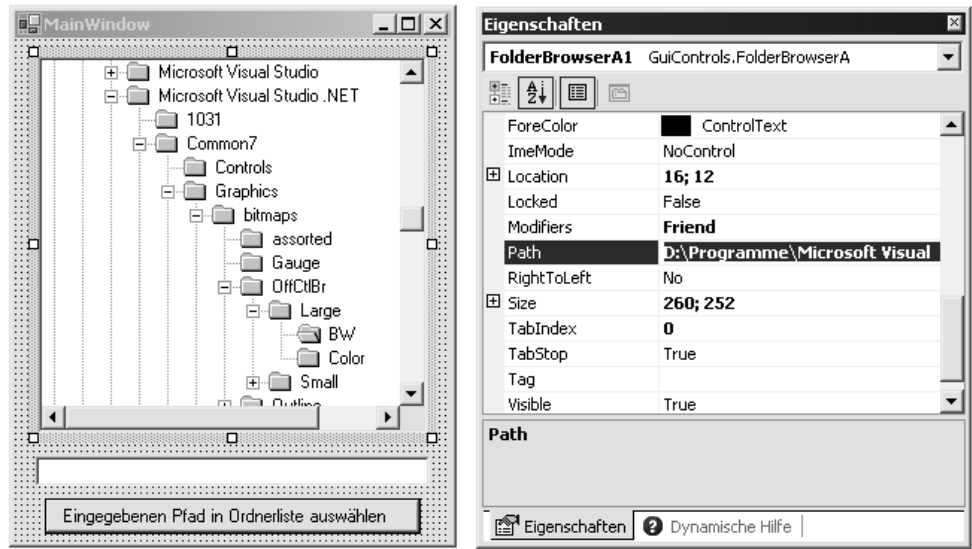


Abbildung 103: Steuerelement für die Verzeichnisauswahl zur Design-Zeit

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows

Forms

Controls

Propor-  
tityGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

Zwei Möglichkeiten bieten sich, das Steuerelement zu definieren:

1. Subclassing durch Ableitung von `TreeView`
2. Definition eines Benutzersteuerelementes, das ein `TreeView-Control` beinhaltet

Die erste Variante lässt sich recht elegant realisieren, hat jedoch den Nachteil, dass der Anwender auf alle Eigenschaften und Methoden der `TreeView`-Basisklasse zugreifen kann. Fügt er beispielsweise eigene Knoten hinzu, kann das die Programmlogik des Verzeichnisdialogs empfindlich stören. Besser ist daher die etwas aufwändigere Variante, ein Benutzersteuerelement anzulegen und ein `TreeView-Control` zu definieren, das dessen Client-Bereich vollständig ausfüllt. So sieht das Steuerelement für den Programmierer, der es im Designer einsetzt, genauso aus wie eine `TreeView`, nur eben mit vordefinierten Knoten. Eigenschaften und Methoden können gezielt auf das eigentliche `TreeView-Control` abgestimmt werden.

Im Beispielprogramm wird das Steuerelement in einer Klassenbibliothek (`GuiControls`) abgelegt, so dass es auch für andere Programme zur Verfügung steht. Das `TreeView-Control` (`TVDirectories`) wird auf dem Benutzersteuerelement (`FolderBrowserA`) platziert. Durch Setzen der Property `Docking` auf `Fill` füllt es den gesamten Client-Bereich aus.

### Initialisierung des Verzeichnisbaumes mit den Bezeichnungen der lokalen Laufwerke

Mit Hilfe einer rekursiven Methode könnte man die gesamte Verzeichnisstruktur des lokalen Dateisystems durchlaufen (siehe z.B. Rezept 9.8 ((Verzeichnisgröße mit Unterverzeichnissen))) und die `TreeView` mit den Daten füllen. Heutige PCs haben auf ihren Festplatten aber oft tausende Dateien und Ordner gespeichert. Eine vollständige Durchsicht aller Ordner würde daher viel zu viel Zeit in Anspruch nehmen. Aus diesem Grund wird im ersten Schritt nur die oberste Ebene, nämlich die der lokalen Laufwerke dargestellt. Die Unterverzeichnisse eines Laufwerks werden erst bei Bedarf ermittelt, also wenn der betreffende Knoten aufgeklappt wird.

Ein günstiger Zeitpunkt für die Initialisierung ist das `Load`-Ereignis. Der Konstruktor ist zu diesem Zeitpunkt bereits vollständig abgearbeitet, durch den Designer angelegte Zuweisungen an Eigenschaften des Controls sind ausgeführt, das Control ist aber noch nicht sichtbar. Listing 200 zeigt die Implementierung der `OnLoad`-Überschreibung. Mit einer WMI-Abfrage (siehe auch Kapitel 11 ((System))) werden Informationen zu den verfügbaren Laufwerken abgerufen. Hat ein Laufwerk ein Volume-Label, wird es für die Namensanzeige verwendet. Ansonsten wird die textuelle Beschreibung, die Windows über die `Description`-Property liefert, angezeigt. In Klammern wird, analog zum Windows-Explorer, der Laufwerksbuchstabe angegeben.

In der geschützten Methode `AddNode` (siehe Listing 201) wird ein neuer Knoten angelegt und der `Nodes`-Collection der `TreeViews` zugeordnet. Als Rückgabewert liefert die Methode die Referenz des neuen Knotens, der noch ergänzt werden muss.

Die WMI-Abfrage liefert ebenfalls eine Angabe zum Laufwerkstyp, auf deren Basis ein passendes Icon ausgewählt wird. Alle Icons befinden sich in einer `ImageList` (`ImageList1`), die der `TreeView` zugeordnet ist, die Auswahl erfolgt über `Indices`.

Wie eingangs erwähnt, ist es an dieser Stelle nicht sinnvoll, für die einzelnen Laufwerke die Verzeichnisstruktur näher zu untersuchen. Insbesondere bei Wechselmedien (Diskette, CD, DVD) kann es zu Verzögerungen kommen, wenn kein oder gar ein defektes Medium eingelegt ist. Ein `TreeView`-Knoten kann allerdings nur aufgeklappt werden, wenn er mindestens einen Unterknoten hat. Daher wird jedem Laufwerksknoten ein Dummy-Knoten untergeordnet, der als Pfadangabe lediglich einen Punkt enthält. Wird ein solcher Ordner später geöffnet, werden die Informationen für das Verzeichnis nachgeladen.

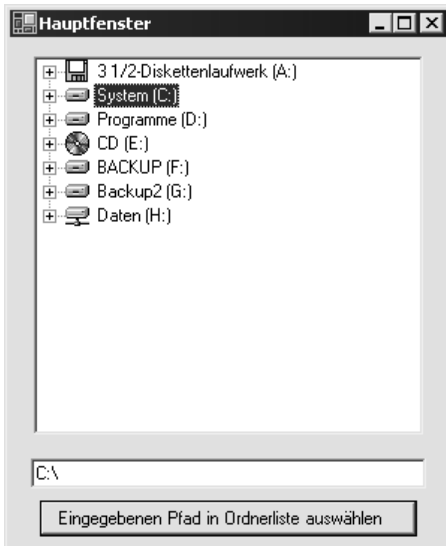


Abbildung 104: Ausgangszustand: Nur die Laufwerke werden angezeigt

```
Protected Overrides Sub OnLoad(ByVal e As System.EventArgs)
    ' TreeView mit Laufwerksbuchstaben und Volume-Bezeichnungen
    ' füllen (oberste Ebene des Verzeichnisbaums)

    Dim tn As TreeNode
    Dim dir As String

    ' Mit WMI Informationen über die Laufwerke holen
    Dim query As New ManagementObjectSearcher("SELECT * FROM " _
        & "Win32_LogicalDisk")

    Dim qc As ManagementObjectCollection = query.Get()
    Dim mo As ManagementObject

    ' Für alle gefundenen Laufwerke
    For Each mo In qc
        Try
            ' Laufwerksbezeichnung (A:, C: ...)
            Dim volID As String = mo.Properties("DeviceID" _
                ).Value.ToString

            ' Typ des Laufwerkes (Festplatte, CD, ...)
            Dim drTyp As Integer = Convert.ToInt32( _
                mo.Properties("DriveType").Value)

            ' Label des Laufwerkes
            Dim volName As String = CStr(mo.Properties( _
                "VolumeName").Value)
```

Listing 200: Initialisierung der TreeView mit Laufwerksbezeichnungen

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Propert-  
yGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schie-  
denes

---

```

' Textuelle Beschreibung
Dim volDesc As String = mo.Properties("Description" _
    ).Value.ToString()

' Wenn vorhanden das Label verwenden, ansonsten die
' Beschreibung einsetzen
Dim t As String
If volName Is Nothing Then
    t = volDesc
Else
    t = volName
End If

' Text zusammensetzen aus Label/Beschreibung und ID
t = String.Format("{0} ({1})", t, volID)

' Knoten erzeugen und anhängen
tn = AddNode(TVDirectories.Nodes, t, volID & "\\")

' Icon anhand des Laufwerkstyps auswählen
tn.ImageIndex = drTyp
tn.SelectedImageIndex = drTyp

' Dummy-Knoten anhängen
AddNode(tn.Nodes, ".", ".")

' Für alle Fälle
Catch ex As Exception
    Debug.WriteLine(ex.Message)
End Try

Next

' TreeView ist initialisiert
IsInitialized = True

' Durch Designer vorgegebenen Pfad setzen
Path = SelectedPath

End Sub

```

---

*Listing 200: Initialisierung der TreeView mit Laufwerksbezeichnungen (Forts.)*

AddNode instanziert einen neuen Knoten, allerdings nicht von der Klasse `System.Windows.Forms.TreeNode`, sondern von der abgeleiteten, geschützten Klasse `TreeNode` (Listing 202), die zusätzlich die `Path`-Angabe implementiert. `TreeNode` ist eine innere Klasse von `FolderBrowserA` und somit nur dieser Klasse bzw. deren Ableitungen bekannt. Nach Setzen der Eigenschaften wird der Knoten der als Parameter übergebenen Auflistung hinzugefügt.

---

```
Protected Overridable Function AddNode(ByVal nodeCollection As _
    TreeNodeCollection, ByVal caption As String, _
    ByVal path As String) As TreeNode

    ' Neuen Knoten anlegen
    Dim tn As New TreeNode()

    ' Eigenschaften setzen
    tn.Text = caption
    tn.Path = path

    ' Knoten der angegebenen Auflistung hinzufügen
    nodeCollection.Add(tn)

    ' Referenz des neuen Knotens zurückgeben
    Return tn

End Function
```

---

*Listing 201: Anlegen eines neuen Knotens und Hinzufügen zu einer angegebenen Auflistung*

---

```
Protected Class TreeNode
    Inherits System.Windows.Forms.TreeNode
    Public Path As String
End Class
```

---

*Listing 202: Hilfsklasse zur Erweiterung des TreeNodes um die Variable Path*

Wurde im Designer keine Pfadangabe eingetragen, zeigt das Steuerelement die lokalen Laufwerke des Computers ähnlich der Abbildung 104 an. Versucht der Anwender jetzt, einen der Laufwerksknoten zu öffnen (durch eine Maus- oder Tastaturaktion), wird das Ereignis `BeforeExpand` der `TreeView` (s. Listing 203) ausgelöst. Hier wird überprüft, ob es sich um einen oben beschriebenen Dummy-Knoten handelt. Ist das der Fall, wird mit Hilfe der Methode `CreateSubnodes` dieses Unterverzeichnis neu aufgebaut.

---

```
Private Sub TVDirectories_BeforeExpand(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.TreeViewCancelEventArgs) _
    Handles TVDirectories.BeforeExpand

    ' Prüfen, ob das erste Unterverzeichnis ein Hilfsverzeichnis
    ' ist
    Dim tn As TreeNode = DirectCast(e.Node, TreeNode)
    If DirectCast(tn.Nodes(0), TreeNode).Path = "." Then
        ' Wenn ja, Unterknoten anlegen
        CreateSubnodes(tn)
    End If
End Sub
```

---

*Listing 203: Verzeichnisstruktur untersuchen, wenn der zu öffnende Knoten noch nicht angelegt wurde*

---

Basics

---

Datum/  
Zeit

---

Anwen-  
dungen

---

Zeich-  
nen

---

Bild-  
bearbei-  
tung

---

Win-  
dows  
Forms

---

Controls

---

Propert-  
yGrid

---

Datei-  
system

---

Netz-  
werk

---

System

---

Daten-  
banken

---

XML

---

Wissen-  
schaft

---

Ver-  
schiede-  
nes

In der Methode `CreateSubnodes` (Listing 204) werden zunächst alle vorhandenen Knoten der angegebenen Auflistung, in diesem Fall der Dummy-Knoten, entfernt. Anschließend wird mit Hilfe von `Directory.GetDirectories` das Unterverzeichnis durchlaufen und für jeden Ordner ein neuer Knoten angelegt. Für jeden Ordner wird überprüft, ob er wiederum Unterverzeichnisse besitzt, und ggf. ein Dummy-Knoten angelegt, damit der Knoten in der `TreeView` geöffnet werden kann.

Die Überprüfung auf Unterverzeichnisse ist allerdings nicht unkritisch. Hier kann es zu Fehlermeldungen des Betriebssystems kommen. Daher müssen alle Zugriffe innerhalb eines `Try`-Blockes erfolgen und die möglichen `Exceptions` abgefangen werden. Das Verzeichnis *System Volume Information* wird vollständig von der Untersuchung ausgeschlossen, da Windows bei diesem Verzeichnis erst nach einiger Bedenkzeit einen Fehler meldet.

Allen hinzugefügten Knoten werden die üblichen Symbole für geschlossene bzw. geöffnete Ordner zugewiesen, die ebenfalls in der `ImageList` enthalten sind. Tritt ein Fehler auf, z.B. weil keine CD eingelegt ist, aber der zugehörige Knoten geöffnet werden sollte, dann wird die Methode beendet, ohne dass neue Knoten hinzugefügt worden sind. Da zuvor die Knotenliste gelöscht worden ist, entfernt die `TreeView` auch automatisch das Plus-Zeichen, so dass dieser Knoten nicht mehr geöffnet werden kann.

---

```
Protected Sub CreateSubnodes(ByVal node As TreeNode)

    ' Ggf. Hilfsordner entfernen
    node.Nodes.Clear()

    Try
        Dim subdir As String
        For Each subdir In Directory.GetDirectories(node.Path)
            Dim tn As TreeNode
            ' Neuen Knoten für dieses Verzeichnis erzeugen
            tn = AddNode(node.Nodes, IO.Path.GetFileName(subdir), _
                subdir)

            If Not IO.Path.GetFileName(subdir) = _
                "System Volume Information" Then

                If Directory.GetDirectories(subdir).Length > 0 Then
                    AddNode(tn.Nodes, ".", ".")
                End If
            End If
            tn.ImageIndex = 7
            tn.SelectedIndex = 8
        Next
    Catch ioex As IOException
    Catch ex As Exception

    End Try
End Sub
```

---

Listing 204: Einrichten der Knoten für die Unterverzeichnisse



Auch wenn weitere Unterverzeichnisse ausgewählt werden, wird wieder der gleiche Mechanismus (BeforeExpand, CreateSubnodes) angestoßen, es sei denn, die Unterknoten bestehen bereits in der TreeView, weil sie schon einmal ausgewählt worden sind.

## Die Schnittstelle zur Außenwelt

Das Steuerelement nützt wenig, wenn die Benutzerauswahl nicht abgefragt werden kann. Außerdem soll ja auch per Programm oder Designer ein bestimmter Pfad vorgegeben werden können. Zu diesem Zweck wird das Steuerelement mit der öffentlichen Eigenschaft Path ausgestattet und stellt zusätzlich das Ereignis FolderSelectionChanged zur Verfügung (Listing 205). Zwei geschützte Member-Variablen speichern Pfad bzw. Zustand:

---

```
' Pfadangabe der aktuellen Auswahl
Protected SelectedPath As String

' Merker für die Initialisierung des Pfades über den Designer
Protected IsInitialized As Boolean = False
```

---

IsInitialized ist bis zum Ende von OnLoad False. Eine Zuweisung an die Path-Eigenschaft zu diesem Zeitpunkt führt lediglich dazu, dass der Wert in SelectedPath gespeichert wird. So wird verhindert, dass Knoten erzeugt werden, bevor das Steuerelement sichtbar wird. Diese Situation tritt auf, wenn vom Designer in InitializeComponent eine Zuweisung vorgesehen wurde.

In allen anderen Fällen wird zunächst sichergestellt, dass das angegebene Verzeichnis auch existiert und im Fehlerfall eine Exception ausgelöst. Durch den Aufruf von SetSelection wird der Pfad in der TreeView aufgeklappt und sichtbar gemacht. Zum Lesen der Path-Eigenschaft wird lediglich der in SelectedPath gespeicherte Wert zurückgegeben.

---

```
' Öffentliches Event, das bei Änderung der Auswahl ausgelöst wird
Public Event FolderSelectionChanged(ByVal sender As Object, _
    ByVal e As EventArgs)

' Öffentliche Eigenschaft Path zum Lesen und Auswählen des
' Verzeichnisses
Public Property Path() As String

    Get
        Return SelectedPath
    End Get

    Set(ByVal Value As String)

        ' Null-Werte ignorieren
        If Value Is Nothing Then Return

        ' Wenn das Control noch nicht angezeigt wird, Pfad nur
        ' speichern, keine weiteren Einstellungen möglich
        If Not IsInitialized Then
            SelectedPath = Value
            Return
        End If
```

---

Listing 205: Öffentliche Member Path und FolderSelectionChanged

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Proper-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

---

```

' Wenn's das Verzeichnis nicht gibt, böse antworten ;-)
If Not Directory.Exists(Value) Then Throw New _
    ArgumentException("Path", _
        "Das angegebene Verzeichnis existiert nicht")

' Ansonsten Wert speichern
SelectedPath = Value

' und Verzeichnis in der TreeView auswählen
SetSelection(Value)

End Set

End Property

```

---

*Listing 205: Öffentliche Member Path und FolderSelectionChanged (Forts.)*

Um ein vorgegebenes Verzeichnis in der TreeView darzustellen und auszuwählen, wird der Pfad rekursiv in die einzelnen Verzeichnisebenen zerlegt. Zwei Überladungen von SetSelection, eine rekursive und eine nicht rekursive, teilen sich diese Arbeit. Aufgerufen wird die nicht rekursive Methode. Sie verhindert zunächst, dass das TreeView-Control während der Bearbeitung neu gezeichnet wird, ruft die rekursive Variante auf, steuert die Ereignis-Auslösung, gibt das Neuzeichnen wieder frei und setzt abschließend den Fokus auf das Control. Listing 206 zeigt beide Methoden.

In der rekursiven Methode wird als Erstes der Pfad des übergeordneten Verzeichnisses ermittelt. Existiert er, dann erfolgt die Rekursion für dieses Verzeichnis. Danach ist der Knoten des übergeordneten Verzeichnisses aufgeklappt und verfügt über einen Knoten für jedes Unterverzeichnis. In dieser Knotenliste muss dann der im Parameter path angegebene Ordner gefunden werden. Der gefundene Knoten wird geöffnet und sichtbar gemacht und die Rekursion ist hier beendet.

Wenn das zu betrachtende Verzeichnis kein übergeordnetes Verzeichnis hat, also ein Knoten der Laufwerksebene ist, dann wird in einer ähnlichen Schleife das Laufwerk gesucht und der zugehörige Knoten aufgeklappt.

---

```

Protected Sub SetSelection(ByVal path As String)

    ' Zeichnen der TreeView unterdrücken
    TVDirectories.BeginUpdate()

    ' Event unterdrücken
    IsSelecting = True

    ' Rekursive Überladung aufrufen, tn ByRef übergeben
    Dim tn As TreeNode
    SetSelection(path, tn)

    ' Zeichnen wieder zulassen
    TVDirectories.EndUpdate()

```

---

*Listing 206: Auswahl eines Knotens per Programm*

```

' Event wieder freigeben
IsSelecting = False

' und auslösen
RaiseEvent FolderSelectionChanged(Me, _
    New EventArgs(SelectedPath))

' Fokus auf TreeView setzen
TVDirectories.Focus()

End Sub

' Rekursive Überladung, die den Verzeichnisbaum durcharbeitet
Protected Sub SetSelection(ByVal path As String, _
    ByRef tn As TreeNode)

' Alle Vergleiche in Großbuchstaben
path = path.ToUpper()

' Elternverzeichnis ermitteln
Dim parentDir As String = IO.Path.GetDirectoryName(path)
Dim i As Integer

' Wenn das existiert, den Knoten dieses Verzeichnisses
' ermitteln
If (Not parentDir Is Nothing) Then
    SetSelection(parentDir, tn)

' Alle Knoten durchlaufen und mit dem gesuchten Verzeichnis
' vergleichen
For i = 0 To tn.Nodes.Count - 1
    If DirectCast(tn.Nodes(i), TreeNode).Path.ToUpper() _
        = path Then

' Der richtige Knoten wurde gefunden
tn = DirectCast(tn.Nodes(i), TreeNode)

' Knoten aufklappen und sichtbar machen
tn.Expand()
tn.EnsureVisible()

' Diesen als selektierten Knoten merken
TVDirectories.SelectedNode = tn

' Keine weiteren Knoten dieser Ebene durchlaufen
Return
End If
Next

' Knoten nicht gefunden
Debug.Assert(False)

```

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Propor-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

---

```

    tn = Nothing

Else
    ' Verzeichnis ist auf Laufwerksebene
    ' Alle zuklappen
    TVDDirectories.CollapseAll()

    ' Alle Knoten der Verzeichnisebene durchlaufen
    For i = 0 To TVDDirectories.Nodes.Count - 1
        If DirectCast(TVDDirectories.Nodes(i), TreeNode _
            ).Path.ToUpper().StartsWith(path) Then

            ' Der richtige Knoten wurde gefunden
            tn = DirectCast(TVDDirectories.Nodes(i), TreeNode)

            ' Knoten aufklappen und sichtbar machen
            tn.Expand()
            tn.EnsureVisible()

            ' Diesen als selektierten Knoten merken
            TVDDirectories.SelectedNode = tn

            ' Keine weiteren Knoten dieser Ebene durchlaufen
        Return
    End If
Next
End If
End Sub

```

---

*Listing 206: Auswahl eines Knotens per Programm (Forts.)*

Letztlich bleibt noch die Aufgabe, eine Änderung der Auswahl per Event zu signalisieren. Listing 207 zeigt die eigens hierfür definierte `EventArgs`-Klasse, die den Pfad als Parameter übergibt, Listing 208 die Implementierung des Event-Handlers.

---

```

Public Class EventArgs
    Inherits System.EventArgs

    ' Öffentlich schreibgeschützt: der selektierte Pfad
    Public ReadOnly Path As String

    ' ctor
    Public Sub New(ByVal path As String)
        Me.Path = path
    End Sub

End Class

```

---

*Listing 207: Eigene Event-Klasse des FolderBrowser-Controls*

```

Private Sub TVDirectories_AfterSelect(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.TreeViewEventArgs) _
    Handles TVDirectories.AfterSelect

    ' Event nur auslösen, wenn Knoten nicht durch SetSelection
    ' geöffnet wurde
    If IsSelecting Then Exit Sub

    ' Pfad ermitteln und Event auslösen
    SelectedPath = DirectCast(e.Node, TreeNode).Path
    RaiseEvent FolderSelectionChanged(Me, _
        New EventArgs(SelectedPath))
End Sub

```

Listing 208: Weiterleitung des Select-Ereignisses

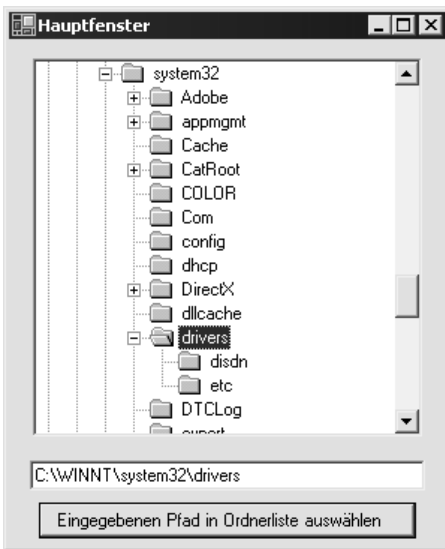


Abbildung 105: Auswahl eines Ordners zur Laufzeit

Auf der CD finden Sie ein Beispielprogramm, das das FolderBrowser-Steuerelement mit einer TextBox und einer Schaltfläche verknüpft. Ein ausgewähltes Verzeichnis wird direkt in der TextBox angezeigt, ein in der TextBox eingetragener Pfad wird nach Drücken der Schaltfläche in der TreeView dargestellt. Der Programmieraufwand ist minimal (s. Listing 209).

```

Private Sub FolderBrowserA1_FolderSelectionChanged( _
    ByVal sender As Object, ByVal e As _
    GuiControls.FolderBrowserA.EventArgs) _
    Handles FolderBrowserA1.FolderSelectionChanged

    ' Neuen Pfad in TextBox anzeigen
    TBSelection.Text = e.Path

```

Listing 209: Einbindung des Steuerelementes auf einem Formular

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Propor-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

---

End Sub

```
Private Sub BTNSelect_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles BTNSelect.Click
```

```
Try
```

```
    ' Eingegebenen Pfad im FolderBrowser-Control anzeigen
```

```
    FolderBrowserA1.Path = TBSelection.Text
```

```
Catch ex As Exception
```

```
    MessageBox.Show(ex.Message)
```

```
End Try
```

```
End Sub
```

---

*Listing 209: Einbindung des Steuerelementes auf einem Formular (Forts.)*

## 129 Ein Windows-Explorer-Steuerelement im Eigenbau

Mit Hilfe des oben beschriebenen Steuerelementes zur Verzeichnisauswahl lässt sich ein neues Steuerelement definieren, das einen ähnlichen Aufbau aufweist wie der Windows-Explorer:

- ▶ Ein Verzeichnisbaum auf der linken Seite (TreeView, FolderBrowserA)
- ▶ Eine Dateiliste auf der rechten Seite (ListView)
- ▶ Ein Splitter dazwischen, um die Fensterbreiten variieren zu können

Die drei Steuerelemente werden auf einem neuen UserControl angelegt und die Docking-Eigenschaften entsprechend gesetzt. Schon steht das Grundgerüst bereit (siehe Abbildung 106).



Abbildung 106: Grundgerüst des Windows-Explorer-Steuerelementes

Wenn der Benutzer ein Verzeichnis auswählt, dann sollen in der ListView alle Dateien dieses Verzeichnisses aufgeführt werden. Auf die Darstellung von Unterordnern, wie sie der richtige Windows-Explorer vorsieht, soll hier der Übersicht halber verzichtet werden. Als Informationen zu den Dateien sollen Name, Größe, Typ und Änderungsdatum angezeigt werden. Selbstverständlich soll auch zu jeder Datei das zugehörige Symbol dargestellt werden.

Ausgangspunkt für das Befüllen der ListView mit den Datei-Informationen ist das Event `FolderSelectionChanged` des Verzeichnisbaums (Listing 210). In einer Schleife werden alle Dateien des ausgewählten Verzeichnisses behandelt. Für jede Datei wird eine eigene `ListViewItem`-Instanz

angelegt. Der Tag-Eigenschaft dieser Instanz wird die Referenz des zugehörigen `FileInformation`-Objektes zugewiesen, um später auf diese Informationen für Sortier- und Maus-Aktionen zurückgreifen zu können.

Zur Ermittlung der Datei-Informationen wird die in Rezept 9.26 ((Icons und Typ einer Datei ermitteln)) beschriebene Klasse `FileInformation` herangezogen. Sie enthält auch die Bitmaps, die zur Darstellung der Datei-Symbole benötigt werden. Zwei Symbole werden bereitgestellt:

1. `SmallIcon` (16 x 16 Pixel)
2. `LargeIcon` (32 x 32 Pixel)

Je nach gewählter Darstellungsart der `ListView` werden entweder die kleinen oder die großen Symbole benötigt. Sie werden jedoch nicht direkt einem `ListViewItem`-Objekt zugewiesen, sondern in `ImageList`-Objekten, hier `ILSmallIcons` und `ILLargeIcons`, gespeichert. Die Verknüpfung zwischen einem `ListView`-Eintrag und den zugehörigen Icons erfolgt über einen Index (`ImageIndex`).

Zur Speicherung der Icons werden somit zusätzlich die beiden `ImageList`-Komponenten benötigt. In Abbildung 106 sind sie bereits mit aufgeführt.

Jedem `ListViewItem`-Objekt, das zunächst nur den Dateinamen als Text erhält, werden weitere untergeordnete `ListViewSubItem`-Objekte hinzugefügt, die die zusätzlichen Informationen in Textform enthalten. Die Icons werden den beiden Bilderlisten hinzugefügt und der Index in `ImageIndex` gespeichert.

Vor Abarbeitung der Schleife, also bevor die `ListView` mit neuen Informationen gefüllt wird, müssen die Listen vollständig gelöscht werden. Beachten Sie hierbei bitte, dass für die `FileInformation`-Objekte explizit `Dispose` aufgerufen werden sollte, damit die Ressourcen für die enthaltenen Bitmaps freigegeben werden können. Diese Aufgabe erledigt die erste Schleife im Listing 210. Da in allen `ListViewItem`-Objekten der Tag-Eigenschaft ein `FileInformation`-Objekt zugewiesen wurde und die Klasse `FileInformation` die Schnittstelle `IDisposable` implementiert, kann in einer `For-Each`-Schleife direkt über einen `TypeCast` auf `IDisposable` die Methode `Dispose` aufgerufen werden.

```
Private Sub FBDirs_FolderSelectionChanged(ByVal sender As Object, _
    ByVal e As FolderBrowserA.EventArgs) _
    Handles FBDirs.FolderSelectionChanged

    ' Alte FileInformation-Objekte entsorgen
    For Each lvi As ListViewItem In LVFiles.Items
        DirectCast(lvi.Tag, IDisposable).Dispose()
    Next

    ' Listen löschen
    LVFiles.Items.Clear()
    ILSmallIcons.Images.Clear()
    ILLargeIcons.Images.Clear()

    ' Fehler bei Dateizugriffen abfangen
    Try
        ' Index für Icons
        Dim i As Integer = 0
```

Listing 210: Befüllen der `ListView` mit den Datei-Informationen

---

```

' Alle Dateien im ausgewählten Verzeichnis
For Each fn As String In Directory.GetFiles(e.Path)

    ' Dateiinformationen abfragen
    Dim finfo As New FileInformation(fn)

    ' Icons in Listen aufnehmen
    ILSmallIcons.Images.Add(finfo.SmallIcon)
    ILLargeIcons.Images.Add(finfo.LargeIcon)

    ' Neue ListView-Zeile für Datei anlegen
    Dim lvi As ListViewItem = LVFiles.Items.Add(finfo.Name)

    ' Verweis auf Dateiinformationen in Tag-Eigenschaft speichern
    lvi.Tag = finfo

    ' Dateigröße in Kilobyte
    lvi.SubItems.Add((finfo.Length + 1023) \ 1024 & " KB")

    ' Dateityp
    lvi.SubItems.Add(finfo.Filetype)

    ' Index der Icons
    lvi.ImageIndex = i

    ' Letzte Änderung
    lvi.SubItems.Add(finfo.LastChanged.ToString())

    i += 1

Next

Catch ex As Exception
    Debug.WriteLine(ex.Message)
End Try
End Sub

```

---

*Listing 210: Befüllen der ListView mit den Datei-Informationen (Forts.)*

In Abbildung 107 sehen Sie ein Beispielfenster, das nur dieses Steuerelement enthält. Für die Eigenschaft `View` der `ListView` wurde hier `DETAILS` gewählt. Diese Ansicht verwendet die kleinen Symbole.

Die Umschaltung der Ansichten erfolgt über ein Kontext-Menü, das der `ListView` zugeordnet wird. Diese bietet die üblichen vier Ansicht-Modi zur Auswahl an:

1. GROßE SYMBOLE
2. KLEINE SYMBOLE
3. LISTE
4. DETAILS



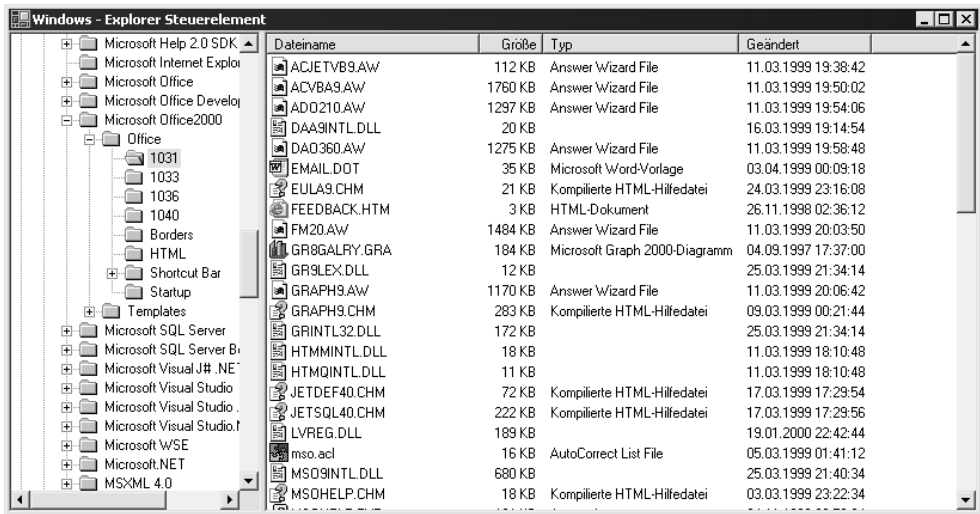


Abbildung 107: Explorer-Steuerelement im Ansichtsmodus Details

Abbildung 108 zeigt die gleiche Liste im Ansichtsmodus LargeIcon. Jedes Kontextmenü verfügt über einen eigenen Ereignis-Handler. Stellvertretend für die anderen drei zeigt Listing 211 den Handler für den Menüpunkt LargeIcon (GROSSE SYMBOLE) und Listing 212 die gemeinsame Methode, die für das Setzen des Punktes zur Visualisierung der aktuellen Auswahl zuständig ist.

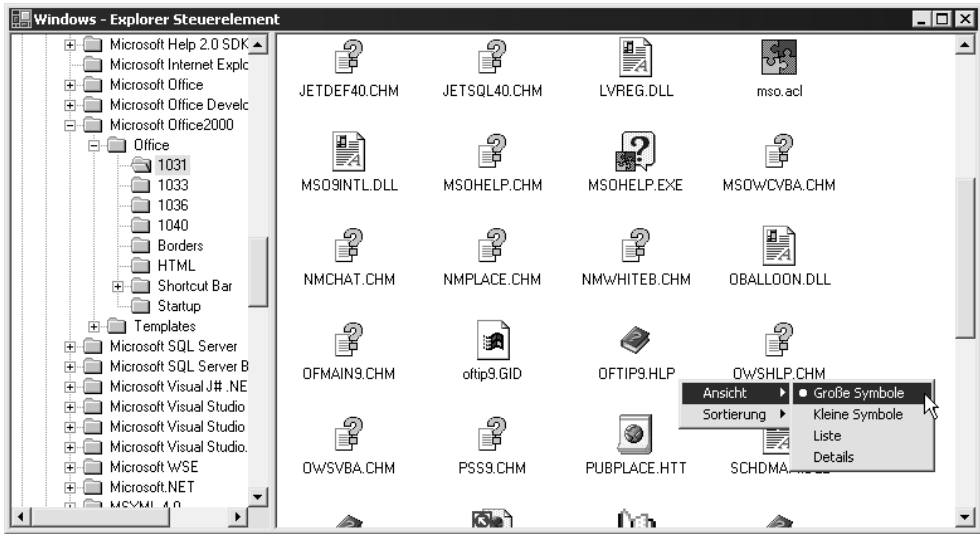


Abbildung 108: Der Ansichtsmodus lässt sich über ein Kontext-Menü umschalten

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Versch-  
iede-  
nes

---

```

Private Sub MenuItem1_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles MenuItem1.Click

    ' Ansicht "Große Symbole" einstellen
    LVFiles.View = View.LargeIcon

    ' Menüanzeigen aktualisieren
    SetViewMenuCheck(CType(sender, MenuItem))

End Sub

```

---

*Listing 211: Umschalten der Ansicht*

---

```

Protected Sub SetViewMenuCheck(ByVal item As MenuItem)
    MenuItem1.Checked = item Is MenuItem1
    MenuItem2.Checked = item Is MenuItem2
    MenuItem3.Checked = item Is MenuItem3
    MenuItem4.Checked = item Is MenuItem4
End Sub

```

---

*Listing 212: Setzen des Auswahlpunktes der Kontext-Menü-Einträge*

## 130 ListView des Explorer-Steuerelementes sortieren

Für den täglichen Umgang mit dem Windows-Explorer sind Sie es sicherlich gewohnt, die angezeigten Informationen nach Dateiname, Typ usw. aufsteigend oder absteigend zu sortieren. In diesem Rezept wird anhand des zuvor beschriebenen Explorer-Steuerelementes erläutert, wie Sie die Informationen, die in einer ListView angezeigt werden, individuell sortieren können.

Meist reichen die eingebauten Sortierfunktionen von Listen-Steuerelementen nicht aus, da sie in der Regel nur alphabetisch sortieren können. Für die Sortierung nach Dateigröße bzw. nach Änderungsdatum ist die alphabetische Reihenfolge aber nicht zu gebrauchen (z.B. kommt 11 in alphabetischer Reihenfolge nach 100). Deswegen muss ein anderer Mechanismus für die Sortierung der Daten vorgesehen werden.

Alle Einträge einer ListView werden in der Auflistung `Items` gespeichert. Diese ist vom Typ `ListViewItemCollection` und referenziert Objekte vom Typ `ListViewItem`. Letztere enthalten die anzuzeigenden Informationen. Im Fall des Explorer-Steuerelementes wurde der `Tag`-Eigenschaft eines jeden `ListViewItem`-Objektes die Referenz des zugehörigen `FileInformation`-Objektes zugewiesen, das seinerseits alle Details der zugehörigen Datei kennt.

Während die angezeigten Informationen in den `SubItems` der ListView alle im Textformat vorliegen und für die Sortierung wenig geeignet sind, kann über die `Tag`-Eigenschaft direkt auf die benötigten Daten des `FileInformation`-Objektes zugegriffen werden. Die Informationen, die für den Sortiervorgang benötigt werden, stehen also bereit.

Die ListView benötigt zum Sortieren aber etwas Hilfe von außen, nämlich ein Objekt, das sie fragen kann, ob ein Eintrag X größer ist als ein Eintrag Y. Ein solches Objekt muss lediglich die Schnittstelle `IComparer` implementieren (Klasse `ExplorerFileSorter`, Listing 213). Diese Schnittstelle wiederum erfordert die Implementierung der Methode `Compare`.

```
Public Class ExplorerFileSorter
    Implements IComparer

    ' Implementierung von IComparer.Compare
    Public Function Compare(ByVal x As Object, ByVal y As Object) _
        As Integer Implements System.Collections.IComparer.Compare
        ...
    End Function

End Class
```

*Listing 213: Diese Klasse soll der ListView beim Vergleich der Einträge helfen*

Da die Dateiliste nach verschiedenen Kriterien sortiert werden soll, werden in der Klasse ExplorerFileSorter die Eigenschaften Criteria und Ascending vorgesehen (Listing 214). So kann festgelegt werden, nach welchem Kriterium und ob auf- oder absteigend sortiert werden soll.

```
' Konstanten für Sortierkriterien
Public Enum Criteria
    ByName
    ByType
    BySize
    ByLastChange
End Enum

' Aufsteigende (True) oder absteigende (False) Sortierung
Protected sortAscending As Boolean = True

' Öffentliche Eigenschaft hierzu
Public Property Ascending() As Boolean
    Get
        Return sortAscending
    End Get
    Set(ByVal Value As Boolean)
        sortAscending = Value
    End Set
End Property

' Einstellung des Sortierkriteriums
Protected sortBy As Criteria = Criteria.ByName
Public Property SortCriteria() As Criteria
    Get
        Return sortBy
    End Get

    Set(ByVal Value As Criteria)
        If sortBy = Value Then
            ' Wenn das Kriterium gleich geblieben ist,
            ' aufsteigend/absteigend invertieren
            sortAscending = Not sortAscending
        End If
    End Set
End Property
```

*Listing 214: Mit den Eigenschaften Criteria und Ascending lässt sich die Sortierreihenfolge festlegen*

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung  
Win-  
dows  
Forms

Controls

Prop-  
ertyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

```

Else
    ' Wenn das Kriterium gewechselt wurde, zunächst
    ' aufsteigend sortieren
    sortAscending = True
End If

' Neues Kriterium speichern
sortBy = Value
End Set
End Property

```

---

*Listing 214: Mit den Eigenschaften Criteria und Ascending lässt sich die Sortierreihenfolge festlegen (Forts.)*

Der eigentliche Sortiervorgang erfolgt, indem die ListView paarweise die Einträge vergleicht und nach irgendeinem Sortieralgorithmus aufsteigend sortiert. Für den paarweisen Vergleich bemüht die ListView das bereitgestellte ExplorerFileSorter-Objekt und ruft die Methode Compare (Listing 215) auf. Als Parameter werden die Referenzen der zu vergleichenden ListViewItem-Objekte (hier x und y) übergeben, als Rückgabewert wird eine Zahl erwartet:

- ▶ 0, wenn gilt  $x = y$
- ▶  $< 0$ , wenn gilt  $x < y$
- ▶  $> 0$ , wenn gilt  $x > y$

Um den Vergleich übersichtlich zu halten werden im ersten Schritt mittels TypeCast zwei Referenzvariablen vom Typ ListViewItem gesetzt. Wieder mittels TypeCast werden zwei Variablen vom Typ FileInformation die Objektreferenzen zugewiesen, die in der jeweiligen Tag-Eigenschaft gespeichert ist. Nun stehen über fil und fi2 die zu vergleichenden Datei-Informationen bereit.

In der folgenden Select Case-Anweisung wird anhand des eingestellten Sortierkriteriums unterschieden, welche Eigenschaft für den Vergleich herangezogen werden soll. Der Vergleich selbst wird an die CompareTo-Methode der entsprechenden Eigenschaft delegiert. Das ist möglich, da die Datentypen der in Frage kommenden Eigenschaften (String, DateTime, Long) alle das Interface IComparable und somit die Methode CompareTo implementieren.

---

```

Public Function Compare(ByVal x As Object, ByVal y As Object) _
    As Integer Implements System.Collections.IComparer.Compare

    ' Typecast auf ListViewItem-Elemente
    Dim lvi1 As ListViewItem = DirectCast(x, ListViewItem)
    Dim lvi2 As ListViewItem = DirectCast(y, ListViewItem)

    ' Dateiinformationen über Tag-Eigenschaft holen
    Dim fil As FileInformation = DirectCast(lvi1.Tag, FileInformation)
    Dim fi2 As FileInformation = DirectCast(lvi2.Tag, FileInformation)

    ' Es kann theoretisch vorkommen, dass mit einem leeren
    ' ListViewItem verglichen wird -> ignorieren
    If fil Is Nothing Or fi2 Is Nothing Then Return 0

    Dim result As Integer

```

---

*Listing 215: Vergleich zweier Dateien nach dem eingestellten Kriterium*

```

' Vergleich anhand des eingestellten Sortierkriteriums an die
' jeweiligen Daten-Objekte delegieren
Select Case sortBy

    ' Stringvergleich der Namen
    Case Criteria.ByName : result = _
        fil.Name.CompareTo(fi2.Name)

    ' Zahlenwertvergleich der Größen
    Case Criteria.BySize : result = _
        fil.Length.CompareTo(fi2.Length)

    ' Stringvergleich der Dateitypen
    Case Criteria.ByType : result = _
        fil.Filetype.CompareTo(fi2.Filetype)

    ' Datumsvergleich der letzten Änderungen
    Case Criteria.ByLastChange : result = _
        fil.LastChanged.CompareTo(fi2.LastChanged)

End Select

' Bei absteigender Sortierung Vorzeichen umkehren
If Not sortAscending Then result = - result

' Ergebnis des Vergleichs zurückgeben
Return result

End Function

```

Listing 215: Vergleich zweier Dateien nach dem eingestellten Kriterium (Forts.)

Das Ergebnis des Vergleichs wird in der Variablen `result` gespeichert. Vor der Rückgabe dieses Wertes muss noch überprüft werden, ob die Liste aufsteigend oder absteigend sortiert werden soll. Im Falle der aufsteigenden Sortierung kann der Wert unverändert weitergegeben werden, im Falle der absteigenden Sortierung muss das Vorzeichen des Ergebnisses invertiert werden, da der Sortieralgorithmus selbst ja nur aufsteigend sortieren kann. Der Vergleichsvorgang ist damit abgeschlossen und die ListView kann das Ergebnis für die Sortierung verwenden.

Abhängig von Sortieralgorithmus (fest eingestellt und in der Dokumentation der ListView-Klasse nicht genannt) und Zusammenstellung der Listenelemente wird die Methode `Compare` so oft aufgerufen, bis alle Elemente sortiert worden sind.

Um die Klasse `ExplorerFileSorter` überhaupt für die Sortierung der ListView nutzen zu können, wird eine Instanz angelegt und in der Variablen `ExpFileSorter` der Klasse `FileBrowser` referenziert:

```
Protected ExpFileSorter As New ExplorerFileSorter
```

Die ListView greift auf die Eigenschaft `ListViewItemSorter` zurück, um das `IComparer`-Objekt aufrufen zu können. Um zu verhindern, dass die Sortierung bei jedem Hinzufügen neuer Dateieinträge erfolgt, wird der Event-Handler aus Listing 210 erweitert (Listing 216). Vor Aufbau der

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Prop-  
ertyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

Listen wird die Eigenschaft `ListViewItemSorter` auf `Nothing` gesetzt, so dass keine Sortierung erfolgt, danach wird ihr wieder die Referenz der `ExplorerFileSorter`-Instanz zugewiesen. Bereits diese Zuweisung führt zum Sortieren der Liste. Ein zusätzlicher Aufruf der Methode `Sort` ist hier nicht nötig.

```
Private Sub FBDirs_FolderSelectionChanged(ByVal sender As Object, _
    ByVal e As FolderBrowserA.EventArgs) _
    Handles FBDirs.FolderSelectionChanged

    ' Keine Sortierung, während die Liste aufgebaut wird
    LVFiles.ListViewItemSorter = Nothing

    ... Aufbau der Listen

    ' Sortierung freigeben und durchführen
    LVFiles.ListViewItemSorter = ExpFileSorter

    ...

End Sub
```

Listing 216: Zuweisen des `ExpFileSorter`-Objektes

## Umschalten der Sortierkriterien

Dem Anwender werden zwei Möglichkeiten angeboten, die Sortierung der Dateiliste nach seinen Wünschen anzupassen:

1. Über ein Kontext-Menü
2. Durch Klick auf den jeweiligen Spaltenkopf (nur in der Ansicht DETAILS)

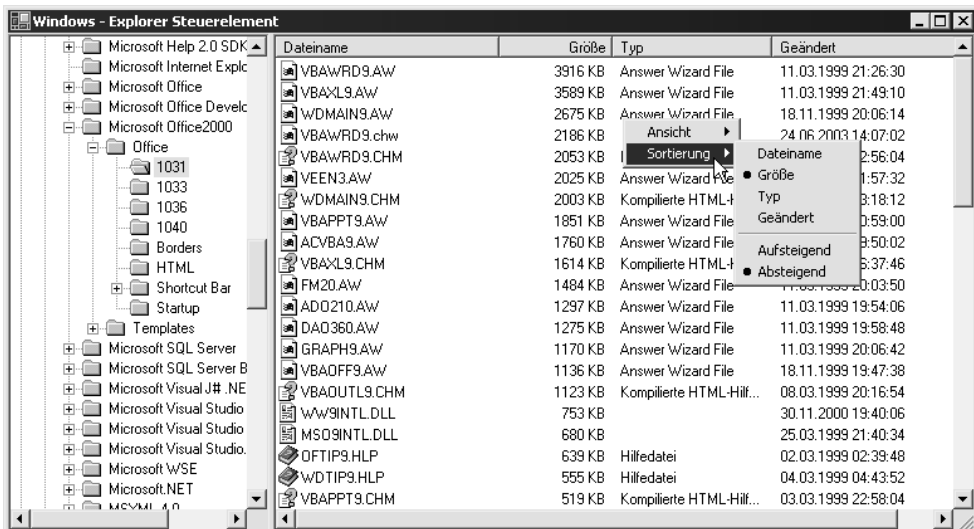


Abbildung 109: Sortieren der Dateiliste

Über das Kontextmenü lassen sich Sortierkriterium und Richtung festlegen (siehe Abbildung 109). Bei einem Mausklick auf einen Spaltenkopf wird die Liste nach den Werten dieser Spalte sortiert. Wurde zuvor nach einem anderen Kriterium sortiert, dann erfolgt die Sortierung aufsteigend, ansonsten wird zwischen auf- und absteigender Sortierung umgeschaltet.

Wieder stellvertretend für die anderen Kriterien wird in Listing 217 die Implementierung des Ereignis-Handlers für das Kontext-Menü DATEINAME dargestellt. Die Listings der übrigen Handler finden Sie auf der Buch-CD. Die Handler legen das Kriterium fest und rufen die Methode `SetOrderMenuCheckAndSort` auf (Listing 218).

---

```
Private Sub MNOrderBy1_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles MNOrderBy1.Click

    ' Sortierkriterium setzen
    ExpFileSorter.SortCriteria = ExplorerFileSorter.Criteria.ByName

    ' ListView sortieren und Menüs aktualisieren
    SetOrderMenuCheckAndSort()

End Sub
```

---

Listing 217: Auswahl des Sortierkriteriums über ein Kontext-Menü

---

```
Protected Sub SetOrderMenuCheckAndSort()

    ' Sortierung durchführen
    LVFiles.Sort()

    ' Punkte setzen
    MNOrderBy1.Checked = _
        ExpFileSorter.SortCriteria = ExplorerFileSorter.Criteria.ByName
    MNOrderBy2.Checked = _
        ExpFileSorter.SortCriteria = ExplorerFileSorter.Criteria.BySize
    MNOrderBy3.Checked = _
        ExpFileSorter.SortCriteria = ExplorerFileSorter.Criteria.ByType
    MNOrderBy4.Checked = ExpFileSorter.SortCriteria = _
        ExplorerFileSorter.Criteria.ByLastChange

    MNOrderAsc.Checked = ExpFileSorter.Ascending
    MNOrderDesc.Checked = Not ExpFileSorter.Ascending

End Sub
```

---

Listing 218: Liste sortieren und Menüs aktualisieren

In `SetOrderMenuCheckAndSort` wird die `ListView` zum Sortieren der Einträge aufgefordert und die Punkte, die die aktuelle Auswahl im Kontext-Menü markieren, gesetzt.

Etwas aufwändiger ist die Sortierung über Mausklicks auf die Spaltenköpfe. Da die Reihenfolge der Spalten im Designer geändert werden kann und sich so deren Index auch ändern kann, muss eine sichere Zuordnung zwischen Spalte und Sortierkriterium vorgesehen werden. Hierzu wird die Definition der Klasse `ColumnHeader` um eine Eigenschaft für das Sortierkriterium erweitert

(Listing 219). Den vier Member-Variablen vom Typ `ColumnHeader` muss von Hand der neue Typ zugewiesen werden:

---

```
Friend WithEvents CHName As ExplorerColumnHeader
Friend WithEvents CHSize As ExplorerColumnHeader
Friend WithEvents CHChanged As ExplorerColumnHeader
Friend WithEvents CHType As ExplorerColumnHeader
```

---

Dann zeigt der Designer die in Abbildung 110 abgebildete Erweiterung. So lässt sich bereits zur Entwurfszeit festlegen, welche Spalte welchem Sortierkriterium zugeordnet wird. Werden die Spalten später verschoben, hat das keinen Einfluss auf diese Zuordnung.

---

```
Public Class ExplorerColumnHeader
    Inherits ColumnHeader

    ' Speicherung des zugehörigen Sortierkriteriums
    Protected sortBy As ExplorerFileSorter.Criteria

    ' Öffentliche Eigenschaft hierzu (für Verwendung im PropertyGrid)
    Public Property SortCriteria() As ExplorerFileSorter.Criteria
        Get
            Return sortBy
        End Get
        Set(ByVal Value As ExplorerFileSorter.Criteria)
            sortBy = Value
        End Set
    End Property

End Class
```

---

Listing 219: `ExplorerColumnHeader` erweitert die Klasse `ColumnHeader` um das Sortierkriterium

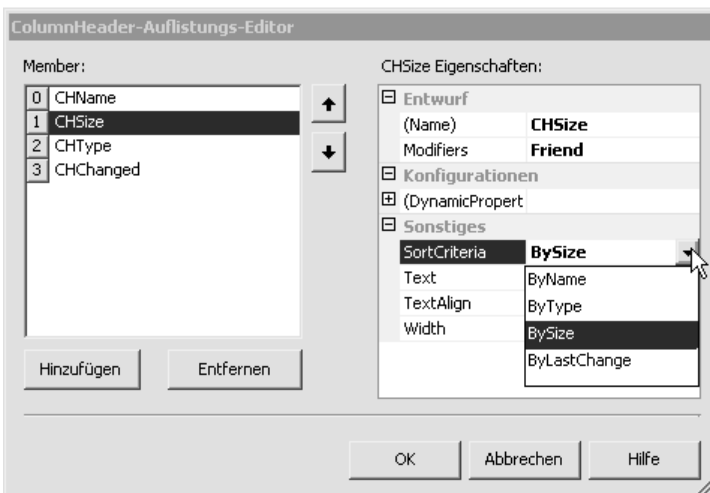


Abbildung 110: Feste Zuordnung des Sortierkriteriums zu einer Spalte im Designer



Bei einem Klick auf einen Spaltenkopf wird der Event-Handler `LVFiles_ColumnClick` (Listing 220) aufgerufen. Übergeben wird der Index der Spalte, der zur Ermittlung des `ExplorerColumnHeader`-Objektes herangezogen wird. Das Sortierkriterium wird festgelegt und anschließend wiederum `SetOrderMenuCheckAndSort` für die Sortierung und Aktualisierung der Kontext-Menüs aufgerufen.

```
Private Sub LVFiles_ColumnClick(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.ColumnClickEventArgs) _
    Handles LVFiles.ColumnClick

    ' Erweitertes columnHeader-Objekt ermitteln
    Dim ch As ExplorerColumnHeader = _
        DirectCast(LVFiles.Columns(e.Column), ExplorerColumnHeader)

    ' Sortierkriterium setzen
    ExpFileSorter.SortCriteria = ch.SortCriteria

    ' ListView sortieren und Menüs aktualisieren
    SetOrderMenuCheckAndSort()

End Sub
```

Listing 220: Sortierung bei Klick auf einen Spaltenkopf

## 131 FolderBrowser-Steuerelement mit zusätzlichen CheckBoxen zum Aufbau von Verzeichnislisten

In Rezept 7.13 ((Steuerelement für Verzeichnisauswahl)) wurde beschrieben, wie ein Steuerelement aufgebaut wird, das die Verzeichnisstruktur eines Computers anzeigen kann. Dieses Steuerelement soll nun so erweitert werden, dass der Anwender individuell Verzeichnisse für eine weitere Bearbeitung markieren kann. Hierzu soll für jeden Verzeichniseintrag eine `CheckBox` angezeigt werden.

Der Anwender kann nun durch Setzen der entsprechenden `CheckBoxen` Verzeichnisse auswählen, die dann z.B. kopiert, archiviert, komprimiert oder gelöscht werden sollen oder welche Aufgabe auch immer Sie damit verbinden möchten. Die Auswahllogik wird folgendermaßen definiert (vergleiche Abbildung 111):

1. Durch Setzen eines `CheckBox`-Häkchens werden das dazugehörige Verzeichnis sowie alle Unterverzeichnisse ausgewählt
2. Durch Entfernen eines `CheckBox`-Häkchens werden die Auswahl für das dazugehörige Verzeichnis sowie für alle Unterverzeichnisse zurückgenommen
3. Bei Zustandsänderung einer `CheckBox` wird die `CheckBox` des übergeordneten Verzeichnisses P wie folgt geändert:
  1. Sind alle Unterverzeichnisse von P ausgewählt, dann erhält P ein schwarzes Häkchen
  2. Ist kein Unterverzeichnis von P ausgewählt, wird das Häkchen entfernt
  3. In allen anderen Fällen wird ein graues Häkchen gesetzt
4. Regel 1, 2 und 3 werden rekursiv angewendet, bis die unterste bzw. oberste Verzeichnisebene erreicht wird

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Propert-  
yGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes



Abbildung 111: Erweiterung der Verzeichnisauswahl mit CheckBoxen

Für die Realisierung der genannten Regeln werden CheckBoxen benötigt, die drei Zustände annehmen können. Daher reicht die vom TreeView-Steuerelement bereitgestellte Möglichkeit, CheckBoxen über die Eigenschaft `CheckBoxes` anzuzeigen, nicht aus. Die von der TreeView angebotenen CheckBoxen unterstützen nur zwei Zustände.

### Platzieren zusätzlicher Steuerelemente auf einem TreeView-Control

Stattdessen wird für jeden Verzeichniseintrag eine gewöhnliche CheckBox auf der TreeView platziert. Ein TreeView-Control kann beliebige Steuerelemente aufnehmen, sie müssen nur der `Controls`-Auflistung der TreeView zugeordnet werden. Im `Paint`-Ereignis werden die CheckBoxen positioniert und angezeigt.

Um eine CheckBox fest an einen Verzeichniseintrag binden zu können, wird die Klasse `FolderBrowserA.TreeNode` erweitert (Listing 221). Als zusätzliches Element wird die Referenz `BoundCheckBox` vorgesehen. Im Konstruktor werden Pfad und Anzeigetext gespeichert. Damit die CheckBox zwischen dem Verzeichnissymbol und dem Text angeordnet werden kann, ohne einen Teil des Textes zu verdecken, werden dem Text einige Leerzeichen vorangestellt.

Der Referenzvariablen `BoundCheckBox` wird eine neu angelegte CheckBox-Instanz zugewiesen. Die CheckBox selbst soll keinen Text darstellen, sondern lediglich die quadratische Schaltfläche anzeigen. Um später bei gegebener CheckBox den zugehörigen Knoten ermitteln zu können, wird die Referenz des Knotens in der Eigenschaft `Tag` gespeichert. Letztlich wird die neue CheckBox der `Controls`-Auflistung des zugrunde liegenden TreeView-Controls hinzugefügt.

---

```
Friend Class TreeNodeExt
    Inherits FolderBrowserA.TreeNode

    ' Gebundene CheckBox für diesen Knoten
    Public BoundCheckBox As CheckBox

    ' Konstruktor legt bereits die CheckBox an
    Public Sub New(ByVal tv As TreeView, ByVal text As String, _
        ByVal path As String)

        ' Verzeichnispfad
        Me.Path = path

        ' Etwas Platz (Leerzeichen) für die CheckBox einräumen
        Me.Text = "    " & text

        ' Neue CheckBox anlegen
        ' Die CheckBox zeigt nur die Schaltfläche an und
        ' enthält keinen Text
```

---

Listing 221: Erweiterung der Klasse `FolderBrowserA.TreeNode` um eine gebundene CheckBox

```

BoundCheckbox = New CheckBox
BoundCheckbox.FlatStyle = FlatStyle.System

' Die Tag-Eigenschaft der CheckBox verweist auf den
' zugehörigen Knoten
BoundCheckbox.Tag = Me

' CheckBox der Controls-Auflistung der TreeView hinzufügen
tv.Controls.Add(BoundCheckbox)

End Sub
End Class

```

Listing 221: Erweiterung der Klasse FolderBrowserA.TreeNode um eine gebundene CheckBox (Forts.)

Neue Verzeichnisknoten werden mit Hilfe der Methode `AddNode` hinzugefügt. Diese wird für die Erweiterung überschrieben (Listing 222). Ein neuer Knoten vom Typ `TreeNodeExt` wird der als Parameter übergebenen Auflistung hinzugefügt. Das Häkchen der zugehörigen CheckBox wird abhängig vom Zustand des Elternknotens gesetzt. Zwei Ereignis-Handler (`BoundCheckbox_CheckStateChanged` und `BoundCheckbox_Click`, siehe unten) werden an die CheckBox gebunden, um auf Auswahl und Zustandsänderung reagieren zu können.

```

Protected Overrides Function AddNode(ByVal nodeCollection As _
    System.Windows.Forms.TreeNodeCollection, _
    ByVal caption As String, ByVal path As String) As TreeNode

    ' neuen Knoten mit CheckBox anlegen
    Dim tn As New TreeNodeExt(TVDirectories, caption, path)

    ' Knoten an übergebene Liste anhängen
    nodeCollection.Add(tn)

    ' Wenn der Elternknoten einen Haken hat, dann auch bei diesem
    ' Knoten einen setzen
    Dim p As TreeNodeExt = DirectCast(tn.Parent, TreeNodeExt)
    If Not p Is Nothing Then
        tn.BoundCheckbox.Checked = p.BoundCheckbox.Checked
    End If

    ' Ereignishandler binden
    AddHandler tn.BoundCheckbox.CheckStateChanged, _
        New EventHandler(AddressOf BoundCheckbox_CheckStateChanged)

    AddHandler tn.BoundCheckbox.Click, New EventHandler(_
        AddressOf BoundCheckbox_Click)

    Return tn

End Function

```

Listing 222: Die überschriebene Methode `AddNode` berücksichtigt auch die CheckBox

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Propert-  
yGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

## Positionieren und Anzeigen der CheckBoxen

Bislang werden die CheckBoxen zwar angelegt, aber noch nicht angezeigt. Die Anzeige erfolgt im Paint-Ereignis der TreeView. Leider kann dieses Ereignis nur über einen Umweg abgefangen werden (siehe Rezept 7.12 ((Abfangen von Windows-Nachrichten))). Hierfür wird eine Instanz der von NativeWindow abgeleiteten Klasse NAWTreeView (Listing 224) angelegt (siehe Listing 223), deren Referenz in der Membervariablen NAWTV gespeichert wird.

---

```
Private Sub TVDirectories_HandleCreated(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles TVDirectories.HandleCreated

    ' NativeWindow-Instanz anlegen und binden
    NAWTV = New NAWTreeView(TVDirectories)
End Sub

Private Sub TVDirectories_HandleDestroyed(ByVal sender As Object, _
    ByVal e As System.EventArgs) _
    Handles TVDirectories.HandleDestroyed

    ' Handle wieder freigeben
    NAWTV.ReleaseHandle()
End Sub
```

---

Listing 223: Binden einer NativeWindow-Instanz für den Zugriff auf die WndProc-Methode

---

```
Protected Class NAWTreeView
    Inherits NativeWindow

    ' Referenz der zugehörigen TreeView
    Private tvDirectories As TreeView

    ' Konstruktor bindet die NativeWindow-Instanz an die TreeView
    Public Sub New(ByVal tvdir As TreeView)
        tvDirectories = tvdir
        AssignHandle(tvdir.Handle)
    End Sub

    Protected Overrides Sub WndProc( _
        ByRef m As System.Windows.Forms.Message)

        ' Methode der Basisklasse aufrufen (wichtig!)
        MyBase.WndProc(m)

        Dim r As Rectangle
        ' WM_PAINT?
        If m.Msg = &HF Then
            For Each c As Control In tvDirectories.Controls
                If TypeOf c Is CheckBox Then
                    Dim cb As CheckBox = DirectCast(c, CheckBox)

                    ' Knoten ermitteln
```

---

Listing 224: Paint-Ereignis der TreeView über eine NativeWindow-Instanz abfangen

```

Dim tn As TreeNodeExt = DirectCast(cb.Tag, TreeNodeExt)

' Nur wenn der Knoten sichtbar ist
' die Checkbox positionieren
If tn.IsVisible Then
    ' Position und Größe berechnen
    r = tn.Bounds
    cb.SetBounds(r.X, r.Y + 1, cb.Height, r.Height)
End If

' CheckBox ist nur sichtbar, wenn auch der Knoten
' sichtbar ist
cb.Visible = tn.IsVisible

End If
Next
End If
End Sub
End Class

```

Listing 224: Paint-Ereignis der TreeView über eine NativeWindow-Instanz abfangen (Forts.)

In der WndProc-Methode von NAWTreeView wird nach Aufruf der gleichnamigen Methode der Basisklasse überprüft, ob es sich um eine WM\_PAINT-Nachricht (entspricht dem Paint-Ereignis anderer Steuerelemente) handelt. Ist das der Fall, dann wird in einer Schleife die gesamte Controls-Auflistung der TreeView durchlaufen. Für jede gefundene CheckBox wird der zugeordnete Knoten ermittelt. Ist dieser sichtbar, dann wird über dessen Eigenschaft Bounds das umschließende Rechteck bestimmt und anschließend mit SetBounds Größe und Position der CheckBox gesetzt. Die Visible-Eigenschaft der CheckBox wird entsprechend der Sichtbarkeit des Knotens gesetzt. Diese ergibt sich aus dem Wert der Eigenschaft IsVisible. Wenn der Knoten nicht sichtbar ist, weil er z.B. durch Scrollen aus dem Anzeigebereich der TreeView herausgeschoben wurde, dann muss auch die zugehörige CheckBox nicht sichtbar sein.

Immer wenn die TreeView neu gezeichnet wird, werden auch die CheckBoxen positioniert. So wird sichergestellt, dass eine CheckBox immer neben ihrem zugehörigen Verzeichnisknoten sichtbar ist.

### Setzen der Häkchen in über- und untergeordneten Knoten

Der Ereignis-Handler BoundCheckbox\_CheckStateChanged wird immer dann aufgerufen, wenn sich der Zustand einer CheckBox geändert hat, auch dann, wenn die Zustandsänderung nicht durch eine Mausaktion, sondern durch programmatisches Setzen der Eigenschaft CheckState erfolgt. Wenn also im Ereignis-Handler die CheckBoxen des übergeordneten bzw. der untergeordneten Verzeichnisse geändert werden, dann wird auch für diese Änderungen jeweils ein Event ausgelöst. Diese Gegebenheit wird genutzt, um rekursiv alle erforderlichen Änderungen vorzunehmen. Zur Steuerung werden zwei Membervariablen definiert:

```

Protected CheckingChildren As Integer = 0
Protected CheckingParents As Integer = 0

```

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Propert-  
yGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes



Anhand der beiden Zähler kann somit bestimmt werden, in welchem Zustand sich die Rekursion befindet. Ohne diese Unterscheidung würde es schnell zu Endlosschleifen kommen. Den zugehörigen Code sehen Sie in Listing 225.

Mit Hilfe zweier TypeCasts werden die auslösende CheckBox und der dazugehörige Knoten ermittelt. Im ersten Schritt (`CheckingChildren = 0`) wird der Elternknoten, sofern er existiert, betrachtet. Es wird überprüft, wie viele Unterverzeichnisse des Elternknotens ein gesetztes Häkchen aufweisen. Ist bei allen Unterverzeichnissen das Häkchen gesetzt, erhält auch der Elternknoten ein schwarzes Häkchen (`CheckState.Checked`). Ist es bei keinem Unterverzeichnis gesetzt, wird auch das Häkchen des Elternknotens gelöscht (`CheckState.Unchecked`). In allen anderen Fällen wird ein graues Häkchen (`CheckState.Indeterminate`) eingetragen. Die Änderung der `CheckState`-Eigenschaft führt zum rekursiven Aufruf des Ereignis-Handlers (s.o.).

Im zweiten Schritt (`CheckingParents = 0`) wird den CheckBoxen der Unterverzeichnisse der gleiche Zustand zugewiesen, wie ihn der Anwender in der aktiven CheckBox eingestellt hat (nur `CheckState.Checked` oder `CheckState.Unchecked` sind möglich). Auch hier führt die Änderung wieder zur Rekursion (s.o.).

Abschließend (alle rekursiven Aufrufe sind beendet, sowohl `CheckingChildren` als auch `CheckingParents` haben den Wert 0) wird ein Ereignis (`DirectoryChecked` oder `DirectoryUnchecked`) ausgelöst, das das Anwendungsprogramm über die Änderung benachrichtigt.

```
Private Sub BoundCheckbox_CheckStateChanged( _
    ByVal sender As Object, ByVal e As System.EventArgs)

    ' Welche CheckBox hat das Ereignis ausgelöst?
    Dim cb As CheckBox = DirectCast(sender, CheckBox)

    ' Welcher Knoten gehört dazu?
    Dim tn As TreeNodeExt = DirectCast(cb.Tag, TreeNodeExt)

    ' Nicht durchlaufen, wenn gerade Unterverzeichnisse rekursiv
    ' abgearbeitet werden
    If CheckingChildren = 0 Then
        Dim pn As TreeNodeExt = DirectCast(tn.Parent, TreeNodeExt)
        If Not pn Is Nothing Then
            Dim noneChecked As Boolean = True
            Dim allChecked As Boolean = True

            ' Alle Knoten auf der Ebene (Geschwister von tn) durchlaufen
            For Each tnSibbling As TreeNodeExt In pn.Nodes
                Select Case tnSibbling.BoundCheckbox.CheckState
                    Case CheckState.Checked
                        noneChecked = False
                    Case CheckState.Indeterminate
                        allChecked = False
                        noneChecked = False
                    Case CheckState.Unchecked
                        allChecked = False
                End Select
            End For
        End If
    End If
```

Listing 225: `BoundCheckbox_CheckStateChanged` wird rekursiv abgearbeitet

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

```

    If Not allChecked And Not noneChecked Then Exit For
Next

    ' Das Setzen des CheckStates löst einen Event aus
    ' Daher wird der Merker inkrementiert, um die Rekursion
    ' einzuschränken
    CheckingParents += 1

    If allChecked Then
        ' Alle Unterverzeichnisse ausgewählt?
        pn.BoundCheckbox.CheckState = CheckState.Checked

    ElseIf noneChecked Then
        ' Kein Unterverzeichnis ausgewählt?
        pn.BoundCheckbox.CheckState = CheckState.Unchecked
    Else
        ' Einige, aber nicht alle ausgewählt
        pn.BoundCheckbox.CheckState = CheckState.Indeterminate
    End If

    ' Merker dekrementieren
    CheckingParents -= 1

End If
End If

' Nicht durchlaufen, wenn gerade Elternverzeichnisse abgearbeitet
' werden
If CheckingParents = 0 Then
    For Each child As TreeNodeExt In tn.Nodes
        CheckingChildren += 1
        '
        child.BoundCheckbox.CheckState = cb.CheckState
        CheckingChildren -= 1
    Next
End If

' Nur, wenn Sender die CheckBox ist, auf die der Anwender geklickt
' hat. Nicht durchlaufen bei rekursiven Aufrufen für Eltern-
' oder Unterverzeichnisse

If CheckingChildren + CheckingParents = 0 Then

    ' Ereignis auslösen
    If cb.Checked Then
        RaiseEvent DirectoryChecked(Me, _
            New SelectionEventArgs(tn.Path))
    Else
        RaiseEvent DirectoryUnchecked(Me, _
            New SelectionEventArgs(tn.Path))
    End If

```

---



```
End If

End Sub
```

Listing 225: BoundCheckbox\_CheckStateChanged wird rekursiv abgearbeitet (Forts.)

Wird eine CheckBox durch eine Benutzeraktion gesetzt, dann soll automatisch der zugehörige Knoten der TreeView aktiviert werden, um eine bessere visuelle Rückmeldung zu erreichen. Hierzu wird im Ereignis-Handler BoundCheckbox\_Click (Listing 226) der TreeView der Fokus zugewiesen und der entsprechende Knoten selektiert.

```
Private Sub BoundCheckbox_Click(ByVal sender As Object, _
    ByVal e As System.EventArgs)

    ' Knoten selektieren
    TVDirectories.SelectedNode = DirectCast(DirectCast( _
        sender, CheckBox).Tag, TreeNodeExt)

    ' Fokus auf TreeView setzen
    TVDirectories.Focus()

End Sub
```

Listing 226: Markieren des Verzeichnisses nach Klick auf die CheckBox

Auflisten der ausgewählten Verzeichnisse

Um die ausgewählten Verzeichnisse weiter bearbeiten zu können, kann das Anwendungsprogramm eine Liste in Form eines String-Arrays abfragen. Hierzu steht die Methode GetSelectedDirectories zur Verfügung. Folgende Annahme wird getroffen:

Das Anwendungsprogramm durchläuft die Liste und bearbeitet jedes aufgeführte Verzeichnis und alle seine Unterverzeichnisse. Ein Verzeichnis soll nur einmal bearbeitet werden.

Daraus folgt, dass ein Verzeichnis nur genau dann in die Liste aufgenommen wird, wenn alle seine Unterverzeichnisse ausgewählt sind. Die Unterverzeichnisse selbst werden nicht mit aufgenommen. Der Verzeichnisbaum muss somit beginnend mit der obersten Ebene rekursiv durchlaufen werden.

Die öffentliche parameterlose Methode GetSelectedDirectories (Listing 227 legt eine leere Hilfsliste in Form eines ArrayList-Objektes an und übergibt diese an die geschützte rekursive Überladung, die die Liste entsprechend füllt. Anschließend wird aus der Liste ein String-Array erzeugt und zurückgegeben.

Im rekursiven Zweig wird die übergebene Knotenliste (beim ersten Aufruf ist es die Liste der Stamm-Knoten, also der Verzeichnisse) durchlaufen. Ist die zugehörige CheckBox gesetzt (CheckBox.Checked), dann wird der Knoten in die Ergebnisliste aufgenommen. Seine Unterverzeichnisse werden nicht weiter betrachtet. Hat die zugehörige CheckBox den Zustand CheckBox.Indeterminate, dann wird zwar der Knoten selbst nicht mit aufgenommen, die Methode aber für die Liste der untergeordneten Verzeichnisknoten rekursiv aufgerufen. Knoten, deren CheckBox kein Häkchen aufweist, sind nicht ausgewählt und werden nicht weiter beachtet.

|                 |
|-----------------|
| Basics          |
| Datum/Zeit      |
| Anwendungen     |
| Zeichen         |
| Bildbearbeitung |
| Windows Forms   |
| Controls        |
| PropertyGrid    |
| Dateisystem     |
| Netzwerk        |
| System          |
| Datenbanken     |
| XML             |
| Wissenschaft    |
| Verchiedenes    |

---

```
Public Overridable Function GetSelectedDirectories() As String()
```

```
    ' Hilfsliste
```

```
    Dim list As New ArrayList
```

```
    ' Aufruf der rekursiven Überladung
```

```
    GetSelectedDirectories(TVDirectories.Nodes, list)
```

```
    ' Rückgabe eines String-Arrays
```

```
    Return DirectCast(list.ToArray(GetType(String)), String())
```

```
End Function
```

```
Protected Overridable Function GetSelectedDirectories( _  
    ByVal nodelist As TreeNodeCollection, ByVal list As ArrayList) _  
    As String()
```

```
    ' Die übergebene Knotenliste durchlaufen
```

```
    For Each tn As TreeNodeExt In nodelist
```

```
        If tn.BoundCheckbox.CheckState = CheckState.Checked Then
```

```
            ' Wenn der Unterknoten ausgewählt ist, nur diesen in die
```

```
            ' Liste aufnehmen
```

```
            list.Add(tn.Path)
```

```
        ElseIf _
```

```
            tn.BoundCheckbox.CheckState = CheckState.Indeterminate Then
```

```
                ' Wenn nur einige Unterknoten dieses Unterknotens ausgewählt
```

```
                ' sind, rekursiver Aufruf für diesen Unterknoten
```

```
                GetSelectedDirectories(tn.Nodes, list)
```

```
        End If
```

```
    Next
```

```
End Function
```

---

Listing 227: Generieren einer Liste, die die ausgewählten Verzeichnisse auflistet

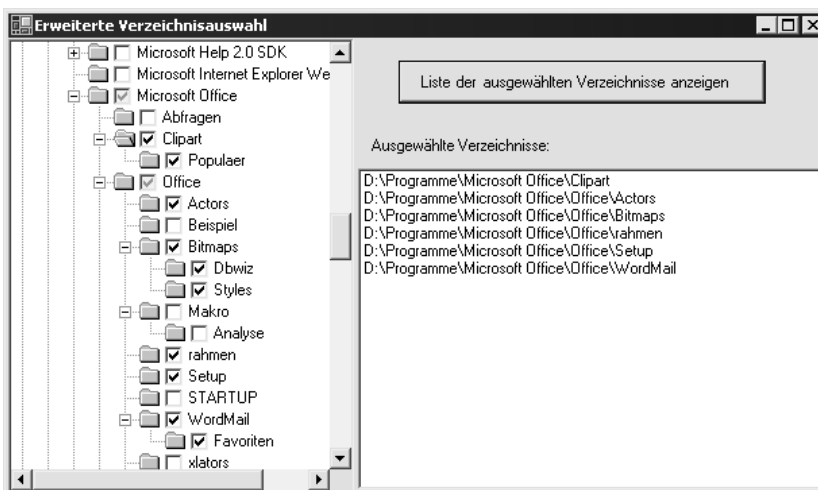


Abbildung 113: Ausgabe der ausgewählten Verzeichnisse im Anwendungsprogramm

Abbildung 113 zeigt eine Beispielanwendung für das Steuerelement in einer Windows-Form. Durch Betätigen der Schaltfläche wird eine ListBox mit den bereitgestellten Informationen gefüllt (Listing 228).

```
' ListBox leeren
LBSelectedDirectories.Items.Clear()

' Array mit ausgewählten Verzeichnissen abfragen
Dim s() As String = FolderBrowserB1.GetSelectedDirectories()

' Verzeichnisse in ListBox anzeigen
LBSelectedDirectories.Items.AddRange(s)
```

Listing 228: Abfrage der ausgewählten Verzeichnisse

## 132 Benutzerdefinierte Steuerelemente mit nicht rechteckigem Umriss

Benutzerdefinierte Steuerelemente bedecken im Normalfall eine rechteckige Fläche. Alle Bereiche, die nicht durch eigene Zeichenoperationen in `OnPaint` übermalt oder durch andere Steuerelemente verborgen werden, werden mit der Hintergrundfarbe gefüllt. Um dieses Verhalten zu ändern, sind mehrere Schritte notwendig.

Die naheliegendste Idee wäre, das Zeichnen des Hintergrundes ganz zu unterdrücken. Leider wird dann auch der Hintergrund des Containers, auf dem das Steuerelement liegt, nicht mehr gezeichnet. Daher ist die einfachste Lösung, als Hintergrundfarbe `Color.Transparent` einzustellen.

Damit eine transparente Hintergrundfarbe überhaupt möglich ist, muss der Stil `SupportsTransparentBackColor` gesetzt werden. Insgesamt empfehlen sich die folgenden Maßnahmen im Konstruktor des Steuerelementes:

```
' Zeichnen des Hintergrundes für Transparenzeffekt erzwingen
SetStyle(Windows.Forms.ControlStyles.Opaque, False)

' Transparenz ermöglichen
SetStyle(Windows.Forms.ControlStyles.SupportsTransparentBackColor, _
    True)

' Transparenten Hintergrund einstellen
Me.BackColor = Color.Transparent

' Flackern bei Größenänderung minimieren
SetStyle(Windows.Forms.ControlStyles.UserPaint, True)
SetStyle(Windows.Forms.ControlStyles.AllPaintingInWmPaint, True)
SetStyle(Windows.Forms.ControlStyles.DoubleBuffer, True)
```

Mit dieser Einstellung wird der Hintergrund des Steuerelementes, also alle Flächen, die nicht anderweitig verdeckt werden, transparent. Er wird durch den Hintergrund des Containers ersetzt. Listing 229 zeigt ein Beispiel für ein kreisförmiges Steuerelement. In `OnResize` werden Mittelpunkt und Radius errechnet, in `OnPaint` der Kreis gezeichnet. Wie das Steuerelement auf einem Fenster mit Hintergrundbild im Designer dargestellt wird, sehen Sie in Abbildung 114. Zur Laufzeit ist die tatsächliche Größe nicht mehr zu erkennen (Abbildung 115).

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Propor-  
tGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

```
Protected Overrides Sub OnPaint(ByVal e As System.Windows.Forms.PaintEventArgs)
```

```
    Dim g As Graphics = e.Graphics
```

```
    ' Koordinatenursprung in den Mittelpunkt legen
```

```
    g.TranslateTransform(Center.X, Center.Y)
```

```
    ' Kreis zeichnen
```

```
    Dim r As New Rectangle(-Radius, -Radius, Radius * 2, Radius * 2)
```

```
    g.FillEllipse(Brushes.White, r)
```

```
End Sub
```

```
Protected Overrides Sub OnResize(ByVal e As System.EventArgs)
```

```
    ' Mittelpunkt berechnen
```

```
    Center = New Point(Width \ 2, Height \ 2)
```

```
    ' Radius ist kleinster Abstand in x- oder y-Richtung
```

```
    Radius = Math.Min(Center.X, Center.Y) - 1
```

```
    ' Bild neu zeichnen
```

```
    Refresh()
```

```
End Sub
```

---

Listing 229: Zeichnen eines kreisförmigen Steuerelementes

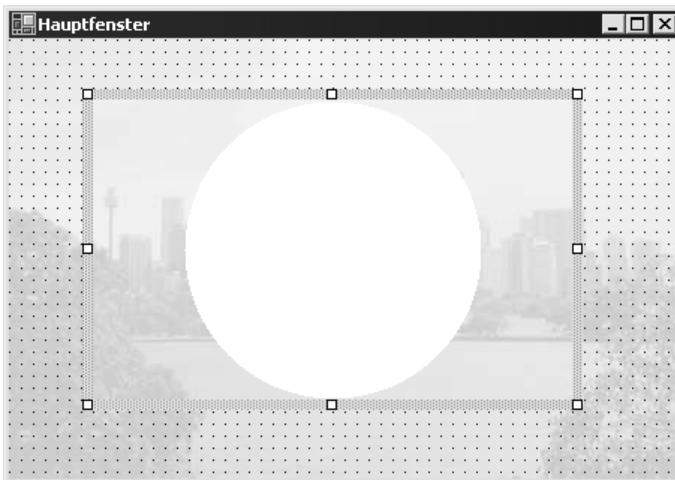


Abbildung 114: Steuerelemente müssen nicht immer rechteckig sein

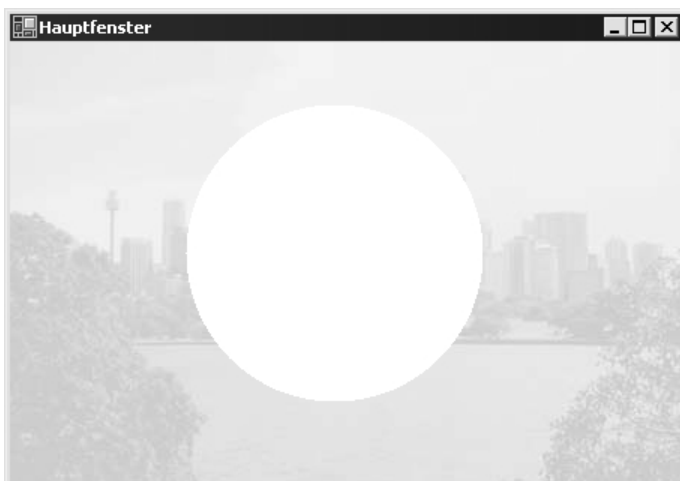


Abbildung 115: Zur Laufzeit sieht man nur die gezeichneten Umrisse

## 133 Mausposition in kreisförmigen Steuerelementen in Winkel umrechnen

Oft benötigt man für kreisförmige Steuerelemente eine Koordinatentransformation, die die kartesischen Koordinaten, die in dem `MouseMove`-, `MouseDown`- und `MouseUp`-Ereignissen übergeben werden, in Winkel und Abstand zum Mittelpunkt umrechnen. Mit etwas Schulmathematik und ein paar Funktionen der Klasse `Math` lässt sich diese Aufgabe leicht bewältigen.

Bei einem gewöhnlichen Koordinatensystem, bei dem die X-Achse nach rechts und die Y-Achse nach oben zeigt, gilt für den Winkel  $\hat{a}$  zwischen der positiven Y-Achse und einem beliebigen Punkt  $P(x, y)$ :

---


$$\tan(\hat{a}) = x / y$$


---

und somit für den Winkel :

---


$$\hat{a} = \text{atan}(x / y).$$


---

Da der Tangens jedoch über einen Vollkreis nicht eindeutig ist und somit der Arcustangens nur den Winkelbereich eines Halbkreises abdeckt, muss für die einzelnen Quadranten des Koordinatensystems eine Fallunterscheidung vorgenommen werden. Glücklicherweise bietet die Klasse `Math` bereits eine Methode, die Ihnen diese Arbeit abnimmt. Statt

---

```
Math.Atan (x / y)
```

---

verwendet man den Ausdruck

---

```
Math.Atan2 (x, y).
```

---

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Propert-  
yGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

Das Ergebnis dieses Funktionsaufrufes ist der Winkel in Bogenmaß zwischen der positiven Y-Achse und dem Punkt  $P(x, y)$ . Er ist positiv für  $x > 0$  und negativ für  $x < 0$ . Dieser Winkel muss nun noch auf das Koordinatensystem des Fensters (siehe Abbildung 116) umgerechnet werden. In diesem System zeigt die Y-Achse nach unten und die Bezugsachse für den gesuchten Winkel ist der negative Teil der Y-Achse. Zudem soll der Winkel auf einen Bereich von  $0^\circ$  bis  $360^\circ$  abgebildet werden. Folgende Umrechnungsformel liefert das gewünschte Ergebnis:

---


$$\alpha = 180^\circ - \text{atan2}(x, y) * 180^\circ / \pi$$


---

Der Radius errechnet sich nach Pythagoras :

---


$$r = \text{sqrt}(x^2 + y^2)$$


---

wobei sqrt die Wurzelfunktion darstellt.

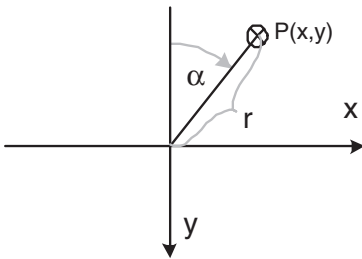


Abbildung 116: Umrechnen der X/Y-Koordinaten in Winkel und Radius

Wendet man die Formeln in der Überschrift der Methode `OnMouseMove` für das Steuerelement des vorherigen Rezeptes an, ergibt sich die in Listing 230 gezeigte Implementierung. Die Mausposition wird zunächst auf den Kreismittelpunkt bezogen. `dx` und `dy` sind dann die Koordinaten entsprechend des Systems in Abbildung 116. Nach erfolgter Berechnung wird das Steuerelement neu gezeichnet.

---

```
Protected Overrides Sub OnMouseMove(ByVal e As _
    System.Windows.Forms.MouseEventArgs)

    Dim dx As Double = e.X - Center.X
    Dim dy As Double = e.Y - Center.Y

    ' Winkel berechnen
    Me.MouseAngle = -Math.Atan2(dx, dy) * 180 / Math.PI + 180

    ' Aus Performance-Gründen wird hier das Quadrat nicht mit z^2
    ' berechnet werden
    Me.MouseRadius = Math.Sqrt(dx * dx + dy * dy)

    Invalidate()

End Sub
```

---

Listing 230: Winkel und Radius berechnen und anschließend Neuzeichnen des Fensters initiieren

Winkel und Radius werden in zwei Member-Variablen gespeichert:

```
' Winkel zwischen y-Achse und Mausposition
Protected MouseAngle As Double

' Abstand zwischen Kreismittelpunkt und Mausposition
Protected MouseRadius As Double
```

Die `OnPaint`-Methode wird für dieses Beispiel um folgende Ausgaben erweitert (siehe auch Abbildung 117):

1. Ausgabe des Winkels als Text
2. Ausgabe des Radius als Text
3. Zeichnen eines Kreisbogens bis zur Mausposition
4. Zeichnen einer Linie vom Mittelpunkt zur Mausposition

Hierzu werden die beiden berechneten Größen `MouseAngle` und `MouseRadius` herangezogen. Die Methode `DrawArc` zeichnet einen Kreisbogen beginnend mit einem Startwinkel über einen vorgegebenen Winkelbereich. Als Bezugsachse gilt die positive X-Achse, so dass sich als Startwinkel  $-90^\circ$  ergibt. Interessanterweise werden die Winkel für `DrawArc` in Grad und nicht, wie sonst in der Mathematik üblich, in Bogenmaß angegeben.

Zum Zeichnen der Linie vom Mittelpunkt zur Mausposition müssen die X/Y-Koordinaten zurückgerechnet werden. Hierbei gilt:

$$x = r * \sin(\alpha)$$

und

$$y = -r * \cos(\alpha).$$

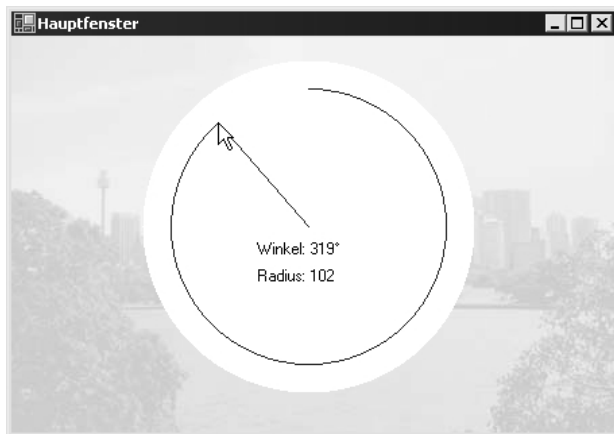


Abbildung 117: Umrechnen der Mausposition in Winkel und Radius

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung  
Win-  
dows  
Forms

Controls

Propor-  
tGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

Die vollständige Implementierung, die auch die notwendigen Typumwandlungen berücksichtigt, sehen Sie in Listing 231.

---

```
Protected Overrides Sub OnPaint(ByVal e As _
    System.Windows.Forms.PaintEventArgs)

    Dim g As Graphics = e.Graphics

    ' Koordinatenursprung in den Mittelpunkt legen
    g.TranslateTransform(Center.X, Center.Y)

    ' Kreis zeichnen
    Dim r As New Rectangle(-Radius, -Radius, Radius * 2, Radius * 2)
    g.FillEllipse(Brushes.White, r)

    ' Winkel und Radius ausgeben
    g.DrawString("Winkel: " & MouseAngle.ToString("0") & "°", _
        Font, Brushes.Black, -40, 10)
    g.DrawString("Radius: " & MouseRadius.ToString("0"), Font, _
        Brushes.Black, -40, 30)

    ' Zeichnen nur, wenn Radius > 0
    If MouseRadius > 0 Then
        ' Winkel als Kreisbogen zeichnen
        g.DrawArc(Pens.DarkBlue, -CSng(MouseRadius), _
            -CSng(MouseRadius), CSng(MouseRadius * 2), _
            CSng(MouseRadius * 2), -90, CSng(MouseAngle))

        ' Linie vom Mittelpunkt zur Mausposition zeichnen
        g.DrawLine(Pens.DarkBlue, 0, 0, CSng(MouseRadius * _
            Math.Sin(Math.PI / 180 * MouseAngle)), _
            -CSng(MouseRadius * Math.Cos(Math.PI / 180 * MouseAngle)))
    End If

End Sub
```

---

Listing 231: Darstellen von Winkel und Radius

## 134 Maus-Ereignisse zur Entwurfszeit abfangen

Im vorherigen Beispiel werden die Mausbewegungen, die das Steuerelement betreffen, in der überschriebenen Methode `OnMouseMove` abgefangen und auch dort das Neuzeichnen des Steuerelementes eingeleitet. Die Maus-Ereignisse werden allerdings nur zur Laufzeit an das Steuerelement weitergereicht. Zur Entwurfszeit fängt der Designer die Ereignisse ab. Sie werden genutzt, um beispielsweise das Control auf dem Fenster verschieben zu können.

Möchten Sie auch zur Entwurfszeit Maus-Ereignisse nutzen (siehe Abbildung 118), müssen Sie dem Designer dies explizit mitteilen. Hierfür ordnet man der Klasse des Steuerelementes über das `Designer`-Attribut eine Klasse zu, die die Verteilung der Ereignisse steuert.



---

```
Imports System.ComponentModel
```

```
<Designer(GetType(CircularUserControl.CircularUserControlDesigner))>
Public Class CircularUserControl
```

---

`CircularUserControlDesigner` ist eine innere Klasse von `CircularUserControl` und ist von `System.Windows.Forms.Design.ControlDesigner` abgeleitet. Überschrieben wird nur die Methode `GetHitTest`, die der Designer aufruft, um festzustellen, ob das Steuerelement die Mausereignisse an diesem Punkt selbst abhandeln möchte (Rückgabewert `True`) oder nicht (Rückgabewert `False`). In der Beispielimplementierung (Listing 232) liefert die Funktion `True` zurück, wenn der Punkt innerhalb des Kreises liegt. Die Mauseaktionen werden also im Entwurfsmodus auf die Kreisfläche beschränkt.

---

```
Class CircularUserControlDesigner
    Inherits System.Windows.Forms.Design.ControlDesigner

    Protected Overrides Function GetHitTest( _
        ByVal point As System.Drawing.Point) As Boolean

        ' Referenz des Steuerelementes
        Dim cuc As CircularUserControl = _
            DirectCast(Control, CircularUserControl)

        ' Koordinaten des angegebenen Punktes sind
        ' Bildschirmkoordinaten und müssen umgerechnet werden
        Dim p1 As Point = cuc.PointToClient(point)

        ' Mittelpunkt des Kreises
        Dim p2 As Point = cuc.Center

        ' x- und y-Koordinaten in Bezug auf Kreismittelpunkt
        Dim dx As Double = p1.X - p2.X
        Dim dy As Double = p1.Y - p2.Y

        ' Quadrat des Abstands zum Mittelpunkt
        Dim d As Double = dx * dx + dy * dy

        ' Rückgabe True, wenn Punkt innerhalb des Kreises liegt
        Return d < cuc.Radius * cuc.Radius

    End Function

End Class
```

---

Listing 232: `CircularUserControlDesigner` steuert die Zuordnung der Mausereignisse zur Entwurfszeit

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Propor-  
tivityGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schie-  
denes

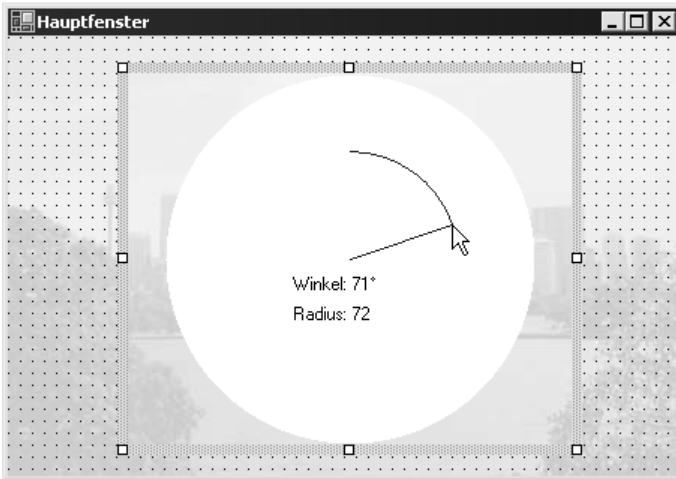


Abbildung 118: Mausaktionen zur Entwurfszeit durchführen

## 135 Ein Steuerelement zur grafischen Anzeige von Zeitbereichen programmieren

Kreisförmige Zeitdiagramme eignen sich hervorragend dazu, Zeitbereiche übersichtlich qualitativ darzustellen. Ordnet man dem Vollkreis eine feste Zeitspanne zu (z.B. 24 Stunden), dann lassen sich die einzelnen Zeitscheiben visuell leicht erfassen (siehe Abbildung 119). Insbesondere für sich zyklisch wiederholende Aufgaben oder Zustände (hier täglich) lässt sich diese Diagrammart vorteilhaft einsetzen.



Abbildung 119: Zeitbereiche als Kreissegmente darstellen

In diesem Rezept werden die Schritte erläutert, die dazu notwendig sind, das Steuerelement komfortabel bedienen zu können. Dazu gehört unter anderem, dass die Zeitscheiben im Entwurfsmodus über das Eigenschaftsfenster hinzugefügt, entfernt und verändert werden können und dass beim Überfahren des Steuerelementes mit der Maus Informationen zu Zeitpunkt und Zeitschei-

ben angezeigt werden. Die grundsätzlichen Schritte im Umgang mit kreisförmigen Steuerelementen wurden in den vorangegangenen Rezepten erläutert. Bis zum einsatzfähigen Control sind aber noch eine Reihe weiterer Aufgaben zu erfüllen.

Die Klasse `TimesliceControl` bildet den Kern des Steuerelementes. Neben den bereits bekannten Eigenschaften `Radius` und `Center` werden die folgenden zusätzlichen Member-Variablen benötigt:

---

```
' Auflistung für Zeitscheiben-Objekte
Protected Slices As New TimesliceCollection(Me)
```

```
' Gesamtbereich (Vollkreis) als Zeitspanne
Protected tsRange As New TimeSpan(1, 0, 0, 0)
```

```
' Anzahl der Skalierungsstriche
Protected nTicks As Integer = 24
```

```
' Befindet sich die Maus innerhalb
Protected MouseInside As Boolean = False
Protected TimeAtMouse As TimeSpan
```

---

Die öffentlichen Eigenschaften `Range` und `Ticks` werden direkt auf die geschützten Member-Variablen `tsRange` und `nTicks` abgebildet, während die Auflistung der Zeitbereiche über eine öffentliche schreibgeschützte Eigenschaft `TimeSlices` erfolgt, der noch ein zusätzliches Attribut zugeordnet wird, das weiter unten näher erläutert wird.

---

```
' Liste der Zeitscheiben-Objekte
<System.ComponentModel.Description( _
"Eingestellte Zeitscheiben"), _
Category("Zeiteinstellungen"), _
DesignerSerializationVisibility( _
DesignerSerializationVisibility.Content)> _
Public ReadOnly Property TimeSlices() As TimesliceCollection
    Get
        Return Slices
    End Get
End Property
```

---

*Listing 233: Öffentliche Eigenschaft `TimeSlices` erlaubt den Zugriff auf die Auflistung*

Ein Zeitbereich wird durch eine Instanz der Klasse `Timeslice` (Listing 234) definiert. Er beinhaltet Beginn und Ende des Zeitbereichs sowie einen Text für die Aufgabe. Um später einmal das Steuerelement problemlos für Zeitsteuerungen erweitern zu können, werden die Start- und Endzeit nicht als absolute Zeiten vom Typ `DateTime` gespeichert, sondern als relative Werte vom Typ `TimeSpan`. So kann man später dem Steuerelement eine Anfangszeit (die sich z.B. täglich um 24 Stunden verschiebt) zuordnen und durch Addition der `TimeSpan`-Werte direkt den korrekten absoluten Zeitpunkt ermitteln.

---

```
<DesignTimeVisible(False)> _
Public Class Timeslice
    Inherits System.ComponentModel.Component
```

---

*Listing 234: Klasse `Timeslice` definiert einen Zeitbereich*

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Propert-  
yGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

---

```

' Ereignis, das ausgelöst wird, wenn sich die Daten der
' Zeitscheibe geändert haben
Public Event ValueChanged As EventHandler

' Anfang und Ende des Bereichs
Protected PStartTime As TimeSpan
Protected PEndTime As TimeSpan

' Aufgabentext
Protected PTask As String

' Öffentliche Properties
<Editor(GetType(TimeUIEditor), GetType(UITypeEditor))> _
    Public Property StartTime() As TimeSpan
    Get
        Return PStartTime
    End Get
    Set(ByVal Value As TimeSpan)
        PStartTime = Value
        RaiseEvent ValueChanged(Me, New EventArgs)
    End Set
End Property

<Editor(GetType(TimeUIEditor), GetType(UITypeEditor))> _
Public Property EndTime() As TimeSpan
...
End Property

<Description("Aufgabe")> _
Public Property Task() As String
...
End Property

' Konstruktoren
Public Sub New(ByVal startTime As TimeSpan, _
    ByVal endTime As TimeSpan)

    Me.StartTime = startTime
    Me.EndTime = endTime
End Sub

Public Sub New()
End Sub

Public Overrides Function ToString() As String
    Return Me.StartTime.ToString() & "-" & Me.EndTime.ToString()
End Function

End Class

```

---

Listing 234: Klasse Timeslice definiert einen Zeitbereich (Forts.)

Besondere Aufmerksamkeit verdienen Attribut und Basisklasse von `Timeslice`. Die Klasse ist abgeleitet von `System.ComponentModel.Component`, um zu erreichen, dass der Designer automatisch den Code zur Instanzierung der `Timeslice`-Objekte sowie zum Hinzufügen der Objekte zur Auflistung generiert (siehe Listing 235). Ohne diese Basisklasse kann der Designer nur sehr umständlich zur Erzeugung des benötigten Codes veranlasst werden. Das zusätzliche Attribut `DesignTimeVisible (False)` verhindert, dass die `Timeslice`-Instanzen im Entwurfsmodus als Komponenten sichtbar gemacht werden.

```

/
'Timeslice1
/
Me.Timeslice1.EndTime = System.TimeSpan.Parse("08:15:00")
Me.Timeslice1.StartTime = System.TimeSpan.Parse("07:35:00")
Me.Timeslice1.Task = "Kinder zur Schule bringen"
/
'Timeslice2
/
Me.Timeslice2.EndTime = System.TimeSpan.Parse("19:20:00")
Me.Timeslice2.StartTime = System.TimeSpan.Parse("19:00:00")
Me.Timeslice2.Task = "Aktuelle Nachrichten ansehen"
/
'Timeslice3
/
Me.Timeslice3.EndTime = System.TimeSpan.Parse("13:30:00")
Me.Timeslice3.StartTime = System.TimeSpan.Parse("12:00:00")
Me.Timeslice3.Task = "Mittagspause"

Me.TimesliceControl1.TimeSlices.Add(Me.Timeslice1)
Me.TimesliceControl1.TimeSlices.Add(Me.Timeslice2)
Me.TimesliceControl1.TimeSlices.Add(Me.Timeslice3)

```

Listing 235: Vom Designer automatisch generierter Code für die definierten Zeitbereiche

Damit der Designer überhaupt die Einstellungen der Auflistung in `InitializeComponent` einträgt, muss der öffentlichen Property `TimeSlices` das Attribut `DesignerSerializationVisibility` mit dem Wert `Content` zugeordnet werden (siehe Listing 233). Ohne die Angabe dieses Attributs wird kein Code für den Aufbau der Liste generiert.

Die einzelnen Zeitscheiben werden in einer spezialisierten Auflistung verwaltet (siehe Listing 237). Beim Hinzufügen neuer `Timeslice`-Instanzen wird automatisch eine Ereignis-Routine (Listing 236) der Klasse `TimesliceControl` gebunden, die das Steuerelement informiert, wenn sich die Daten des `Timeslice`-Objektes geändert haben. Damit das auch funktioniert, wenn neue Zeitscheiben über den Auflistungs-Editor des Designers (der leider die `Add`-Methode umgeht) hinzugefügt werden, erfolgt die Bindung auch in der überschriebenen Methode `OnInsertComplete`. Alternativ ließe sich der Auflistungs-Editor durch Attribute auch dazu bewegen, einen speziellen Konstruktor der Klasse `Timeslice` mit entsprechenden Parametern aufzurufen. Abbildung 120 zeigt den Editor, den der Designer im Entwurfsmodus zur Bearbeitung der Auflistung zur Verfügung stellt.

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Propert-  
yGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

```
Public Sub TimesliceValueChanged( _
    ByVal sender As Object, ByVal e As EventArgs)

    Me.Invalidate()
End Sub
```

---

*Listing 236: Ereignis-Handler der Klasse TimesliceControl löst das Neuzeichnen des Steuerelementes aus*

---

```
Public Class TimesliceCollection
    Inherits CollectionBase

    ' Das Steuerelement, zu dem die Auflistung gehört
    Protected OwnerControl As TimesliceControl

    ' Konstruktor übernimmt das Parent-Control
    Public Sub New(ByVal owner As TimesliceControl)
        OwnerControl = owner
    End Sub

    ' Zeitscheibe hinzufügen
    Public Sub Add(ByVal slice As Timeslice)
        List.Add(slice)

        ' Ereignis-Handler für Änderungen binden
        AddHandler slice.ValueChanged, AddressOf _
            OwnerControl.TimesliceValueChanged

    End Sub

    ' Indizierter Zugriff auf die Liste
    Public ReadOnly Property Item(ByVal index As Integer) As Timeslice
        Get
            Return DirectCast(List(index), Timeslice)
        End Get
    End Property

    ' Bindung des Ereignis-Handlers, wenn neue Elemente über den
    ' Designer hinzugefügt werden
    Protected Overrides Sub OnInsertComplete( _
        ByVal index As Integer, ByVal value As Object)

        MyBase.OnInsertComplete(index, value)

        AddHandler DirectCast(value, Timeslice).ValueChanged, _
            AddressOf OwnerControl.TimesliceValueChanged

    End Sub

End Class
```

---

*Listing 237: Verwaltung der Zeitscheiben in einer typisierten Liste*

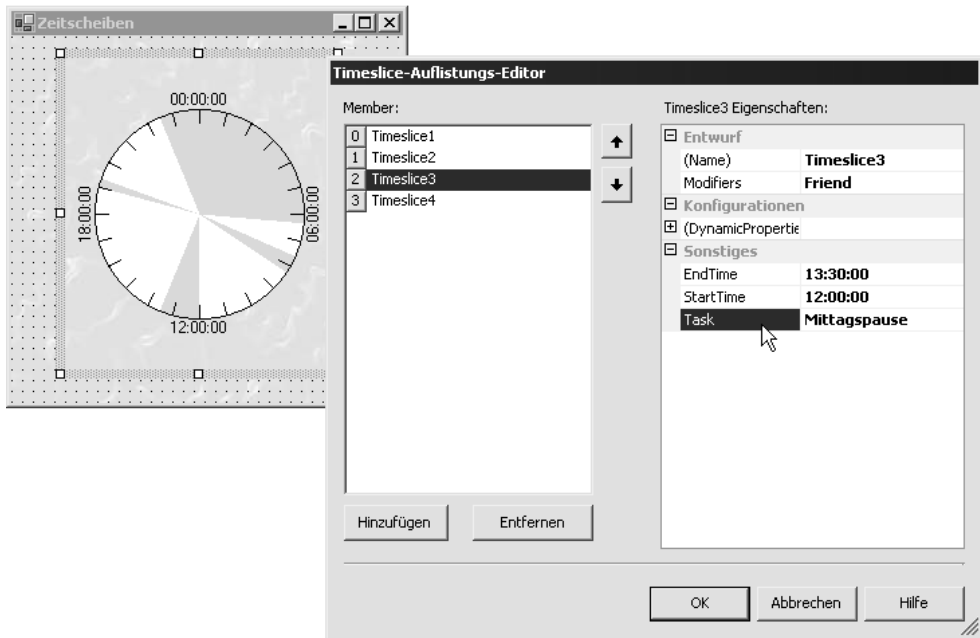


Abbildung 120: Ansicht des speziellen Editors für die Zeitscheibenliste im Entwurfsmodus

Den beiden `TimeSpan`-Eigenschaften der Klasse `Timeslice` wurde per Attribut ein spezieller Editor zugeordnet (vergl. Listing 234):

```
<Editor(GetType(TimeUIEditor), GetType(UITypeEditor))> _
    Public Property StartTime() As TimeSpan
```

Dieser Editor dient zur einfacheren Eingabe der Zeitangaben (siehe Abbildung 121) und wird in Rezept 8.11 ((Einen eigenen DropDown-Editor anzeigen)) näher erläutert.

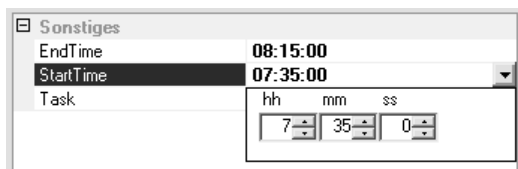


Abbildung 121: DropDown-Editor für einfachere Zeiteingabe

## Zeichnen der Zeitscheiben

Ähnlich wie in den vorherigen Beispielen wird in der Überschrift der Methode `OnResize` Mittelpunkt und Radius des Kreises festgelegt (Listing 238). Allerdings wird der Radius etwas kleiner gewählt, um noch Platz für zusätzliche Informationen bereit zu stellen.

---

```
Protected Overrides Sub OnResize(ByVal e As System.EventArgs)
    Center = New Point(Width \ 2, Height \ 2)
    Radius = Math.Min(Center.X, Center.Y) - 1 - CInt(Font.Size * 2.5)
    Refresh()
End Sub
```

---

*Listing 238: Neuberechnung der Kreiskoordinaten bei Ändern der Größe des Steuerelementes*

Eine Reihe von Hilfsfunktionen dient zur Umrechnung von Koordinaten (Listing 239). `GetTime` ermittelt den Zeitpunkt zu einer gegebenen Position, `GetPoint` liefert die Koordinaten eines Punktes für ein angegebenes Zeitverhältnis und `GetRadius` liefert den Abstand eines gegebenen Punktes zum Mittelpunkt des Kreises.

---

```
' Zeitpunkt an X/Y-Koordinate berechnen
Protected Function GetTime(ByVal x As Integer, ByVal y As Integer) _
    As TimeSpan

    Dim angle As Double = Math.PI - _
        Math.Atan2(x - Center.X, y - Center.Y)

    Return New TimeSpan(CLng(10000 * Range.TotalMilliseconds * _
        angle / 2 / Math.PI))

End Function

' Punkt zu vorgegebenem Verhältnis (Zeitpunkt/Gesamtbereich)
' und angegebenen Radius ermitteln
' Wird z.B. zum Zeichnen der Skala benötigt
Protected Function GetPoint(ByVal ratio As Double, _
    ByVal radius As Double) As Point

    ' Winkel ergibt sich aus dem Verhältnis (ratio=1 = Vollkreis)
    Dim angle As Double = Math.PI * 2 * ratio

    ' Koordinaten berechnen
    Dim x As Integer = CInt(Math.Sin(angle) * radius)
    Dim y As Integer = -CInt(Math.Cos(angle) * radius)

    ' Punkt zurückgeben
    Return New Point(x, y)

End Function

' Abstand zum Mittelpunkt für gegebenen Punkt berechnen
Protected Function GetRadius(ByVal x As Integer, _
    ByVal y As Integer) As Integer

    Return CInt(Math.Sqrt((Center.X - x) * (Center.X - x) + _
        (Center.Y - y) * (Center.Y - y)))

End Function
```

---

*Listing 239: Hilfsfunktionen für die Darstellung des Steuerelementes*



Gezeichnet wird das Steuerelement wie üblich in OnPaint (Listing 240). Folgende Teilschritte werden abgearbeitet:

1. Zeichnen der Kreisfüllung
2. Zeichnen der Segmente mit semitransparenter Farbe, um Überlappungen sichtbar zu machen
3. Zeichnen der Skalenstriche
4. Ausgabe der zugeordneten Zeiten für die Positionen Oben, Rechts, Unten und Links

```
Protected Overrides Sub OnPaint( _
    ByVal e As System.Windows.Forms.PaintEventArgs)

    Dim g As Graphics = e.Graphics
    g.TranslateTransform(Center.X, Center.Y)

    ' Kreis zeichnen
    Dim r As New Rectangle(-Radius, -Radius, Radius * 2, Radius * 2)
    g.FillEllipse(Brushes.White, r)

    Dim angle1, angle2 As Double

    ' Brush zum Zeichnen der Segmente
    Dim b As Brush = New SolidBrush(Color.FromArgb(60, _
        Color.DarkCyan))

    ' Segmente zeichnen
    Dim ms As Double = Range.TotalMilliseconds
    Dim ts As Timeslice
    For Each ts In Slices
        angle1 = -90 + 360 / ms * ts.StartTime.TotalMilliseconds
        angle2 = 360 / ms * _
            ts.EndTime.Subtract(ts.StartTime).TotalMilliseconds
        If ts.EndTime.CompareTo(ts.StartTime) < 0 Then angle2 += 360
        g.FillPie(b, r, CSng(angle1), CSng(angle2))
    Next

    ' Skalenstriche zeichnen
    Dim i As Integer
    For i = 0 To nTicks - 1
        Dim p1 As Point = GetPoint(i / nTicks, Radius)
        Dim p2 As Point = GetPoint(i / nTicks, Radius - 10)
        g.DrawLine(Pens.Black, p1, p2)
    Next

    ' Skalenwert oben
    Dim t As String = "00:00:00"
    Dim sz As SizeF = g.MeasureString(t, Font)
    Dim p As Point = GetPoint(0, Radius)
    g.DrawString(t, Font, Brushes.Black, p.X - sz.Width / 2, _
        p.Y - sz.Height)
```

Listing 240: Zeichnen des gesamten Steuerelementes in OnPaint

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

```

' Skalenwert unten
t = TimeSpan.FromMilliseconds( _
    Range.TotalMilliseconds / 2).ToString()

sz = g.MeasureString(t, Font)
p = GetPoint(0.5, Radius)
g.DrawString(t, Font, Brushes.Black, p.X - sz.Width / 2, p.Y)

' Skalenwert links
g.RotateTransform( - 90)
t = TimeSpan.FromMilliseconds( _
    Range.TotalMilliseconds * 3 / 4).ToString()

sz = g.MeasureString(t, Font)
p = GetPoint(0, Radius)
g.DrawString(t, Font, Brushes.Black, p.X - sz.Width / 2, _
    p.Y - sz.Height)

' Skalenwert rechts
t = TimeSpan.FromMilliseconds( _
    Range.TotalMilliseconds / 4).ToString()

sz = g.MeasureString(t, Font)
p = GetPoint(0.5, Radius)
g.DrawString(t, Font, Brushes.Black, p.X - sz.Width / 2, p.Y)

' Umriss des Kreises zeichnen
g.DrawEllipse(Pens.Black, r)

b.Dispose()

End Sub

```

---

*Listing 240: Zeichnen des gesamten Steuerelementes in OnPaint (Forts.)*

## Auf Mausbewegungen reagieren

Führt der Anwender die Maus über den Kreisbereich des Steuerelementes, so werden ihm zusätzliche Informationen angezeigt (Abbildung 122). Grundsätzlich wird rechts unten der Zeitpunkt angezeigt, der der Position unter dem Mauszeiger zugeordnet ist. So lässt sich schnell abschätzen, wann ein Segment beginnt oder endet.

Wird der Mauszeiger über ein Segment bewegt, wird der Text der Task-Eigenschaft eingeblendet. Steht der Mauszeiger außerhalb eines Segmentes, wird der Text wieder ausgeblendet.

Wenn der Mauszeiger den Kreisbereich wieder verlässt, werden alle zusätzlichen Ausgaben wieder gelöscht. Listing 241 zeigt die Implementierung der `OnMouseMove`-Überschreibung, in der ermittelt wird, ob sich der Cursor innerhalb oder außerhalb des Kreises befindet.



Abbildung 122: Anzeige zusätzlicher Informationen für den Zeitpunkt unter dem Mauszeiger

```
Protected Overrides Sub OnMouseMove( _
    ByVal e As System.Windows.Forms.MouseEventArgs)

    ' Position innerhalb des Kreises?
    Dim r As Integer = GetRadius(e.X, e.Y)
    MouseInside = r <= Radius
    If MouseInside Then
        ' Wenn ja, Zeitpunkt feststellen und speichern
        TimeAtMouse = GetTime(e.X, e.Y)
    End If

    ' Neu zeichnen
    Refresh()

End Sub
```

Listing 241: Bei Mausbewegungen auf dem Steuerelement Neuzeichnen veranlassen

Für die Ausgabe der Texte wird die OnPaint-Methode wie folgt erweitert:

```
If MouseInside Then
    ' Text für Zeit an Cursorposition ausgeben
    t = TimeAtMouse.ToString().Substring(0, 8)
    sz = g.MeasureString(t, Font)
    g.DrawString(t, Font, Brushes.Black, Width \ 2 - sz.Width, p.Y)

    ' Liste der Zeitscheiben durchlaufen, bis ein Bereich für
    ' die Position unter dem Cursor gefunden wurde
    For Each tsl As Timeslice In Slices
        If TimeInTimeslice(TimeAtMouse, tsl) Then
            ' Zeitbereich gefunden
            ' Text ausgeben und Suche abbrechen
            sz = g.MeasureString(tsl.Task, Font)
```

---

```

        g.DrawString(tsl.Task, Font, Brushes.Black, -sz.Width / 2, _
            Radius + sz.Height * 1.5F)
    Exit For
End If
Next
End If

```

---

Die Hilfsfunktion `TimeInTimeslice` übernimmt hierbei die Berechnung, ob der Zeitpunkt, der der Cursor-Position zugeordnet ist, innerhalb des untersuchten Zeitbereiches liegt. In einer Fallunterscheidung wird gesondert berücksichtigt, dass sich ein Zeitbereich wie im Bild dargestellt auch über den Zeitpunkt Null hinaus erstrecken kann.

---

```

Protected Function TimeInTimeslice(ByVal time As TimeSpan, _
    ByVal slice As Timeslice) As Boolean

    ' Bereichsabfrage abhängig davon, ob Zeitscheibe über den
    ' Nullpunkt hinausgeht
    If TimeSpan.op_LessThanOrEqual(slice.StartTime, slice.EndTime) _
    Then
        Return TimeSpan.op_LessThanOrEqual(slice.StartTime, time) _
            And TimeSpan.op_LessThanOrEqual(time, slice.EndTime)
    Else
        Return TimeSpan.op_LessThanOrEqual(slice.StartTime, time) _
            And TimeSpan.op_LessThanOrEqual(time, Me.Range) _
            Or TimeSpan.op_LessThanOrEqual(time, slice.EndTime)
    End If

End Function

```

---

*Listing 242: Überprüfen, ob der Zeitpunkt time im Bereich slice liegt*

## 136 Neue Zeitscheiben zur Laufzeit mit der Maus hinzufügen

Soll der Anwender die Möglichkeit haben, zur Laufzeit neue Zeitscheiben hinzuzufügen, stehen Ihnen viele Möglichkeiten zur Realisierung offen. Beispielsweise können Sie ein `PropertyGrid`-Control bereitstellen, in dem die Eigenschaften genauso bearbeitet werden können wie zur Entwurfszeit (siehe Kapitel 0 ((

Eigenschaftsfenster (`PropertyGrid`))), oder Sie reagieren auf Maus-Ereignisse und erlauben das Aufziehen neuer Zeitscheiben mit der Maus. Letzteres lässt sich in folgende Schritte unterteilen:

1. Beginn des Vorgangs über den Aufruf eines Kontextmenüs
2. Anzeige eines Dialogs zur Eingabe des Aufgabentextes
3. Beginn des Ziehvorgangs im `MouseDown`-Ereignis
4. Visualisierung des Bereiches in `OnMouseMove` / `OnPaint`
5. Abschluss des Ziehvorgangs und Hinzufügen der neuen Zeitscheibe in `OnMouseUp`

Wenn das Kontextmenü nicht im Entwurfsmodus zur Verfügung stehen soll, dann darf es nicht in der Initialisierungsphase hinzugefügt werden. Stattdessen wird es in `OnLoad` an das Steuerelement

gebunden (Listing 243). Im Ereignis-Handler des Menüpunktes (Listing 244) wird ein Dialog angezeigt, der dem Anwender die Möglichkeit gibt, den Aufgabentext einzugeben und den Vorgang fortzusetzen oder abubrechen (siehe Abbildung 123 und Listing 245). Bestätigt der Anwender die Eingabe mit OK, dann wird eine Instanz von `Timeslice` angelegt und der Referenzvariablen `TemporaryTimeslice` zugewiesen.

```
Protected Overrides Sub OnLoad(ByVal e As System.EventArgs)
    ' Kontextmenü nur für Laufzeitmodus
    If Not designmode Then Me.ContextMenu = ContextMenu1
End Sub
```

Listing 243: Kontextmenü nicht im Entwurfsmodus bereitstellen

```
' Verweis auf temporär angelegte Zeitscheibe
Protected TemporaryTimeslice As Timeslice = Nothing

' Linke Maustaste wurde gedrückt, Anwender zieht Zeitscheibe auf
Protected CreateNewSliceStarted As Boolean

Private Sub MenuItem1_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles MenuItem1.Click

    ' Text für Aufgabenstellung einlesen
    Dim text As String = "Aufgabe " & Slices.Count + 1
    If DialogTimesliceTask.CreateAndShow(text) = DialogResult.OK Then
        ' Neuanlegen durch Mausaktion starten
        TemporaryTimeslice = New Timeslice
        TemporaryTimeslice.Task = text
    End If
End Sub
```

Listing 244: Kontextmenü zum Hinzufügen einer neuen Zeitscheibe

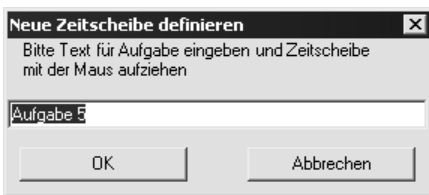


Abbildung 123: Texteingabe vor Definition des Zeitbereichs

```
Public Shared Function CreateAndShow(ByRef text As String) _
    As DialogResult

    ' Dialog-Instanz anlegen
    Dim dialog As New DialogTimesliceTask
```

Listing 245: Anzeige des Dialoges und Transfer des Textes in der statischen Methode `CreateAndShow`

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

```

' Text in Textbox eintragen
dialog.TXTTask.Text = text

' Dialog modal anzeigen
Dim dr As DialogResult = dialog.ShowDialog()

' Text zurücklesen
text = dialog.TXTTask.Text
dialog.Dispose()
Return dr

```

End Function

---

*Listing 245: Anzeige des Dialoges und Transfer des Textes in der statischen Methode CreateAndShow (Forts.)*

Nach Betätigung der OK-Schaltfläche beginnt der Zieh-Vorgang durch Drücken der linken Maustaste. Im `MouseDown`-Ereignis (Listing 246) wird ein Flag gesetzt (`CreateNewSliceStarted`) und die Anfangszeit in das temporäre Zeitscheiben-Objekt eingetragen.

---

```

Protected Overrides Sub OnMouseDown( _
    ByVal e As System.Windows.Forms.MouseEventArgs)

    ' Ziehen nur mit linker Maustaste
    If e.Button <> MouseButton.Left Then
        TemporaryTimeslice = Nothing
        CreateNewSliceStarted = False
        Exit Sub
    End If

    ' Modus = Neue Zeitscheibe anlegen?
    If Not TemporaryTimeslice Is Nothing Then
        ' OK, jetzt wird der Bereich mit der Maus aufgezogen
        CreateNewSliceStarted = True
        ' Zeitpunkt merken
        TemporaryTimeslice.StartTime = GetTime(e.X, e.Y)
    End If

End Sub

```

---

*Listing 246: Beginn des Ziehvorgangs in OnMouseDown*

Bei jeder Mausbewegung mit gedrückter linker Maustaste muss nun der aufgezeichnete Bereich angezeigt werden. Dazu wird in `OnMouseMove` der bisherige Code erweitert:

---

```

' Zeitpunkt speichern für Aufziehen einer neuen Zeitscheibe
If CreateNewSliceStarted Then
    TemporaryTimeslice.EndTime = TimeAtMouse
End If

```

---

So wird die Endzeit der Zeitscheibe definiert und kann in OnPaint dargestellt werden:

```
' Wenn eine temporäre Zeitscheibe existiert, diese zeichnen
If Not TemporaryTimeslice Is Nothing Then
    ts = TemporaryTimeslice
    angle1 = -90 + 360 / ms * ts.StartTime.TotalMilliseconds
    angle2 = 360 / ms * _
        ts.EndTime.Subtract(ts.StartTime).TotalMilliseconds
    If ts.EndTime.CompareTo(ts.StartTime) < 0 Then angle2 += 360
    g.FillPie(Brushes.Red, r, CSng(angle1), CSng(angle2))
End If
```

Die Darstellung entspricht der der anderen Zeitscheiben, außer, dass eine andere Füllfarbe gewählt wird, um sie besonders hervorzuheben (siehe Abbildung 124). In OnMouseUp (Listing 247) wird der Zieh-Vorgang beendet und das temporäre Timeslice-Objekt der Slices-Auflistung hinzugefügt.



Abbildung 124: Aufziehen eines neuen Zeitbereiches mit der Maus

```
Protected Overrides Sub OnMouseUp( _
    ByVal e As System.Windows.Forms.MouseEventArgs)

    If Not TemporaryTimeslice Is Nothing Then
        ' Endzeit endgültig festlegen
        TemporaryTimeslice.EndTime = GetTime(e.X, e.Y)

        ' Zeitscheibe der Auflistung hinzufügen
        Slices.Add(TemporaryTimeslice)

        ' Ziehvorgang beendet
        CreateNewSliceStarted = False
        TemporaryTimeslice = Nothing
    End If
End Sub
```

Listing 247: Abschluss des Zieh-Vorgangs

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Prop-  
ertyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

End If

End Sub

---

*Listing 247: Abschluss des Zieh-Vorgangs (Forts.)*

### Vorschläge für Erweiterungen

Es gibt viele Möglichkeiten, das vorgestellte Steuerelement für eigene Zwecke zu erweitern. Beispielsweise können Sie einen Timer hinzufügen und Ereignisse auslösen, wenn eine Zeitscheibe beginnt bzw. abläuft. Mit einer dicken farbigen Linie könnten Sie die aktuelle Zeit markieren.

Sollen viele überlappende Segmente dargestellt werden, ist die Wahl der Farbe sowie der Transparenz von entscheidender Bedeutung. Eventuell ist es von Vorteil, die Segmentgrenzen besonders hervorzuheben. Die Segmentfarbe könnte als Bestandteil des `Timeslice`-Objektes dem Anwender zur Änderung angeboten werden. Auch Farbverläufe lassen sich für eine bessere visuelle Anzeige nutzen.

## 137 Nachrichten verschicken mit `SendMessage`

Windows arbeitet intern größtenteils mit Nachrichten. Die Aufrufe vieler API-Funktionen zur Manipulation von Fenstern werden umgesetzt, indem via `SendMessage` entsprechende Botschaften an die Fenster geschickt werden. Einige Methoden verschiedener Steuerelemente lassen sich nur auf diesem Wege ausführen.

`SendMessage` ist eine API-Funktion, die im Framework leider nicht direkt zur Verfügung gestellt wird. Zwei Varianten stehen zur Nutzung der Funktion zur Verfügung:

### 1. Einbindung der API-Funktion über eine `Declare`-Anweisung:

1. `wparam` und `lparam` als `IntPtr`

---

```
Public Declare Function SendMessage Lib "user32.dll" Alias _
    "SendMessageW" (ByVal hwnd As IntPtr, ByVal message As Integer, _
    ByVal wparam As IntPtr, ByVal lparam As IntPtr) As Integer
```

---

2. `wparam` und `lparam` als `Integer`

---

```
Public Declare Function SendMessage Lib "user32.dll" Alias _
    "SendMessageW" (ByVal hwnd As IntPtr, ByVal message As Integer, _
    ByVal wparam As Integer, ByVal lparam As Integer) As Integer
```

---

### 2. Indirekter Aufruf mit native Code über `NativeWindow`

Die zweite Variante ist etwas komplizierter, verzichtet aber auf den direkten Aufruf von API-Funktionen. Listing 248 zeigt die Implementierung, bei der zunächst eine Instanz von `NativeWindow` angelegt wird. Anschließend wird ein `Message`-Objekt mit den betreffenden Daten generiert und letztlich durch Aufruf der `WndProc`-Methode des Fensters der Aufruf von `SendMessage` simuliert.



```

Public Shared Sub SendMessage2(ByVal hwnd As IntPtr, _
    ByVal message As Integer, ByVal wParam As IntPtr, _
    ByVal lParam As IntPtr)

    ' Instanz von NativeWindow für Handle abrufen
    Dim nw As NativeWindow = NativeWindow.FromHandle(hwnd)
    ' Definition des Message-Objektes
    Dim m As System.Windows.Forms.Message = _
        System.Windows.Forms.Message.Create(hwnd, message, _
            wParam, lParam)

    ' Aufruf der WndProc-Methode
    nw.DefWndProc(m)

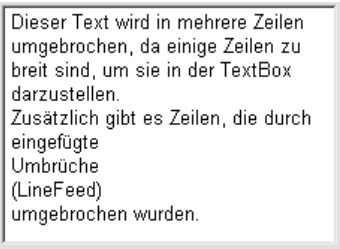
End Sub

```

Listing 248: SendMessage ohne API-Funktionsaufruf

## 138 Zeilen ermitteln, wie sie in einer mehrzeiligen TextBox dargestellt werden

Eine TextBox, deren Multiline-Eigenschaft auf True gestellt ist, kann mehrzeilige Texte darstellen und auch automatisch umbrechen (Abbildung 125). Zeilen werden automatisch umgebrochen, wenn die Eigenschaft WordWrap den Wert True besitzt und die Breite der TextBox nicht für eine einzeilige Darstellung ausreicht. In diesem und den nachfolgenden Rezepten finden Sie Lösungen für den Umgang mit mehrzeiligen TextBoxen, die die Möglichkeiten des Frameworks ergänzen. Leider wurde nur ein kleiner Teil der für TextBoxen nutzbaren API-Funktionen in die TextBox-Klasse aufgenommen. Die wichtigsten der fehlenden werden hier beschrieben.



Dieser Text wird in mehrere Zeilen umgebrochen, da einige Zeilen zu breit sind, um sie in der TextBox darzustellen. Zusätzlich gibt es Zeilen, die durch eingefügte Umbrüche (LineFeed) umgebrochen wurden.

Abbildung 125: Automatische und manuelle Zeilenumbrüche in einer mehrzeiligen TextBox

### Text in Zeilen zerlegen

Das TextBox-Steuerelement stellt eine Eigenschaft namens Lines zur Verfügung, die ein String-Array zurückgibt, das für jede Zeile ein Element enthält. So ließe sich die Aufgabenstellung einfach erledigen, wäre da nicht das Problem, dass die automatischen Umbrüche nicht berücksichtigt werden. Folgendes Array mit sechs Elementen liefert TextBox.Lines für den im Bild dargestellten Text zurück:

```

0: Dieser Text wird In mehrere Zeilen umgebrochen, da einige Zeilen zu breit sind, um
   sie In der TextBox darzustellen.
1: Zusätzlich gibt es Zeilen, die durch

```

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

```

2: eingefügte
3: Umbrüche
4: (LineFeed)
5: umgebrochen wurden.
6:

```

---

Die erste automatisch umgebrochene Zeile wird in einem einzigen Element gespeichert. Um auch diese mit einzubeziehen, kann man die `TextBox` via `SendMessage` anweisen, für jeden automatischen Umbruch ebenfalls ein Zeilenumbruchszeichen einzutragen. Die erforderliche Message heißt `EM_FMTLINES` und hat den hexadezimalen Wert `08`. Während jeder explizite Zeilenumbruch mit der Zeichenfolge `CR LF` (Carriage-Return Linefeed) gespeichert wird, werden die automatischen Umbrüche mit `CR CR LF` gekennzeichnet. So kann die `TextBox.Lines` liefert in diesem Fall aber leider auch ein falsches Ergebnis, diesmal mit zu vielen Zeilen:

---

```

0: Dieser Text wird in mehrere Zeilen
1:
2: umgebrochen, da einige Zeilen zu breit
3:
4: sind, um sie in der TextBox darzustellen.
5: Zusätzlich gibt es Zeilen, die durch
6: eingefügte
7: Umbrüche
8: (LineFeed)
9: umgebrochen wurden.
10:

```

---

Die Kombinationen `CR CR LF` werden jetzt als zwei Zeilen interpretiert. Dadurch ergeben sich für die beiden automatischen Umbrüche insgesamt vier Zeilen mehr. Daher werden erst durch einen Aufruf von `Replace` alle `CRs` aus dem String entfernt und anschließend die String-Methode `Split` aufgerufen, die den Text anhand des Zeichens `LF` umbrechen kann:

---

```

0: Dieser Text wird in mehrere Zeilen
1: umgebrochen, da einige Zeilen zu breit
2: sind, um sie in der TextBox darzustellen.
3: Zusätzlich gibt es Zeilen, die durch
4: eingefügte
5: Umbrüche
6: (LineFeed)
7: umgebrochen wurden.
8:

```

---

Auf diese Weise erfolgt der korrekte Umbruch, wie ihn auch die `TextBox` vorgenommen hat. Listing 249 zeigt die vollständige Implementierung zur Ermittlung der korrekten Zeilenumbrüche, Abbildung 126 die Ausgabe des Testprogramms.

---

```
Public Shared Function GetLines(ByVal tb As TextBox) As String()
```

---

*Listing 249: Zahl der Zeilen einer mehrzeiligen TextBox ermitteln*

```

' CR CR LF für automatische Umbrüche eintragen
API.SendMessage(tb.Handle, API.TBMultiline.EM_FMTLINES, 1, 0)

' Zeilen ermitteln
Dim text As String = tb.Text

' Eingefügte Umbrüche wieder entfernen
API.SendMessage(tb.Handle, API.TBMultiline.EM_FMTLINES, 0, 0)

' Returns entfernen
text = text.Replace(vbCr, "")

' Text in Zeilen aufspalten
Dim Lines() As String = text.Split(vbLf)

Return Lines

```

End Function

Listing 249: Zahl der Zeilen einer mehrzeiligen TextBox ermitteln (Forts.)



Abbildung 126: Der Text einer mehrzeiligen TextBox, wie er von der TextBox umgebrochen wird

## 139 Anzahl der Zeilen einer mehrzeiligen TextBox ermitteln

Das vorangegangene Rezept hat gezeigt, dass die Lines-Eigenschaft der TextBox wenig geeignet ist, die tatsächliche Anzahl von Zeilen, die eine TextBox anzeigt, zu ermitteln. Natürlich können Sie die Methode `GetLines` aufrufen und die Anzahl der Array-Elemente feststellen. Es geht aber auch einfacher und vor allem schneller.

Eine TextBox gibt die Anzahl der Zeilen beim Zusenden der Message `EM_GETLINECOUNT` preis. Zusätzliche Parameter sind nicht erforderlich. In Listing 250 ist die Implementierung der Methode `GetLineCount` aufgeführt, die lediglich `SendMessage` aufruft und den erhaltenen Funk-

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Propert-  
yGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

tionswert zurückgibt. Abbildung 127 zeigt den Vergleich zwischen dem Ergebnis von `GetLineCount` und `TextBox.Lines.Length`.

---

```
Public Shared Function GetLineCount(ByVal tb As TextBox) _
    As Integer

    ' Anzahl der Zeilen mit EM_GETLINECOUNT ermitteln
    Return API.SendMessage(tb.Handle, _
        API.TBMultiline.EM_GETLINECOUNT, 0, 0)

End Function
```

---

*Listing 250: Ermitteln der Anzahl der Zeilen einer mehrzeiligen TextBox*

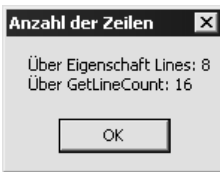


Abbildung 127: Anzahl der Zeilen ermitteln

## 140 Zeilenindex aus Zeichenindex ermitteln (mehrzeilige TextBox)

Wenn Sie den Index eines Zeichens kennen (z.B. der aktuellen Cursor-Position), können Sie über die Nachricht `EM_LINEFROMCHAR` die zugehörige Zeile ermitteln. Die aktuelle Cursor-Position ergibt sich aus der Summe der Eigenschaften `SelectionStart` und `SelectionLength`:

---

```
CursorPos = TBText.SelectionStart + TBText.SelectionLength
```

---

Abbildung 128 zeigt den Textausschnitt, wie ihn die TextBox darstellt. Der Cursor steht in der siebten Zeile (Index 6). Abbildung 129 präsentiert das Ergebnis der Abfrage, das durch den Aufruf der Methode `GetLineFromCharIndex` (Listing 251) berechnet wurde. Die Indizierung der Zeilen beginnt bei Null.

Abbildung 128: In welcher Zeile steht der Cursor?

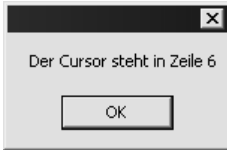


Abbildung 129: Zeilenindex aus Zeichenindex berechnet

```
Public Shared Function GetLineFromCharIndex(ByVal tb As TextBox, _
    ByVal charPos As Integer) As Integer

    ' Zeilennummer mit EM_LINEFROMCHAR ermitteln
    Return API.SendMessage(tb.Handle, _
        API.TBMultiline.EM_LINEFROMCHAR, charPos, 0)

End Function
```

Listing 251: Über SendMessage die Zeile zur angegebenen Zeichenposition feststellen

## 141 Index des ersten Zeichens einer Zeile ermitteln (mehrzeilige TextBox)

Beispielsweise um eine Zeile in einer TextBox markieren zu können, müssen Sie wissen, welches das erste Zeichen der betreffenden Zeile ist. Dieses lässt sich über die Nachricht `EM_LINEINDEX` ermitteln. Die Methode `GetIndexOfFirstCharInLine` (Listing 252) kapselt den entsprechenden `SendMessage`-Aufruf. `GetIndexOfFirstCharInLine` berechnet den Index des ersten Zeichens der Zeile, deren Index (0-basiert) als Parameter übergeben wird.

```
Public Shared Function GetIndexOfFirstCharInLine(_
    ByVal tb As TextBox, ByVal lineIndex As Integer) As Integer

    ' Nachricht EM_LINEINDEX senden
    Return API.SendMessage(tb.Handle, API.TBMultiline.EM_LINEINDEX, _
        lineIndex, 0)

End Function
```

Listing 252: Erste Zeichen einer Zeile ermitteln mit `EM_LINEINDEX`

## 142 Index der ersten sichtbaren Zeile einer mehrzeiligen TextBox bestimmen

Ist die Anzahl der Zeilen eines Textes größer als die der darstellbaren Zeilen einer TextBox, dann kann die TextBox nur einen Ausschnitt anzeigen. Die oberste sichtbare Zeile, die in Abhängigkeit von der Scrollposition in der TextBox angezeigt wird, lässt sich abermals über `SendMessage` und die Nachricht `EM_GETFIRSTVISIBLELINE` ermitteln. Listing 253 zeigt die Implementierung der Methode `GetFirstVisibleLine`, die diesen Aufruf kapselt.

---

```
Public Shared Function GetFirstVisibleLine(ByVal tb As TextBox)
```

```
    ' Nachricht EM_GETFIRSTVISIBLELINE senden
    Return API.SendMessage(tb.Handle, _
        API.TBMultiline.EM_GETFIRSTVISIBLELINE, 0, 0)
```

```
End Function
```

---

*Listing 253: Mit SendMessage die erste sichtbare Zeile einer mehrzeiligen TextBox feststellen*

## 143 Zeichenindex aus Grafikkoordinaten berechnen (mehrzeilige TextBox)

Möchten Sie wissen, über welchem Zeichen einer TextBox sich die Maus befindet, können Sie auch das über `SendMessage` erfahren. Um die Nachricht (`EM_CHARFROMPOS`) nutzen zu können, müssen die Koordinaten in einem `Int32`-Wert zusammengefasst werden. Die Y-Koordinate wird in die oberen 16 Bit (Highword) und die X-Koordinate in die unteren 16 Bit (Lowword) eingetragen. Listing 254 zeigt die vollständige Implementierung der Methode `GetCharIndexFromPos`.

Der Wert, den `SendMessage` bei diesem Aufruf zurückgibt, ist zusammengesetzt aus der Zeilennummer und der Zeichenposition. Um die Zeichenposition zu separieren werden die oberen 16 Bit durch eine AND-Verknüpfung ausgeblendet.

Im Beispielprogramm wird im `MouseMove`-Ereignis der TextBox die Methode `GetCharIndexFromPos` aufgerufen und das Ergebnis auf einem Label dargestellt (siehe Abbildung 130). Zusätzlich wird das zugehörige Zeichen ausgegeben. Listing 255 zeigt den notwendigen Code als Beispielanwendung für die Methode `GetCharIndexFromPos`.

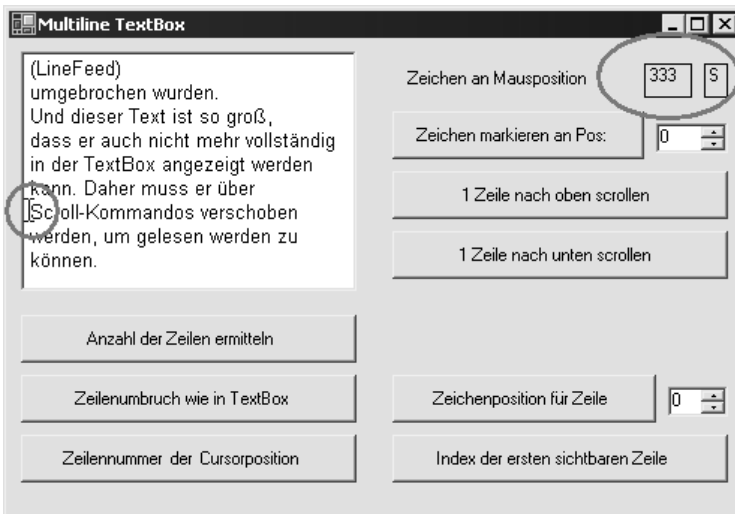


Abbildung 130: Zeichen und -index an der Mausposition ermitteln

```

Public Shared Function GetCharIndexFromPos(ByVal tb As TextBox, _
    ByVal x As Integer, ByVal y As Integer) As Integer

    ' Koordinaten zusammensetzen (Highword = y, Lowword=x)
    Dim pos As Integer = (y << 16) Or x

    ' Position abfragen. Lowword enthält Zeilennummer
    Return API.SendMessage(tb.Handle, _
        API.TBMultiline.EM_CHARFROMPOS, 0, pos) And &HFFFF

End Function

```

*Listing 254: Mit der Nachricht EM\_CHARFROMPOS und ein paar Bit-Operationen erhält man die Zeichenposition zu einer gegebenen Koordinate*

```

Private Sub TBText_MouseMove(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.MouseEventArgs) _
    Handles TBText.MouseMove

    ' Zeichenindex aus Cursorposition ermitteln
    Dim n As Integer = GetCharIndexFromPos(TBText, e.X, e.Y)

    ' Zeichenindex anzeigen
    LBLCharIndex.Text = n

    ' Wenn der Index gültig ist, das Zeichen an dieser Position
    ' anzeigen
    If n >= 0 And n < TBText.TextLength Then
        LBLChar.Text = TBText.Text.Chars(n)
    End If

End Sub

```

*Listing 255: Anzeigen des Zeichens und seiner Position, das sich unter dem Mauszeiger befindet*

## 144 Koordinate eines Zeichens ermitteln (mehrzeilige TextBox)

Auch der umgekehrte Vorgang zum vorangegangenen Beispiel, nämlich zu einem vorgegebenen Zeichen die Koordinate im Koordinatensystem der TextBox zu ermitteln, lässt sich über `SendMessage` vornehmen. Die Nachricht hierzu hat die Bezeichnung `EM_POSFROMCHAR`. Listing 256 zeigt die vollständige Methode `GetPointFromCharIndex`, die den `SendMessage`-Aufruf kapselt. Als Parameter wird der Zeichenindex an `SendMessage` übergeben, zurück kommt die X/Y-Koordinate, verpackt in einem 32 Bit-Wert. Praktischerweise kennt die Struktur `Point` einen Konstruktor, der die Koordinate in diesem Format annimmt und umwandelt. So kann man auf eigene Bit-Operationen an dieser Stelle verzichten.

In Abbildung 131 sehen Sie eine Anwendung der Methode `GetPointFromCharIndex`. Über ein `NumericUpDown`-Feld lässt sich eine Zeichenposition vorgeben und auf Tastendruck das entsprechende Zeichen in der TextBox markieren. Zur Markierung wird temporär eine kleine halbtransparente Ellipse über das Zeichen gezeichnet.

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Proper-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

---

```
Public Shared Function GetPointFromCharIndex(ByVal tb As TextBox, _
    ByVal charIndex As Integer)

    ' Position abfragen
    Dim pos As Integer = API.SendMessage(tb.Handle, _
        API.TBMultiline.EM_POSFROMCHAR, charIndex, 0)

    ' Point-Struktur anlegen und zurückgeben
    Return New Point(pos)

End Function
```

---

Listing 256: Grafikkordinate eines Zeichens in einer mehrzeiligen TextBox abfragen

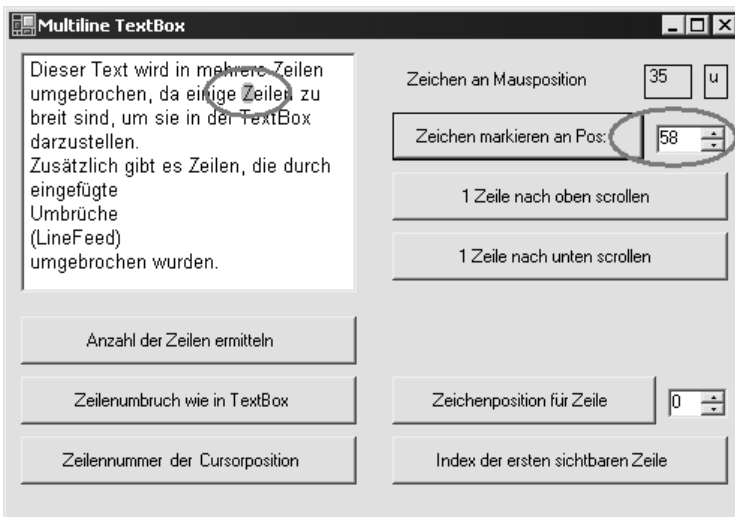


Abbildung 131: Das Zeichen an der ausgewählten Position (rechts) wird im Text mit einer kleinen durchsichtigen Ellipse gekennzeichnet

## 145 Mehrzeilige TextBox per Code auf- und abwärts scrollen

Sicher kennen Sie die Methode `ScrollToCaret`, um den Inhalt einer TextBox so zu verschieben, dass die Einfügemarke sichtbar wird. Wenn Sie die TextBox aber nur um eine bestimmte Anzahl von Zeilen auf- oder abwärts verschieben wollen, gibt es keine (leicht zugängliche) Methode im Framework. Aber selbstverständlich wieder eine Windows-Nachricht: `EM_LINESCROLL`. Die Methode `LineScroll` (Listing 257) kapselt den `SendMessage`-Aufruf. Ein Aufruf mit dem Wert 1 verschiebt den Text um eine Zeile nach oben, während ein Wert von -1 für eine Verschiebung um eine Zeile nach unten führt.



```
Public Shared Sub LineScroll(ByVal tb As TextBox, _
    ByVal lines As Integer)

    ' Nachricht EM_LINESCROLL senden
    API.SendMessage(tb.Handle, API.TBMultiline.EM_LINESCROLL, _
        0, lines)

End Sub
```

*Listing 257: Den Inhalt einer TextBox um eine vorgegebene Zahl von Zeilen nach oben oder unten verschieben*

## 146 Tabulatorpositionen in einer mehrzeiligen TextBox setzen

Nur, wenn die Multiline-Eigenschaft einer TextBox auf `True` gesetzt wird, unterstützt sie die tabellarische Schreibweise. Um auch Tabulatorzeichen bei der Eingabe zuzulassen, muss zusätzlich die Eigenschaft `AcceptsTab` auf `True` gesetzt werden. Sonst führt die Betätigung der `Tab`-Taste lediglich dazu, dass das nächste Steuerelement den Fokus erhält.

Die Tabulatorpositionen (Tab Stops) ihrerseits lassen sich nicht über die Eigenschaften und Methoden der TextBox einstellen. Hier ist wieder ein Aufruf von `SendMessage` gefragt. Über die Nachricht `EM_SETTABSTOPS` lassen sich die Tabulatorpositionen auf zwei Arten einstellen:

1. Alle Tabulatorpositionen haben denselben Abstand.

Hierzu wird beim Aufruf von `SendMessage` für den Parameter `wparam` der Wert 1 und für den Parameter `lparam` die Referenz einer Integer-Variablen, die den Abstandswert enthält, übergeben

2. Jede Tabulatorposition hat eine individuelle Position.

Hierzu wird beim Aufruf von `SendMessage` für den Parameter `lparam` die Referenz eines Integer-Arrays übergeben, das die gewünschten Positionen enthält. Für den Parameter `wparam` wird als Wert die Anzahl der Elemente des Arrays übergeben.

Für diese Aufgabe wird die Methode `SendMessageRef` definiert. Sie unterscheidet sich von den `SendMessage`-Überladungen dadurch, dass der Parameter `lparam` als Referenzparameter definiert wird. Leider lässt VB.NET es nicht zu, dass sich beim Überladen die Parameter nur durch `ByVal` bzw. `ByRef` unterscheiden. Die Definition sieht folgendermaßen aus:

```
Public Declare Function SendMessageRef Lib "user32.dll" Alias _
    "SendMessageW" (ByVal hwnd As IntPtr, ByVal message As Integer, _
        ByVal wparam As Int32, ByRef lparam As Int32) As Integer
```

Der folgende Aufruf legt den Abstand der Tabulatorpositionen konstant auf 20 Pixel fest:

```
' Alle Tabulatorpositionen haben einen Abstand von 20 Pixel
Dim tabwidth As Integer = 20
API.SendMessageRef(TBTabStop1.Handle, _
    API.TBMultiline.EM_SETTABSTOPS, 1, tabwidth)
```

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tung  
Win-  
dows  
Forms

Controls

Propor-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

Mit diesem Aufruf legen Sie Tabulatorpositionen an den Positionen 30, 50, 80 und 120 fest:

---

```
' Array für Tabulatorpositionen
Dim tabstops() As Integer = {30, 50, 80, 120}

' Tabulatorpositionen festlegen
API.SendMessageRef(TBTabStop2.Handle, _
    API.TBMultiline.EM_SETTABSTOPS, tabstops.Length, tabstops(0))
```

---

Beide Varianten werden in einem Beispiel eingesetzt. Das Ergebnis sehen Sie in Abbildung 132.

Bedenken Sie dabei bitte, dass die Einheit für die Tabulatorpositionen Pixel ist und nicht Zeichen. Bei Änderung der Schriftart bzw. -größe kann sich die Aufteilung des Textes auf die Spalten verschieben.

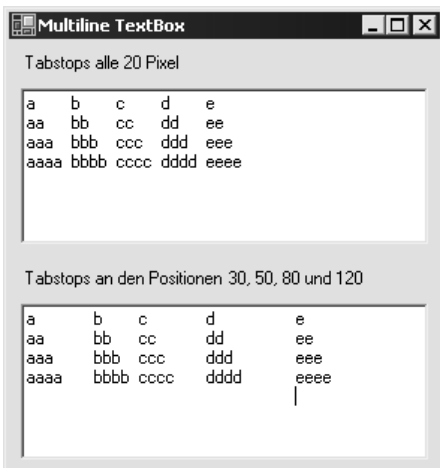


Abbildung 132: Linksbündige Tabstops können in einer mehrzeiligen TextBox frei definiert werden

# Eigenschaftsfenster (PropertyGrid)

Wie bereits in der Einleitung erwähnt, kommt man mit dem Eigenschaftsfenster bei der Programmierung mit Visual Studio ständig in Berührung, meist ohne es bewusst wahrzunehmen. Das Eigenschaftsfenster lässt sich zum einen vielfältig für eigene Anwendungen einsetzen, zum anderen muss man die Funktionsweise verstehen, will man eigene Steuerelemente programmieren und dem Anwender auch für diese den gewohnten Komfort bei deren Nutzung in der Entwicklungsumgebung bieten. Dieses Kapitel zeigt Wege auf, wie Sie das `PropertyGrid`-Steuerelement für eigene Objekte einsetzen und anwendungsspezifische Erweiterungen und Darstellungen vorgesehen werden können.

In den Beispielen werden, bis auf eine Ausnahme, jeweils in einem `PropertyGrid`-Control, das zur Laufzeit auf einem Fenster angezeigt wird, die Eigenschaften ebenfalls zur Laufzeit erzeugter Objekte dargestellt. Die gezeigten Vorgehensweisen lassen sich aber direkt auf Steuerelemente, deren Eigenschaften im Designer angezeigt werden sollen, übertragen. Da Steuerelemente von der Klasse `Control` abgeleitet sind, beinhalten sie bereits viele geerbte Eigenschaften, die in den Beispielen vom Wesentlichen ablenken würden.

## 147 Grundlegende Attribute

Nach der Installation von Visual Studio .NET befindet sich das `PropertyGrid`-Steuerelement noch nicht in der Toolbox, sondern muss über das Kontextmenü **ELEMENTE HINZUFÜGEN / ENTFERNEN...** eingebunden werden (s. Abbildung 133). Danach kann es wie jedes andere sichtbare Steuerelement auf eigenen Fenstern und benutzerdefinierten Steuerelementen eingesetzt werden.

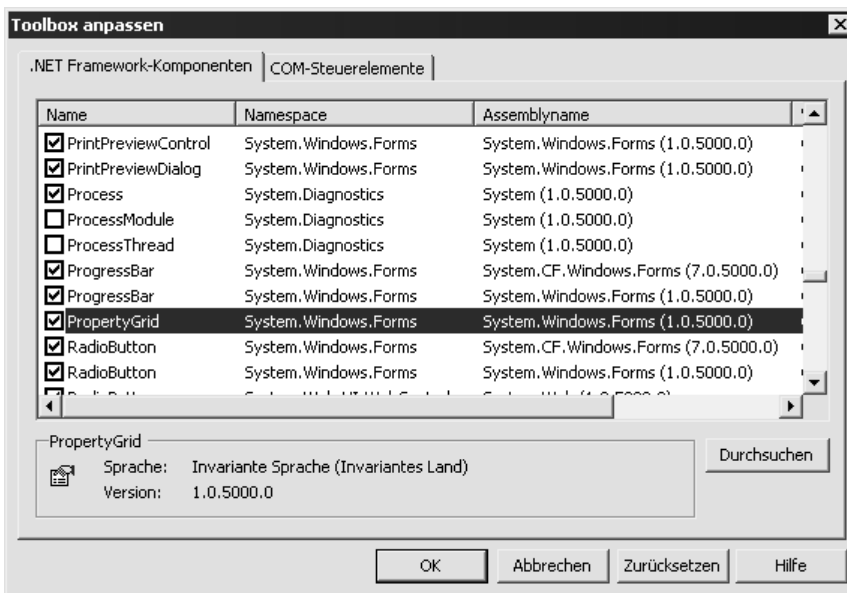


Abbildung 133: Hinzufügen des `PropertyGrid`-Steuerelementes

Zur Demonstration wird eine Beispielklasse verwendet, die zur Beschreibung der amerikanischen Präsidenten dienen soll. Alle Daten werden in Form von geschützten Feldern angelegt, die durch öffentliche Eigenschaften repräsentiert werden. Die Klasse wird ergänzt um einen Konstruktor und eine ToString-Überschreibung. Listing 258 zeigt die Implementierung.

---

```
Public Class USPresident

    ' Parteizugehörigkeiten
    Public Enum USPartyAffiliation
        Federalist
        Democrat
        DemocraticRepublican
        Republican
    End Enum

    ' Definition der geschützten Felder

    ' Name des Präsidenten
    Protected PName As String
    ' Laufende Nummerierung des Amtes
    Protected PPresidentialNumber As Integer
    ' Geburtsjahr
    Protected PBirthday As DateTime
    ' Beginn und Ende der Amtszeit
    Protected PStartPresidency As Integer
    Protected PEndPresidency As Integer
    ' Parteizugehörigkeit
    Protected PPartyAffiliation As USPartyAffiliation
    ' Bilddatei
    Protected PPicture As String

    Public Sub New(ByVal name As String, _
        ByVal presNumber As Integer, _
        ByVal birthday As DateTime, ByVal startPres As Integer, _
        ByVal endPres As Integer, ByVal party As USPartyAffiliation, _
        ByVal pic As String)

        Me.PName = name
        Me.PPresidentialNumber = presNumber
        Me.PBirthday = birthday
        Me.PStartPresidency = startPres
        Me.PEndPresidency = endPres
        Me.PPartyAffiliation = party
        Me.PPicture = pic
    End Sub

    Public Overrides Function ToString() As String
        Return String.Format("{0} ({1} - {2})", PName, _
            PStartPresidency, PEndPresidency)
    End Function
```

---

```

' Definition der öffentlichen Eigenschaften
Public Property Name() As String
    Get
        Return PName
    End Get
    Set(ByVal Value As String)
        PName = Value
    End Set
End Property

' Alle weiteren Properties nach dem gleichen Muster
' ...
End Class

```

Listing 258: Beispielklasse USPresident (Forts.)

Im Beispielprogramm werden einer `ListBox` Instanzen dieser Klasse zugewiesen. Wird ein Eintrag ausgewählt, dann werden das zugehörige Portrait angezeigt und die Eigenschaften des Objektes im Eigenschaftsfenster dargestellt (Abbildung 134). Zum Anzeigen der Eigenschaften ist nur eine einzige Zuweisung erforderlich:

```
PPGPresident.SelectedObject = LBPresidents.SelectedItem
```

Alles andere erledigt das Steuerelement selbst. Via Reflection ermittelt es sämtliche öffentliche Eigenschaften und deren Datentyp und zeigt diese entsprechend an. Alle Eigenschaften werden der Rubrik `SONSTIGES` zugeordnet.

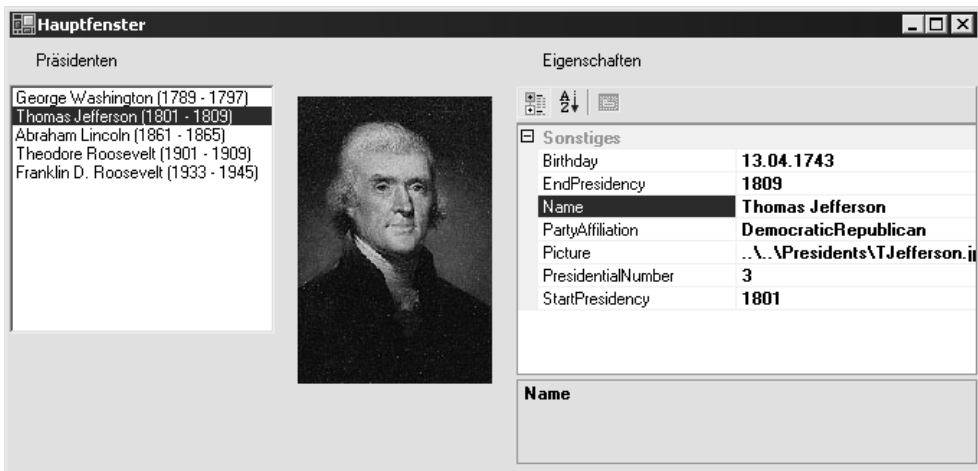


Abbildung 134: Einfacher Einsatz des Eigenschaftsfensters in eigener Anwendung

Die Eigenschaften lassen sich ändern, sofern es die passende `Property-Set-Methode` gibt. In dieser Methode muss ggf. Code vorgesehen werden, um auf die Änderung der Eigenschaft zu reagieren. Möchten Sie Eigenschaften schreibgeschützt anzeigen, lassen Sie die `Property-Set-Methode` weg.

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Prop-  
ertyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

Vieles wird bei der Darstellung der Eigenschaften im PropertyGrid über Attribute gesteuert. Ob eine Eigenschaft überhaupt angezeigt wird, legen Sie mit dem Attribut `BrowsableAttribute` fest. Möchten Sie eine Eigenschaft von der Darstellung im PropertyGrid ausschließen, deklarieren Sie sie, wie in Listing 259 gezeigt.

---

```
Protected PInternalData As String
<Browsable(False)> _
Public Property InternalData() As String
    Get
        Return PInternalData
    End Get
    Set(ByVal Value As String)
        PInternalData = Value
    End Set
End Property
```

---

*Listing 259: Ausschließen einer Eigenschaft von der Darstellung im PropertyGrid*

Sollen die Eigenschaften in andere Kategorien eingeordnet werden als `SONSTIGES`, dann können Sie das über das Attribut `CategoryAttribute` steuern. Die Klasse `CategoryAttribute` kennt bereits einige Standard-Kategorien, die sich hauptsächlich auf Windows-Steuerelemente beziehen. Dazu gehören z.B. `Action`, `Appearance` und `Layout`. Hinweise zu den vordefinierten Kategorien finden Sie in der MSDN-Dokumentation in der Beschreibung der `CategoryAttribute`-Klasse. Möchten Sie die Standardnamen verwenden, dann benutzen Sie die englischen Bezeichnungen. Sie werden automatisch in die Zielsprache übersetzt.

Eigene Kategorien definieren Sie, indem Sie den frei wählbaren Namen im Konstruktor des `Category`-Attributs angeben. Listing 260 zeigt exemplarisch die Definition für eine Eigenschaft, Abbildung 135 die Auswirkung im PropertyGrid.

---

```
<Category("Persönliches")> _
Public Property Birthday() As DateTime
    Get
        Return PBirthday
    End Get
    Set(ByVal Value As DateTime)
        PBirthday = Value
    End Set
End Property
```

---

*Listing 260: Einordnung der Eigenschaften in Kategorien*

Bestimmt kennen Sie die Möglichkeit, sich im untersten Feld des Eigenschaftsfensters eine Beschreibung der Eigenschaft anzeigen zu lassen. Auch diese Beschreibung lässt sich über ein Attribut vorgeben. Hierfür ist das `Description`-Attribut vorgesehen. Listing 261 zeigt die Definition, Abbildung 136 die Anzeige.



Abbildung 135: Eigenschaften verschiedenen Kategorien zuordnen

```
<Category("Amt"), _
Description("Jahr, in dem die Präsidentschaft endete")> _
Public Property EndPresidency() As Integer
    Get
        Return PEndPresidency
    End Get
    Set(ByVal Value As Integer)
        PEndPresidency = Value
    End Set
End Property
```

Listing 261: Hinzufügen einer Beschreibung mit Hilfe des Description-Attributs



Abbildung 136: Erklären Sie die Eigenschaften, damit der Anwender sie leichter verstehen kann

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Propor-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

## 148 Eigenschaften mehrerer Objekte gleichzeitig anzeigen

Im vorangegangenen Beispiel konnte in der ListBox immer nur ein Eintrag zur Zeit gewählt werden. Dieser wurde dann der `SelectedObject`-Eigenschaft des `PropertyGrids` zugewiesen. Das `PropertyGrid` verfügt zusätzlich aber auch über die Eigenschaft `SelectedObjects`, der sie ein beliebiges `Object`-Array zuweisen können. Im Eigenschaftsfenster werden dann nur die Eigenschaften angezeigt, die alle Objekte gemeinsam unterstützen. Werte werden nur angezeigt, wenn sie für alle Objekte gleichermaßen gelten.

In Listing 262 sehen Sie den angepassten Code für die Mehrfachauswahl, Abbildung 137 zeigt das Ergebnis für zwei gleichzeitig ausgewählte Präsidenten.

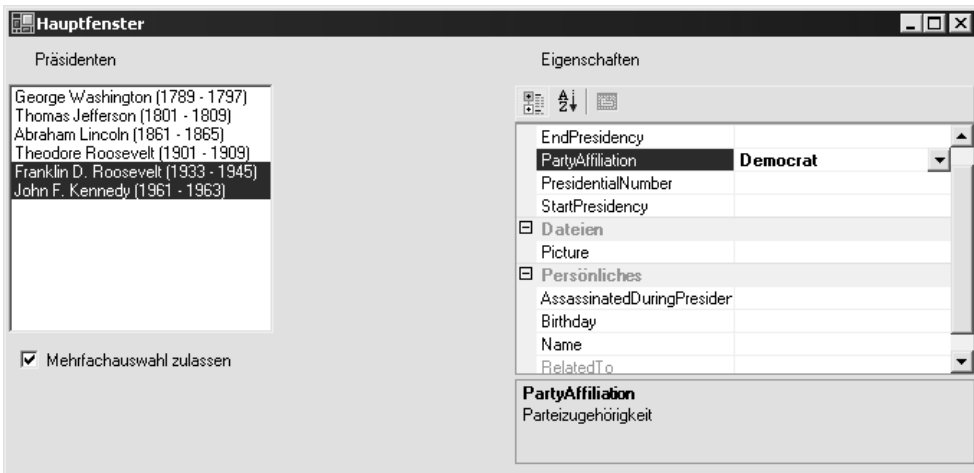


Abbildung 137: Eigenschaften bei gleichzeitiger Auswahl mehrerer Objekte

```
Private Sub LBPresidents_SelectedIndexChanged( _
    ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles LBPresidents.SelectedIndexChanged

    ' Präsident ermitteln
    Dim p As USPresident
    p = DirectCast(LBPresidents.SelectedItem, USPresident)

    ' Ressourcen freigeben
    If Not Portrait.Image Is Nothing Then Portrait.Image.Dispose()

    ' Mehrfachauswahl oder nicht?
    If CHKMultiSelect.Checked Then
        ' Kein Bild
        Portrait.Image = Nothing

    ' Object-Array für ausgewählte Präsidenten anlegen
    Dim items(LBPresidents.SelectedItems.Count - 1) As Object
```

Listing 262: Mehrfachauswahl über die `SelectedObjects`-Eigenschaft



```

    LBPresidents.SelectedItems.CopyTo(items, 0)

    ' Objekte im PropertyGrid auswählen
    PPGPresident.SelectedObjects = items
Else

    ' Bild anzeigen
    Portrait.Image = New Bitmap(p.Picture)
    ' Eigenschaften der Auswahl anzeigen
    PPGPresident.SelectedObject = LBPresidents.SelectedItem
End If

End Sub

```

Listing 262: Mehrfachauswahl über die SelectedObjects-Eigenschaft (Forts.)

## 149 Abfangen ungültiger Werte

Wie immer, wenn Anwender Daten eingeben können, müssen diese auf Plausibilität und Gültigkeit überprüft und im Fehlerfall zurückgewiesen werden. Die Überprüfung erfolgt nicht im PropertyGrid-Control, sondern in der Set-Methode der jeweiligen Eigenschaft. Ist das eingegebene Datum ungültig, kann die Methode eine `Exception` auslösen und das PropertyGrid zu einer Fehlermeldung veranlassen (siehe Abbildung 138).



Abbildung 138: Zurückweisen ungültiger Werte

Im Beispiel wird die laufende Nummerierung der Präsidenten auf den Bereich von 1 bis 100 eingeschränkt und im Fehlerfall eine `Exception` vom Typ `ArgumentOutOfRangeException` ausgelöst (Listing 263).

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Prop-  
ertyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

```

Public Property PresidentialNumber() As Integer
    Get
        Return PPresidentialNumber
    End Get
    Set(ByVal Value As Integer)
        If Value < 1 Or Value > 100 Then
            Throw New ArgumentOutOfRangeException( _
                "PresidentialNumber", Value, _
                "Wert muss zwischen 1 und 100 liegen")
        End If
        PPresidentialNumber = Value
    End Set
End Property

```

Listing 263: Auslösen einer Exception, um die Annahme ungültiger oder unplausibler Daten zu unterbinden

## 150 Standardwerte für Eigenschaften

Standardwerte lassen sich mit Hilfe des Attributs `DefaultValue` definieren. Sie haben zwei Aufgaben. Zum einen zeigen sie durch Normal- oder Fettschrift an, ob die eingestellte Eigenschaft der Standardwert ist. Der Standardwert wird mit Normalschrift angezeigt (siehe Abbildung 139, Eigenschaft `AssassinatedDuringPresidency`), vom Standardwert abweichende Werte fett.

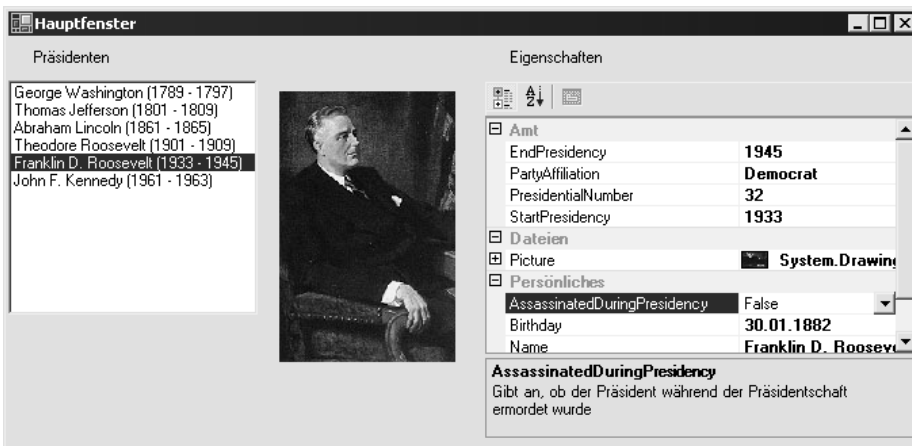


Abbildung 139: Anzeige des Standardwertes in Normalschrift

Der Standardwert hat auch Einfluss auf die Code-Generierung des Designers. Nur für Werte, die vom angegebenen Standardwert abweichen, wird in `InitializeComponent` Code angelegt.

Außerdem werden Standardwerte für einige Editoren benötigt. Zur Demonstration wurde in der Beispielklasse der Datentyp der Eigenschaft `Picture` von `String` in `Image` umgewandelt. Das `PropertyGrid` bietet dann für diese Eigenschaft automatisch einen Dateidialog an, um eine passende Bilddatei auswählen zu können. Möchten Sie diesen Eintrag löschen, um dem Objekt keine Bilddatei zuzuordnen, dann geht das nur, wenn für die Eigenschaft ein Standardwert (hier `Nothing`) definiert wurde. Listing 264 zeigt die Definitionen der Eigenschaften `AssassinatedDuringPresidency` und `Picture`.

```

Protected PAssassinatedDuringPresidency As Boolean

<Category("Persönliches"), _
Description("Gibt an, ob der Präsident während der " & _
    "Präsidentschaft ermordet wurde"), _
DefaultValue(False)> _
Public Property AssassinatedDuringPresidency() As Boolean
    Get
        Return PAssassinatedDuringPresidency
    End Get
    Set(ByVal Value As Boolean)
        PAssassinatedDuringPresidency = Value
    End Set
End Property

Protected PPicture As Image
<Category("Dateien"), _
DefaultValue(GetType(Object), Nothing), _
Description("Pfad der Portrait-Datei")> _
Public Property Picture() As Image
    Get
        Return PPicture
    End Get
    Set(ByVal Value As Image)
        PPicture = Value
    End Set
End Property

```

*Listing 264: Standardwerte für Eigenschaften*

## 151 Festlegen einer Standard-Eigenschaft

Sie können festlegen, welche Eigenschaft selektiert werden soll, wenn das `PropertyGrid-Control` die Eigenschaften eines Objektes neu anzeigt. Dazu brauchen Sie lediglich der betreffenden Klasse das Attribut `DefaultProperty` zuzuweisen und den Namen der Eigenschaft anzugeben:

```

<DefaultProperty("Address")> _
Public Class Person
    Public Property Address() As String
    ...
End Property
    Public Property Name() As String
    ...
End Property
End Class

```

## 152 Eigenschaften gegen Veränderungen im `PropertyGrid-Control` schützen

Soll eine Eigenschaft zwar prinzipiell für Schreib- und Lesezugriffe verfügbar sein, aber im `PropertyGrid-Control` schreibgeschützt dargestellt werden, dann können Sie dieser Eigenschaft das

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Proper-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

ReadOnly-Attribut zuordnen. Listing 265 zeigt ein Beispiel, bei dem die Eigenschaft Birthday geschützt wird, Abbildung 140 zeigt die Darstellung der schreibgeschützten Eigenschaft im PropertyGrid.

```
<ReadOnlyAttribute(True)> _
Public Property Birthday() As DateTime
    Get
        Return PBirthday
    End Get
    Set(ByVal Value As DateTime)
        PBirthday = Value
    End Set
End Property
```

Listing 265: Schreibschutz einer Eigenschaft, der nur für die Darstellung im PropertyGrid gilt

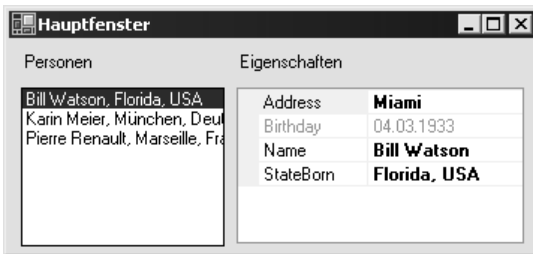


Abbildung 140: Die schreibgeschützte Eigenschaft wird grau dargestellt

### Hinweis

Beachten Sie bitte, dass Sie bei Verwendung des `ReadOnly`-Attributs entgegen der sonstigen Syntax für Attribute den Klassennamen (`ReadOnlyAttribute`) vollständig ausschreiben müssen, da es sonst zu einer Kollision mit dem Schlüsselwort `ReadOnly` kommt.

## 153 Enumerationswerte kombinieren

Wie Sie in Rezept 8.1 ((Grundlegende Attribute)) bereits gesehen haben, können Sie auch Enumerationsen als Eigenschaftstyp verwenden. Das PropertyGrid zeigt einen solchen Wert als Text an, sofern der Zahlenwert in der Enumeration vorkommt. Für Werte aus dem Enum-Datentyp `VehicleManufacturer` (siehe Listing 266) werden die Bezeichnungen der Konstanten angezeigt. Eine Kombination dieser Werte, selbst wenn die zugrunde liegenden Zahlenwerte es zuließen, ist nicht möglich. Das Eigenschaftsfenster bietet nur die Mitglieder der Enumeration zur Auswahl an.

```
Public Enum VehicleManufacturer
    VW
    BMW
    GM
    DaimlerChrysler
End Enum
```

Listing 266: Enumerationen mit und ohne `Flags`-Attribut

```

<Flags(>> _
Public Enum VehicleEquipment
    Airconditioner = 1
    Radio = 2
    Telephone = 4
End Enum

```

Listing 266: Enumerationen mit und ohne Flags-Attribut (Forts.)

Sollen mehrere Enum-Konstanten miteinander kombiniert werden (z.B. Ausstattungsmerkmale, die sich nicht gegenseitig ausschließen), dann müssen zwei Voraussetzungen erfüllt sein:

1. Die Werte müssen voneinander unabhängige Zweierpotenzen sein, damit die Bits ohne Kollision mit einer Oder-Verknüpfung kombiniert werden können.
2. Der Enumeration muss das `Flags`-Attribut zugewiesen werden, damit das `PropertyGrid` die Kombinationen anzeigen und zulassen kann.

In Abbildung 141 sehen Sie die Anzeige der kombinierten Ausstattungsmerkmale `AIRCONDITIONER` und `TELEPHONE`. Der im Bild angezeigte Eintrag wurde folgendermaßen im Programm eingefügt:

```

LBVehicles.Items.Add( _
    New Vehicle(VehicleManufacturer.DaimlerChrysler, _
        VehicleEquipment.Airconditioner Or VehicleEquipment.Telephone))

```

Während der Compiler das `Flags`-Attribut ignoriert (die Oder-Verknüpfung im Sourcecode ist davon unabhängig immer zulässig), entscheidet das `PropertyGrid` anhand dieses Attributs, wie die Werte darzustellen sind. Sind mehrere Bits gesetzt, werden die Namen der betreffenden Konstanten mit Kommas abgetrennt angezeigt.

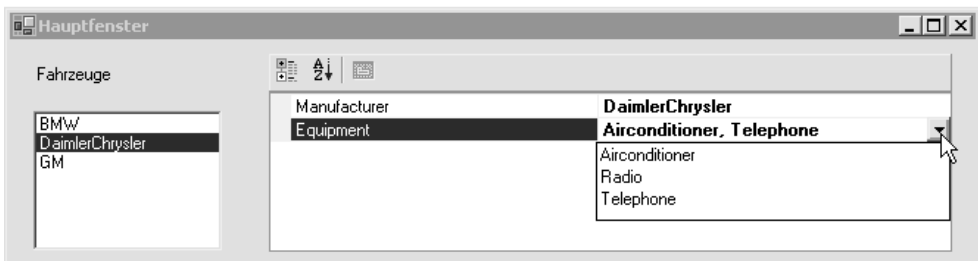


Abbildung 141: Kombinationen von Enum-Werten zulassen mit dem `Flags`-Attribut

## 154 Geschachtelte expandierbare Eigenschaften

Im Umgang mit den Eigenschaften von Fenstern und Steuerelementen ist Ihnen bestimmt schon aufgefallen, dass das `PropertyGrid` die Daten geschachtelter Eigenschaften, die selbst auf einer Klasse oder Struktur basieren (z.B. `Size`), in einer speziellen Syntax darstellt. In Abbildung 141 sehen Sie eine typische Darstellung solcher Eigenschaften, hier für ein Fenster. Neben dem Eigenschaftsnamen werden die Werte im Beispiel durch ein Semikolon voneinander getrennt. Klappt man die Eigenschaft auf, werden die einzelnen Werte dieser Eigenschaft, hier `Width` und `Height`, angezeigt.

Der Anwender kann wahlweise die aufgeklappten Eigenschaften ändern oder den zusammengesetzten Text bearbeiten. Beide Wege führen zum gleichen Ergebnis. Um für eigene Klassen auch ein derartiges Verhalten erreichen zu können, müssen Sie einen eigenen Typ-Converter programmieren. Dieser hat zwei Aufgaben:

1. Umwandlung der Daten in den Anzeigetext
2. Erstellung eines neuen Objektes aus einem eingegebenen oder geänderten Anzeigetext

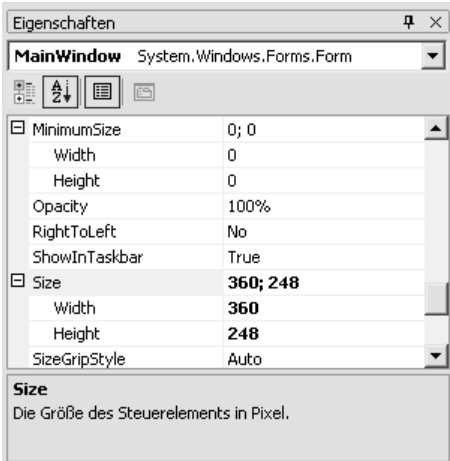


Abbildung 142: Expandierte Eigenschaften vom Typ Size

Abbildung 143 zeigt ein Beispiel für die Eigenschaften eines Angestellten. Neben dem Namen wird der benutzte Computer aufgeführt. Dieser wiederum besitzt die Eigenschaften `OperatingSystem` und `ProcessorSpeed`. Die Definition der Klassen sehen Sie in Listing 267 bzw. Listing 268. Die Eigenschaft `Computer` wird mit der Syntax `[OperatingSystem-Komma-Leerzeichen-ProcessorSpeed-Leerzeichen-MHZ]` angezeigt. Ohne Definition eines geeigneten Typ-Converters wäre die Umwandlung nicht möglich.

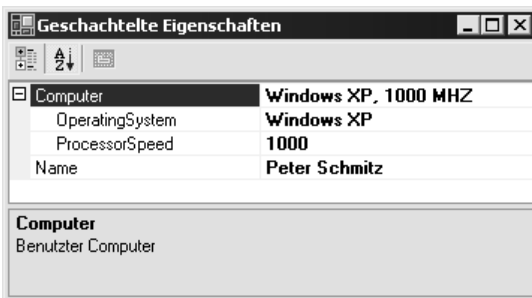


Abbildung 143: Expandieren von Eigenschaften mit Hilfe eines spezifischen Converters

---

```
Public Class Employee

    <Description("Name des Mitarbeiters")> _
```

---

Listing 267: Ein Mitarbeiter hat einen Namen und einen Computer

```
Public Property Name() As String
...
End Property

<Description("Benutzter Computer")> _
Public Property Computer() As PC
...
End Property

End Class
```

Listing 267: Ein Mitarbeiter hat einen Namen und einen Computer (Forts.)

```
<TypeConverter(GetType(PCConverter))> _
Public Class PC
    <Description("Betriebssystem")> _
    Public Property OperatingSystem() As String
    ...
End Property

    <Description("Prozessorgeschwindigkeit")> _
    Public Property ProcessorSpeed() As Integer
    ...
End Property
End Class
```

Listing 268: Definition der Klasse PC mit Angabe des Typ-Converters

Die Definition des eigenen Typ-Converters gestaltet sich allerdings recht einfach, da das Framework bereits mit `ExpandableObjectConverter` eine geeignete Basisklasse zur Verfügung stellt. Es reicht aus, die vier in Tabelle 23 beschriebenen Methoden zu überschreiben. Gebunden wird der Typ-Converter an die betreffende Klasse (hier `PC`) mit Hilfe des `TypeConverter`-Attributs (siehe Listing 268).

| Methode        | Bedeutung                                                                      |
|----------------|--------------------------------------------------------------------------------|
| CanConvertFrom | gibt an, ob basierend auf dem Anzeigetext ein neues Objekt erzeugt werden kann |
| CanConvertTo   | gibt an, ob die Eigenschaft in einen String gewandelt werden kann              |
| ConvertFrom    | Erstellung eines neuen Objektes aus den Daten im Anzeigetext                   |
| ConvertTo      | Erzeugung des Anzeigetextes                                                    |

Tabelle 23: Zu überschreibende Methoden der Klasse `ExpandableObjectConverter` für die expandierbare Ansicht im `PropertyGrid`

Wie Sie die Syntax der Eigenschaftszeile gestalten, bleibt Ihnen überlassen. Sie können die Daten nach eigenen Wünschen formatieren und zusammenstellen. Allerdings müssen Sie auch selber sicherstellen, dass sich die Daten aus dem erzeugten Text wieder extrahieren lassen. Listing 269 zeigt die Implementierung der Klasse `PCConverter`. Die beiden Methoden `CanConvertTo` und `CanConvertFrom` überprüfen, ob in oder aus einem String gewandelt werden soll. Falls nicht, wird die

Abarbeitung an die Basisklasse weitergeleitet. `ConvertTo` formatiert den String in der o.g. Syntax. `ConvertFrom` zerlegt den String und extrahiert wieder die Bezeichnung des Betriebssystems und die Prozessorgeschwindigkeit und erzeugt eine neue Instanz der Klasse `PC`.

---

```
Public Class PCConverter
    Inherits ExpandableObjectConverter

    Public Overloads Overrides Function CanConvertFrom( _
        ByVal context As System.ComponentModel.ITypeDescriptorContext, _
        ByVal sourceType As System.Type) As Boolean

        ' Konvertierung von String zulassen
        If sourceType Is GetType(String) Then Return True

        ' Sonst Delegation an die Basisklasse
        Return MyBase.CanConvertFrom(context, sourceType)

End Function

Public Overloads Overrides Function CanConvertTo( _
    ByVal context As System.ComponentModel.ITypeDescriptorContext, _
    ByVal destinationType As System.Type) As Boolean

    ' Konvertieren in einen String zulassen
    If destinationType Is GetType(String) Then Return True

    ' Sonst Delegation an die Basisklasse
    Return MyBase.CanConvertFrom(context, destinationType)

End Function

' Konvertierung in einen String
Public Overloads Overrides Function ConvertTo( _
    ByVal context As System.ComponentModel.ITypeDescriptorContext, _
    ByVal culture As System.Globalization.CultureInfo, _
    ByVal value As Object, ByVal destinationType As System.Type) _
    As Object

    If destinationType Is GetType(String) Then
        ' PC-Daten in String konvertieren
        Dim comp As PC = DirectCast(value, PC)
        Return String.Format("{0}, {1} MHZ", comp.OperatingSystem, _
            comp.ProcessorSpeed)

    Else

        ' Sonst Delegation an die Basisklasse
        Return MyBase.ConvertTo(context, culture, value, _
            destinationType)

    End If
End Function
```

---



```

Public Overloads Overrides Function ConvertFrom( _
    ByVal context As System.ComponentModel.ITypeDescriptorContext, _
    ByVal culture As System.Globalization.CultureInfo, _
    ByVal value As Object) As Object

    ' Strings konvertieren
    If TypeOf value Is String Then
        ' Trennzeichen suchen und Werte separieren
        Dim t As String = CStr(value)
        Dim pos As Integer = t.IndexOf(",")
        Dim os As String = t.Substring(0, pos)
        Dim tspeed As String = t.Substring(pos + 2)
        pos = tspeed.IndexOf(" ")
        tspeed = tspeed.Substring(0, pos)

        ' Neues Objekt anlegen und initialisieren
        Dim comp As New PC
        comp.OperatingSystem = os
        comp.ProcessorSpeed = Integer.Parse(tspeed)

        ' Das neue Objekt zurückgeben
        Return comp
    End If

    ' Sonst Delegation an die Basisklasse
    Return MyBase.ConvertFrom(context, culture, value)

End Function

End Class

```

Listing 269: Ein speziell für die Klasse PC entwickelter Converter (Forts.)

**Tipp**

Wenn Sie die betreffenden Klassen (hier die Klasse PC) selbst programmiert haben, bietet es sich an, einen Teil der Aufgaben in diese Klassen zu verlagern. Die Umwandlung der Daten in einen String können Sie mit einer Überschreibung der Methode `ToString` realisieren, die Umwandlung in der anderen Richtung durch Anlage eines geeigneten Konstruktors, der den Anzeigetext entgegennimmt und die Felder initialisiert. So können Sie diese Funktionalität auch für andere Aufgaben nutzen.

## 155 DropDown-Liste mit Standardwerten für Texteigenschaften

Wiederum über ein Attribut kann man einer Texteigenschaft eine `TypeConverter`-Klasse zuweisen, die einen Satz fest definierten Textes zur Verfügung stellt. Im Eigenschaftsfenster wird dem Anwender dann eine DropDown-Liste angeboten, aus der er einen der Werte auswählen kann. Wahlweise kann man erlauben, auch andere Texte einzugeben oder die Auswahl auf die Liste beschränken. In Abbildung 144 sehen Sie ein Beispiel, bei dem für die Eigenschaft `City` mehrere Städtenamen zur Auswahl angeboten werden, aber auch andere Texte eingetragen werden können.

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes



Abbildung 144: Text-Auswahlliste mit vorgegebenen Städtenamen

Listing 270 zeigt die Implementierung der Klasse `Location`, deren Instanzeigenschaften im `PropertyGrid` angezeigt werden, Listing 271 die Implementierung der `TypeConverter`-Klasse. Tabelle 24 führt die zu überschreibenden Methoden auf.

---

```
Public Class Location

    Protected LCity As String

    ' Eigenschaft City und Angabe des Converters
    <TypeConverter(GetType(CityStringConverter))> _
    Public Property City() As String
        Get
            Return LCity
        End Get
        Set(ByVal Value As String)
            LCity = Value
        End Set
    End Property
End Class
```

---

Listing 270: Zuweisung eines `TypeConverters` für eine Textauswahl

---

```
Public Class CityStringConverter
    Inherits StringConverter

    Private Shared defaultCities As New StandardValuesCollection _
        (New String() {"New York", "Tokio", "Berlin", _
            "Canberra", "Buenos Aires"})

    ' Rückgabe der vorgegebenen Standardwerte
    Public Overloads Overrides Function GetStandardValues( _
        ByVal context As System.ComponentModel.ITypeDescriptorContext) _
        As System.ComponentModel.TypeConverter.StandardValuesCollection

        Return defaultCities
    End Function
```

---

Listing 271: Implementierung einer `TypeConverter`-Klasse für die Bereitstellung vordefinierter Städtenamen

```
Public Overloads Overrides Function GetStandardValuesExclusive( _
    ByVal context As System.ComponentModel.ITypeDescriptorContext) _
    As Boolean

    ' Rückgabe True, wenn nur die in GetStandardValues angegebenen
    ' Werte zulässig sein sollen
    ' Rückgabe False, wenn zusätzlich beliebige Texte eingegeben
    ' werden dürfen

    Return False

End Function

Public Overloads Overrides Function GetStandardValuesSupported( _
    ByVal context As System.ComponentModel.ITypeDescriptorContext) _
    As Boolean

    ' Rückgabe True, wenn Standardwerte angezeigt werden sollen
    ' False, wenn nicht
    Return True

End Function

End Class
```

Listing 271: Implementierung einer TypeConverter-Klasse für die Bereitstellung vordefinierter Städtenamen (Forts.)

| Methode                    | Bedeutung                                                                    |
|----------------------------|------------------------------------------------------------------------------|
| GetStandardValuesSupported | True, wenn Standardwerte verwendet werden sollen                             |
| GetStandardValues          | Rückgabe einer Liste mit den vordefinierten Texten                           |
| GetStandardValuesExclusive | True, wenn die Eingabe auf die Standardwerte beschränkt ist, ansonsten False |

Tabelle 24: Zu überschreibende Methoden einer Ableitung von der Klasse StringConverter

Wie Sie dem Listing entnehmen können, besteht die Programmierung hauptsächlich daraus, die Liste aufzubauen und zurückzugeben. Die Überschreibungen der Methoden sind mit Hilfe der Entwicklungsumgebung schnell realisiert.

## 156 Visualisierung von Eigenschaftswerten mit Miniaturbildern

»Ein Bild sagt mehr als tausend Worte« rät uns ein Sprichwort. Auch Eigenschaftswerte lassen sich oft sinnvoll durch eine kleine Grafik visualisieren. In Abbildung 145 sehen Sie die grafische Darstellung einer Zeitangabe in Sekunden, die ein Tortendiagramm (oder vielleicht treffender Törtchendiagramm) zeigt, das den Anteil des Eigenschaftswertes an einer vollen Minute darstellt. Zum Zeichnen können Sie die komplette Palette der Zeichenmethoden von GDI+ verwenden, wenngleich der Raum für diese Zeichnung äußerst eingeschränkt ist und wenig Platz für Details bietet.

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Propert-  
yGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

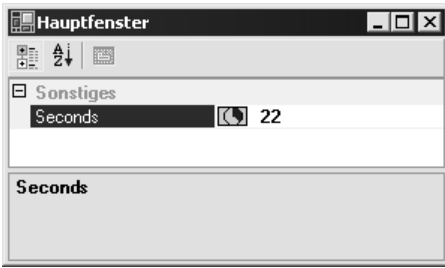


Abbildung 145: Grafische Darstellung der Eigenschaft Seconds als Miniaturbild

Zur Realisierung dieser Darstellung muss der betreffenden Eigenschaft über das `Editor`-Attribut eine von `UITypeEditor` abgeleitete Klasse zugewiesen werden. Listing 272 zeigt die Implementierung der Beispielsklasse.

---

```
Public Class VisualizedSecond

    ' Eigenschaft Seconds und Angabe des Editors
    <Editor(GetType(VisualizedSecondConverter), _
    GetType(UITypeEditor))> _
    Public Property Seconds() As Integer
    ...
End Property

End Class
```

---

Listing 272: Definition eines Editors zur Visualisierung der Eigenschaft Seconds

Der Editor selbst ist eine Ableitung der Klasse `UITypeEditor` (Listing 273). `GetPaintValueSupported` gibt `True` zurück, damit das `PropertyGrid` erfährt, dass eine Miniaturgrafik zu zeichnen ist. Ähnlich dem `Paint`-Ereignis wird `PaintValue` aufgerufen, wenn die Darstellung gezeichnet werden muss. `e.Bounds` gibt das Rechteck an, das für die Zeichenoperationen zur Verfügung steht. Nur innerhalb dieses Rechtecks sollte gezeichnet werden.

---

```
Public Class VisualizedSecondConverter
    Inherits UITypeEditor

    Public Overloads Overrides Function GetPaintValueSupported( _
        ByVal context As System.ComponentModel.ITypeDescriptorContext) _
        As Boolean

        ' Rückgabe True, damit das PropertyGrid weiß, dass etwas
        ' gezeichnet werden soll
        Return True

    End Function

    Public Overloads Overrides Sub PaintValue( _
```

---

Listing 273: Implementierung des Editors zum Zeichnen des Tortendiagramms

```

ByVal e As System.Drawing.Design.PaintValueEventArgs

' Zeichnen innerhalb von e.Bounds
' Hintergrund hellblau
e.Graphics.FillRectangle(Brushes.LightBlue, e.Bounds)

' Hilfsgrößen für Kreis ermitteln
Dim radius As Integer = Math.Min(e.Bounds.Width, _
    e.Bounds.Height) \ 2
Dim mx As Integer = e.Bounds.Width \ 2 + e.Bounds.Left
Dim my As Integer = e.Bounds.Height \ 2 + e.Bounds.Top
Dim r As New Rectangle(mx - radius, my - radius, radius * 2, _
    radius * 2)

' Kreis zeichnen
e.Graphics.DrawEllipse(Pens.Black, r)

' Kreisausschnitt dunkelblau zeichnen
e.Graphics.FillPie(Brushes.DarkBlue, r, -90, CSng(e.Value) _
    * 6.0F)

End Sub

End Class

```

Listing 273: Implementierung des Editors zum Zeichnen des Tortendiagramms (Forts.)

Im Beispiel wird ein Vollkreis gezeichnet sowie ein gefüllter Ausschnitt. Der Winkel entspricht dem Verhältnis von Eigenschaftswert zu 60 Sekunden.

Hinweis

Zur Ausgabe des Eigenschaftswertes neben der Grafik benutzt das PropertyGrid dasselbe Graphics-Objekt. Denken Sie also daran, vor dem Verlassen der PaintValue-Methode eventuell vorgenommene Änderungen wie Koordinatentransformationen wieder rückgängig zu machen.

## 157 Einen eigenen DropDown-Editor anzeigen

Manche Daten lassen sich besser über spezialisierte Steuerelemente manipulieren als über die einfachen Texteingabefelder. So ist es z.B. recht mühsam, die Zeitangaben eines TimeSpan-Wertes zu ändern, will man doch nur mal eben ein paar Sekunden oder Minuten hinzufügen oder abziehen. Um beispielsweise für eine Eieruhr eine Zeiteinstellung vorzusehen, drängt sich der Einsatz von NumericUpDown-Steuerelementen förmlich auf, bringen sie doch schon alle Voraussetzungen mit, um Zahlenwerte über Pfeiltasten zu verändern.

Im PropertyGrid können Sie eigene (kleine) Editorfenster als DropDown-Fenster anzeigen und so dem Anwender eine komfortable Dateneingabe ermöglichen. Abbildung 146 zeigt als Beispiel eine Zeiteingabe, die mit einem eigenen Editor realisiert wurde.

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Prop-  
ertyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

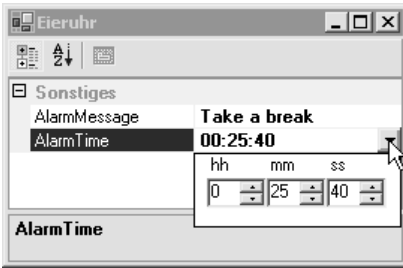


Abbildung 146: DropDown-Editor für die schnelle und komfortable Eingabe von Zeiten

Auch ein solcher Editor ist schnell erstellt und eingebunden. Zunächst wird ein benutzerdefinierteres Steuerelement benötigt, das die drei `NumericUpDown`-Controls sowie die Labels für die Beschriftung enthält. In Listing 274 sehen Sie den Aufbau, wie er im Designer angezeigt wird. In der Klasse dieses Controls (`TimeEditorControl`, siehe Abbildung 147) wird zusätzlich eine Eigenschaft eingerichtet, um die Alarmzeit des Objektes in die `NumericUpDown`-Felder einzutragen und vice versa.

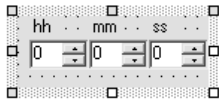


Abbildung 147: Der Editor basiert auf einem Benutzersteuerelement

---

```
Public Class TimeEditorControl
    Inherits System.Windows.Forms.UserControl

    #Region " Vom Windows Form Designer generierter Code "
    ...

    ' Transfer der Zeiteinstellung <-> Eingabefelder
    Public Property AlarmTime() As TimeSpan
        Get
            ' Wert aus Eingabefeldern zusammensetzen
            Return New TimeSpan(CInt(NUDHour.Value), _
                                CInt(NUDMinute.Value), CInt(NUDSecond.Value))
        End Get

        Set(ByVal Value As TimeSpan)
            ' Eingabefelder setzen
            NUDHour.Value = Value.Hours
            NUDMinute.Value = Value.Minutes
            NUDSecond.Value = Value.Seconds
        End Set
    End Property

End Class
```

---

Listing 274: Ergänzung des Editor-Controls um die Eigenschaft `AlarmTime`

Um dem PropertyGrid über ein Attribut das Vorhandensein eines DropDown-Editors bekannt machen zu können, muss eine von UITypedEditor abgeleitete Klasse (hier AlarmTimeUITypedEditor, siehe Listing 275) angelegt werden. Zwei Methoden werden überschrieben:

1. GetEditStyle, um den Editor als DropDown-Editor anzumelden, und
2. EditValue, um den ausgewählten Wert im Editor anzuzeigen.

GetEditStyle überprüft lediglich den Datentyp und gibt den Enumerationswert UITypedEditorEditStyle.DropDown zurück. Das PropertyGrid-Steuerelement erfährt hierüber, dass es den Pfeil zum Aufklappen eines DropDown-Feldes anzeigen soll.

Entscheidet sich der Anwender, auf diesen Pfeil zu klicken, ruft das PropertyGrid-Control die Methode EditValue auf. Der Parameter context enthält Informationen zum Objekt, dessen Eigenschaften dargestellt werden, der Parameter value den aktuellen Wert der zu ändernden Eigenschaft.

Benötigt wird eine Instanz des zuvor beschriebenen Benutzersteuerelementes TimeEditorControl, die entweder neu angelegt oder wieder verwendet werden kann. Über die o.g. Eigenschaft AlarmTime wird die bislang eingestellte Zeit in die NumericUpDown-Felder übertragen. Anschließend wird über den Parameter provider mit Hilfe der Methode DropDownControl der Editor angezeigt. Der Aufruf erfolgt modal und ist erst beendet, wenn das Editor-Fenster wieder zugeklappt worden ist. Danach werden die Daten zurückgelesen und als Funktionswert zurückgegeben.

```
Imports System.ComponentModel
Imports System.Drawing.Design
Imports System.Windows.Forms.Design

Public Class AlarmTimeUITypedEditor
    Inherits UITypedEditor
    Private Clock As AlarmClock
    Private TimeEditor As TimeEditorControl

    Public Overloads Overrides Function EditValue( _
        ByVal context As System.ComponentModel.ITypeDescriptorContext, _
        ByVal provider As System.IServiceProvider, _
        ByVal value As Object) As Object

        ' EditorService zum Aufklappen des Editors ermitteln
        Dim wfes As IWindowsFormsEditorService
        wfes = DirectCast(provider.GetService(GetType( _
            IWindowsFormsEditorService)), IWindowsFormsEditorService)

        ' Instanz des Editorcontrols verwenden oder neu anlegen
        If TimeEditor Is Nothing Then
            TimeEditor = New TimeEditorControl
        End If

        ' Zeit in Steuerelemente übertragen
        TimeEditor.AlarmTime = CType(value, TimeSpan)
```

Listing 275: Von UITypedEditor abgeleitete Klasse AlarmTimeUITypedEditor zur Steuerung der Anzeige des DropDown-Editors

---

```

' EditorControl als DropDownControl anzeigen
wfes.DropDownControl(TimeEditor)

' Werte aus Steuerelementen zurücklesen und weiterreichen
Return TimeEditor.AlarmTime

```

End Function

```

Public Overloads Overrides Function GetEditStyle( _
    ByVal context As System.ComponentModel.ITypeDescriptorContext) _
    As System.Drawing.Design.UITypeEditorEditStyle

' DropDown-Control nur für unsere AlarmClock
If TypeOf context.Instance Is AlarmClock Then
    Return UITypeEditorEditStyle.DropDown
Else
    Return MyBase.GetEditStyle(context)
End If

```

End Function

End Class

---

*Listing 275: Von UITypeEditor abgeleitete Klasse AlarmTimeUIEditor zur Steuerung der Anzeige des DropDown-Editors (Forts.)*

Im letzten Schritt wird über das Editor-Attribut der eigentlichen Datenklasse (AlarmClock) der Editor bekannt gemacht. So erfährt das PropertyGrid-Control, dass es für die Eigenschaft AlarmTime einen passenden Editor gibt, und kann ihn aufrufen. Listing 276 zeigt die Implementierung.

---

```

Public Class AlarmClock
    Protected AMessage As String
    Public Property AlarmMessage() As String
        Get
            Return AMessage
        End Get
        Set(ByVal Value As String)
            AMessage = Value
        End Set
    End Property

    Protected ATime As TimeSpan
    <Editor(GetType(AlarmTimeUIEditor), GetType(UITypeEditor))> _
    Public Property AlarmTime() As TimeSpan
        Get
            Return ATime
        End Get
        Set(ByVal Value As TimeSpan)
            ATime = Value

```

---

*Listing 276: In der Klasse AlarmClock wird über das Editor-Attribut der eigene DropDown-Editor bekannt gegeben.*



```
End Set
End Property
End Class
```

Listing 276: In der Klasse AlarmClock wird über das Editor-Attribut der eigene DropDown-Editor bekannt gegeben. (Forts.)

## 158 Eigenschaften über einen modalen Dialog bearbeiten

Sind die Informationen einer Eigenschaft zu komplex, um sie im PropertyGrid direkt bearbeiten zu können, und ist der Raum für die Darstellung in einem DropDown-Editor nicht ausreichend, dann können Sie auch einen eigenen Dialog zur Verfügung stellen. Der Dialog kann beliebig aufgebaut sein und alle Daten übersichtlich präsentieren. Wählt der Anwender im Eigenschaftsfenster eine Eigenschaft aus, der ein modaler Dialog zugeordnet ist, dann wird in der betreffenden Zeile ganz rechts eine Taste mit drei Punkten angezeigt (siehe Abbildung 148). Sie kennen dieses Verhalten bereits von Eigenschaften wie BackgroundImage oder Font.



Abbildung 148: Für komplexe Eigenschaften wird eine Taste angezeigt, über die ein modaler Dialog geöffnet werden kann

Im Beispiel werden die Eigenschaften eines Containerschiffs dargestellt, bestehend aus Schiffsname, max. Anzahl Container, Kapitän und 1. Offizier. Die Eigenschaften Captain und FirstOfficer sind vom Typ Person, einer Klasse, die nähere Spezifikationen der Seeleute zulässt. Für die Darstellung einer Person wird ein eigener Editor eingesetzt. Klickt der Anwender auf die Taste mit der Ellipse, wird dieser Dialog automatisch geöffnet (Abbildung 149).

Die Vorgehensweise, um anwendungsspezifische modale Dialoge bereitzustellen, ist ähnlich der im vorangegangenen Beispiel. Betrachten wir zunächst die Datenklasse ContainerShip (Listing 277). Sie enthält alle Eigenschaften, die im PropertyGrid angezeigt werden, jeweils versehen mit Beschreibungsattributen. Weitere Attribute sind nicht erforderlich. Beachten Sie, dass die Eigenschaften Captain und FirstOfficer vom Typ Person sind.

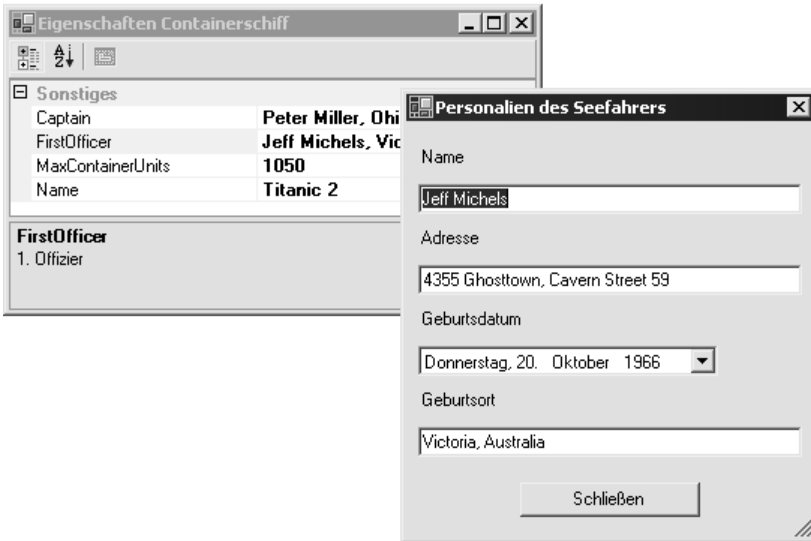


Abbildung 149: Anzeige eines modalen Dialogs zur Einstellung der Eigenschaften des 1. Offiziers

```
Public Class ContainerShip

    Public Sub New(ByVal name As String,...)
        ...
    End Sub

    <Description("Name des Schiffs")> _
    Public Property Name() As String
        ...
    End Property

    <Description("Anzahl der Containerplätze")> _
    Public Property MaxContainerUnits() As Integer
        ...
    End Property

    <Description("Kapitän")> _
    Public Property Captain() As Person
        ...
    End Property

    <Description("1. Offizier")> _
    Public Property FirstOfficer() As Person
        ...
    End Property

End Class
```

Listing 277: Datenklasse ContainerShip

In der Klasse `Person` werden die Seeleute näher spezifiziert (Listing 278). Das `Editor`-Attribut wird hier der Klasse zugeordnet, damit der Mechanismus zum Aufruf des Dialoges für jede Eigenschaft vom Typ `Person` automatisch zur Verfügung steht.

```
<Editor(GetType(PersonUIEditor), GetType(UITypeEditor))> _
Public Class Person

    Public Sub New(ByVal name As String, ...)
    End Sub

    Public Overrides Function ToString() As String
        Return PName & ", " & PStateBorn
    End Function

    Public Property Name() As String
    ...
    End Property

    Public Property Address() As String
    ...
    End Property

    Public Property Birthday() As DateTime
    ...
    End Property

    Public Property StateBorn() As String
    ...
    End Property
End Class
```

Listing 278: Klasse `Person` mit zugewiesenem `Editor` vom Typ `PersonUIEditor`

Der Texteintrag, der für die Personen im Eigenschaftsfenster angezeigt wird, wird durch Aufruf der Methode `ToString` ermittelt. Die Überschriftung dieser Methode in der Klasse `Person` generiert den Anzeigetext, bestehend aus Name und Geburtsort. Ohne weitere Maßnahmen ist der Eintrag schreibgeschützt, d.h. die Daten können nur über den zusätzlichen Editor geändert werden, nicht jedoch durch Änderung des Textes.

Wie bereits erwähnt, können Sie den Dialog nach eigenen Vorstellungen gestalten. Hier wird ein Dialogfenster erstellt, das die vier Personeneigenschaften in TextBoxen und einem `DateTimePicker` darstellen kann (Abbildung 150). Eine Taste **SCHLIEßEN** steht zum Beenden des Dialogs, verbunden mit der Übernahme der Daten zur Verfügung.

Auch die Art der Datenübergabe an den Dialog und zurück können Sie frei wählen. Hier wird ein zweiter Konstruktor bereitgestellt, der die Referenz des Personen-Objektes übernimmt, in der Membervariable `Sailor` speichert und die Steuerelemente initialisiert. Denken Sie bei einer solchen Variante unbedingt daran, den parameterlosen Konstruktor oder `InitializeComponent` direkt aufzurufen, damit die Steuerelemente überhaupt angelegt werden.

Wird der Dialog über die **SCHLIEßEN** -Taste geschlossen, dann werden die Daten aus den Steuerelementen wieder in das Personen-Objekt übertragen. Im Gegensatz zum vorigen Beispiel werden die Daten des bestehenden Objektes verändert, statt ein neues zu erzeugen. Daher wird auch die

ursprüngliche Referenz wieder als Funktionswert zurückgegeben. Schließt der Anwender den Dialog auf andere Weise, werden die ursprünglichen Daten beibehalten.



Abbildung 150: Erstellung des Dialogfensters

```
Public Class SailorEditor
    Inherits System.Windows.Forms.Form

    #Region " Vom Windows Form Designer generierter Code "
    ...

    ' Referenz des zu ändernden Datenobjektes
    Protected Sailor As Person

    ' Konstruktor zur Übernahme der Daten
    Public Sub New(ByVal sailor As Person)
        MyClass.New()

        Me.Sailor = sailor
        TBName.Text = sailor.Name
        TBAddress.Text = sailor.Address
        TBStateBorn.Text = sailor.StateBorn
        DTPBirthday.Value = sailor.Birthday
    End Sub

    ' Rückübertragung der Daten und Schließen des Fensters
    Private Sub BTNOK_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles BTNOK.Click

        Sailor.Name = TBName.Text
        Sailor.Address = TBAddress.Text
        Sailor.StateBorn = TBStateBorn.Text
        Sailor.Birthday = DTPBirthday.Value
    End Sub
End Class
```

Listing 279: Zusätzlicher Code zum Instanzieren und Schließen des Editors

```

    Me.Close()
    Me.Dispose()
End Sub

```

```
End Class
```

*Listing 279: Zusätzlicher Code zum Instanzieren und Schließen des Editors (Forts.)*

Der `UITypeEditor`, der vom `PropertyGrid-Control` instanziiert wird, wenn eine Eigenschaft angezeigt werden soll, der das entsprechende Editor-Attribut zugeordnet worden ist, ist sehr ähnlich aufgebaut wie der im vorangegangenen Beispiel (siehe Listing 275). Listing 280 zeigt die Implementierung. `GetEditStyle` liefert hier den Enumerationswert `UITypeEditorEditStyle.Modal` zurück, so dass das `PropertyGrid` veranlasst wird, die Taste mit der Ellipse anzuzeigen.

Klickt der Anwender auf die Taste, wird die Methode `EditValue` aufgerufen. In `EditValue` wird der Dialog instanziiert und modal angezeigt. Nach Schließen des Dialogs wird die Referenz der Daten vom Typ `Person` zurückgegeben.

```

Imports System.ComponentModel
Imports System.Drawing.Design
Imports System.Windows.Forms.Design

Public Class PersonUIEditor
    Inherits UITypeEditor

    Public Overloads Overrides Function GetEditStyle( _
        ByVal context As System.ComponentModel.ITypeDescriptorContext) _
        As System.Drawing.Design.UITypeEditorEditStyle

        ' Modaler Dialog verfügbar
        Return UITypeEditorEditStyle.Modal
    End Function

    Public Overloads Overrides Function EditValue( _
        ByVal context As System.ComponentModel.ITypeDescriptorContext, _
        ByVal provider As System.IServiceProvider, _
        ByVal value As Object) As Object

        ' EditorService zum Anzeigen des Editors ermitteln
        Dim wfes As IWindowsFormsEditorService
        wfes = DirectCast(provider.GetService(GetType( _
            IWindowsFormsEditorService)), IWindowsFormsEditorService)

        ' Die Daten sind vom Typ Personenobjekt
        Dim sailor As Person = DirectCast(value, Person)

        ' Dialoginstanz anlegen
        Dim dialog As New SailorEditor(sailor)

        ' Dialog anzeigen

```

*Listing 280: Ableitung der Klasse `UITypeEditor` zur Bereitstellung des spezifischen Dialogs*

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Prop-  
ertyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

---

```
wfes.ShowDialog(dialog)

' Referenz des evtl. geänderten Objektes zurückgeben
Return sailor
```

```
End Function
```

```
End Class
```

---

*Listing 280: Ableitung der Klasse UITypeEditor zur Bereitstellung des spezifischen Dialogs (Forts.)*

## 159 Datei öffnen-Dialog für Eigenschaften bereitstellen

Während der Datei öffnen-Dialog für manche Datentypen wie Bitmap und Image automatisch als modaler Dialog (siehe auch Rezept [8.4](#) ((Standardwerte für Eigenschaften))) zur Verfügung gestellt wird, müssen Sie für Dateien mit eigenen Dateierweiterungen ein paar Kleinigkeiten ergänzen. Wieder wird auch hier über ein Attribut der betreffenden Eigenschaft ein Editor zugewiesen (Listing 281).

---

```
<System.ComponentModel.Editor(GetType(XYZFilenameEditor), _
GetType(System.Drawing.Design.UITypeEditor))> _
Public Property InfoFile() As String
Get
Return IInfoFile
End Get
Set(ByVal Value As String)
IInfoFile = Value
End Set
End Property
```

---

*Listing 281: Bindung der XYZFilenameEditor-Klasse an die Eigenschaft InfoFile*

Für den Vorgang gibt es bereits eine passende Basisklasse (FilenameEditor). Prinzipiell könnten Sie diese Klasse direkt verwenden, hätten dann aber keine Möglichkeit, die Dateien nach ihren Typen zu filtern. Stattdessen leiten Sie eine spezialisierte Klasse, hier XYZFilenameEditor, von dieser ab (Listing 282).

---

```
Public Class XYZFilenameEditor
Inherits System.Windows.Forms.Design.FileNameEditor

Protected Overrides Sub InitializeDialog( _
ByVal openFileDialog As System.Windows.Forms.OpenFileDialog)

' Filter für eigene Dateierweiterungen einstellen
openFileDialog.Filter = "XYZ-Dateien (*.xyz) |*.xyz"

End Sub

End Class
```

---

*Listing 282: XYZFilenameEditor filtert die Dateien vom Typ XYZ heraus*

Der Editor ist voll funktionsfähig, auch ohne dass Sie eine Methode überschreiben. Um den Dateifilter zu konfigurieren, müssen Sie jedoch die Methode `InitializeDialog` überschreiben. Sie liefert als Parameter eine Referenz des anzuzeigenden Datei-Öffnen-Dialogs. Hierüber können Sie Ihre benötigten Einstellungen, im Beispiel die Filterung von XYZ-Dateien, vornehmen. Abbildung 151 zeigt ein Beispiel für das Eigenschaftsfenster und den angezeigten Dialog.

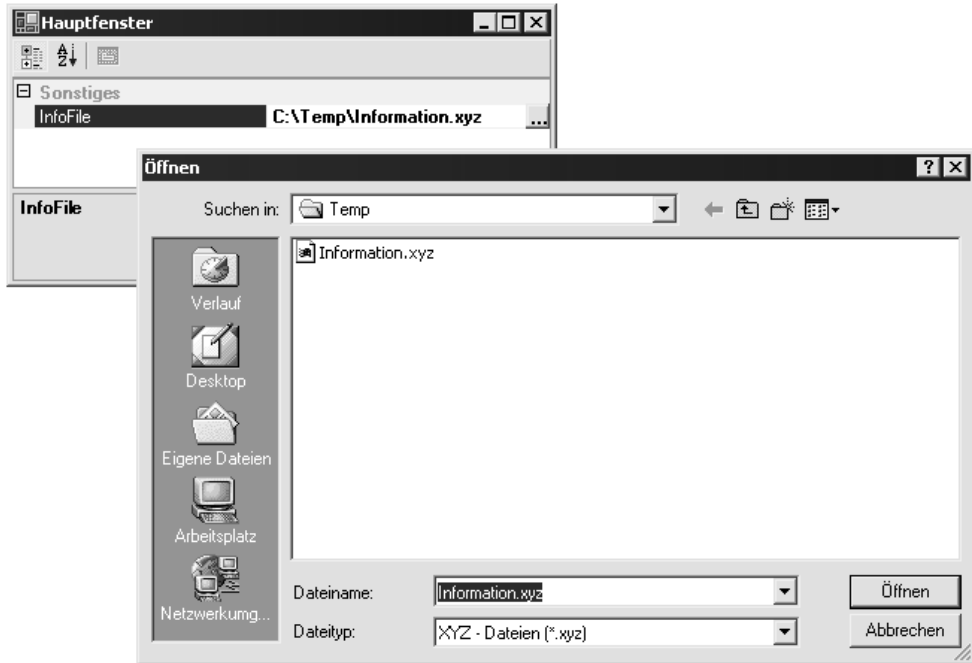


Abbildung 151: OpenFile-Dialog an Eigenschaft koppeln

## 160 Auflistungen anzeigen und bearbeiten

Wenn Sie die Items-Auflistung eines `ListView`-Steuerelementes öffnen, wird ein spezialisierter Editor angezeigt (Abbildung 152), der folgende Bedienelemente enthält:

1. Liste aller Elemente der Auflistung mit der Möglichkeit, ein einzelnes Element auszuwählen
2. Anzeige aller Eigenschaften eines ausgewählten Elements
3. Taste HINZUFÜGEN, um neue Objekte der Auflistung hinzuzufügen
4. Taste ENTFERNEN, um das ausgewählte Element zu löschen

In diesem Rezept zeigen wir, wie Sie auch für eigene Auflistungen einen angepassten Editor anzeigen und so dem Anwender eine einfache Möglichkeit an die Hand geben können, die Liste und die enthaltenen Daten zu bearbeiten. Zunächst die Erläuterung der verwendeten Beispielklassen für die Daten:

Die Klasse `Person` definiert einige allgemeine Eigenschaften einer Person (Listing 283, die Implementierung der Methoden wurde ausgelassen). Zwei abgeleitete Klassen (Listing 284), `Controller` und `Developer`, definieren die spezialisierten Eigenschaften `ControlledProject` bzw. `OperatingSystem`.

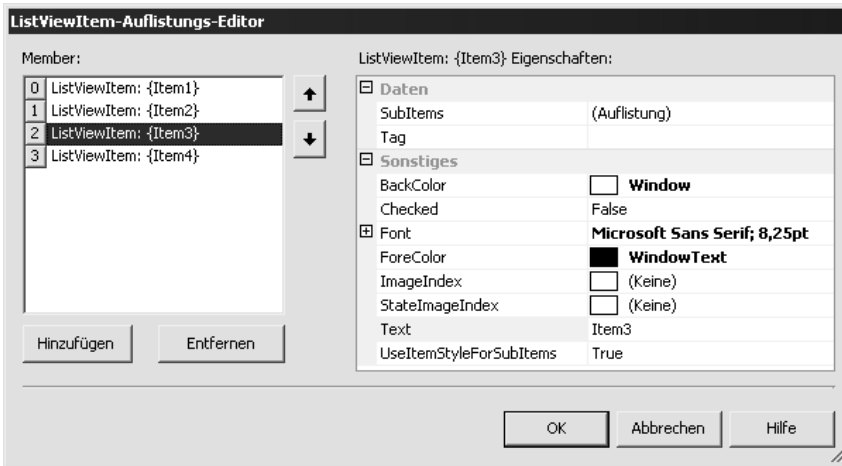


Abbildung 152: Auflistungseitor für ListViewItem-Objekte

---

```
Public Class Person

    Public Sub New()
    End Sub
    Public Sub New(...)
    End Sub

    Public Overrides Function ToString() As String
        Return PName & ", " & PStateBorn
    End Function

    Public Property Address() As String
    ...
    End Property
    Public Property FullName() As String
    ...
    End Property

    Public Property Birthday() As DateTime
    ...
    End Property
    Public Property StateBorn() As String
    ...
    End Property
End Class
```

---

Listing 283: Klasse Person

---

```
Public Class Controller
    Inherits Person
```

---

Listing 284: Abgeleitete Klassen Controller und Developer



```

Public Sub New()
End Sub

Public Sub New(...)
...
End Sub

Public Overrides Function ToString() As String
    Return Me.FullName & "(Controller)"
End Function

Public Property ControlledProject() As String
...
End Property
End Class

Public Class Developer
    Inherits Person

    Public Sub New()
    End Sub

    Public Sub New(...)
    ...
    End Sub

    Public Overrides Function ToString() As String
        Return Me.FullName & "(Developer)"
    End Function

    Public Property Operatingsystem() As String
    ...
    End Property
End Class

```

*Listing 284: Abgeleitete Klassen Controller und Developer (Forts.)*

Eine Klasse `Company` definiert als Eigenschaft eine Auflistung (`UntypedEmployeeList1`), der Instanzen der o.g. Klassen hinzugefügt werden (Listing 285). Als Liste kommt ein `ArrayList` zum Einsatz.

```

Public Class Company

    Protected CUTEmployees1 As New ArrayList

    <System.ComponentModel.Description( _
        "Untypisierte Angestelltenliste")> _
    Public Property UntypedEmployeeList1() As ArrayList
        Get
            Return CUTEmployees1
        End Get
    End Property
End Class

```

*Listing 285: Mitarbeiterliste in Form eines ArrayList-Objektes*

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Prop-  
ertyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

```

Set(ByVal Value As ArrayList)
    CUTEmployees1 = Value
End Set
End Property

```

```
End Class
```

*Listing 285: Mitarbeiterliste in Form eines ArrayList-Objektes (Forts.)*

Im Hauptprogramm wird eine Instanz der Klasse `Company` erzeugt und der Liste Instanzen von `Person`, `Controller` und `Developer` hinzugefügt. Die Referenz des `Company`-Objektes wird der `SelectedObject`-Eigenschaft des `PropertyGrid`-Controls zugewiesen:

```

Dim comp As New Company
Dim p As Person
p = New Person("Bill Watson", "Miami", _
    New DateTime(1933, 3, 4), "Florida, USA")
comp.UntypedEmployeeList1.Add(p)
p = New Developer("Karin Meier", "Köln", _
    New DateTime(1973, 7, 30), "München, Deutschland", "XP")
comp.UntypedEmployeeList1.Add(p)
p = New Controller("Pierre Renault", "Paris", _
    New DateTime(1966, 2, 4), "Marseille, France", "Metro 2005")
comp.UntypedEmployeeList1.Add(p)
PGTest.SelectedObject = comp

```

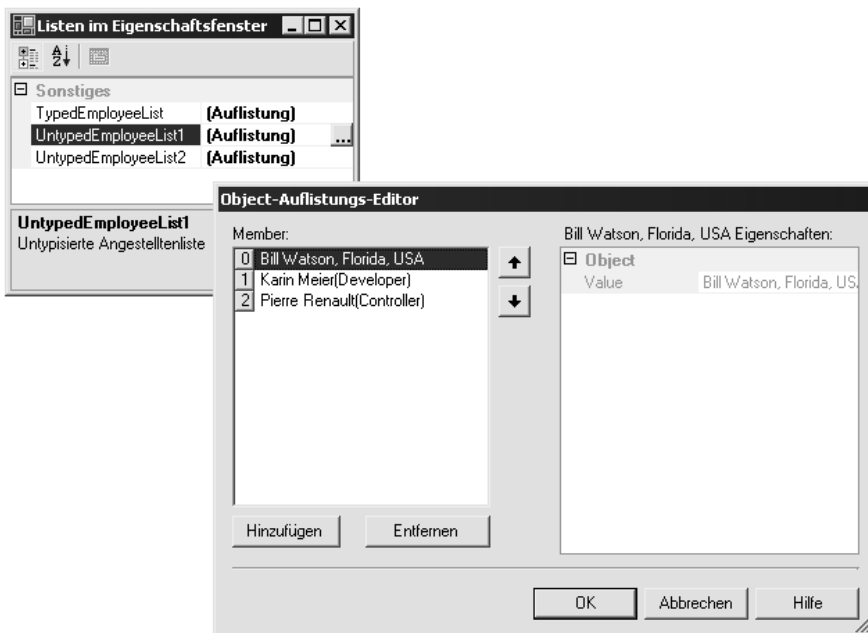


Abbildung 153: Eine `ArrayList`-Auflistung wird standardmäßig mit dem `Object-Auflistungs-Editor` angezeigt

Das Eigenschaftsfenster zeigt die Eigenschaft `UntypedEmployeeList1` als Auflistung an und bietet auch bereits die Möglichkeit, einen Auflistungs-Editor zu öffnen (Abbildung 153). Leider ist dieser erste Ansatz recht unbefriedigend, da dieser Editor nur mit `Object`-Referenzen umgehen kann und weder die Daten anzeigt, noch eine Möglichkeit zum Hinzufügen weiterer Personen-Objekte bietet.

Mit den folgenden zwei Lösungsansätzen zeigen wir Ihnen, wie Sie erreichen können, dass der Auflistungs-Editor die Eigenschaften richtig anzeigt und das Hinzufügen neuer Personen unterstützt.

### Einsatz einer typisierten Liste

Das zuvor eingesetzte `ArrayList` als Auflistungstyp verwaltet alle Objekte über `Object`-Referenzen. Somit hat auch der Editor keinerlei Typ-Informationen zu den aufgelisteten Objekten. Setzt man statt des `ArrayList`-Objektes eine typisierte Liste ein, die nur Objekte vom Typ `Person` aufnehmen kann, dann hat der Editor alle Informationen, die er für den Umgang mit Instanzen der Klasse `Person` benötigt. Listing 286 zeigt den exemplarischen Aufbau der typisierten Liste.

```
Public Class EmployeeList
    Inherits System.Collections.CollectionBase

    Public Sub Add(ByVal employee As Person)
        list.Add(employee)
    End Sub

    Public Sub Remove(ByVal index As Integer)
        If index > Count - 1 Or index < 0 Then
            System.Windows.Forms.MessageBox.Show("Index Not valid!")
        Else
            list.RemoveAt(index)
        End If
    End Sub

    Public ReadOnly Property Item(ByVal index As Integer) As Person
        Get
            Return CType(list.Item(index), Person)
        End Get
    End Property

End Class
```

Listing 286: Typisierte Liste für die Mitarbeiter-Auflistung

In dieser Klasse wird die `Add`-Methode so definiert, dass nur Referenzen vom Typ `Person` hinzugefügt werden können. Zusätzlich müssen die Methode `Remove` und die Eigenschaft `Item` implementiert werden. Der Klasse `Company` wird eine Eigenschaft `TypedEmployeeList` vom Typ `EmployeeList` hinzugefügt (Listing 287).

```
Protected CEmployees As New EmployeeList

<System.ComponentModel.Description( _
```

Listing 287: Typisierte Liste als Member-Variable von `Company`

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Proper-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

```

"Typisierte Angestelltenliste"> _
Public Property TypedEmployeeList() As EmployeeList
    Get
        Return CEmployees
    End Get
    Set(ByVal Value As EmployeeList)
        CEmployees = Value
    End Set
End Property

```

Listing 287: Typisierte Liste als Member-Variable von Company (Forts.)

Dieser Auflistung werden im Hauptprogramm die gleichen Instanzen hinzugefügt wie bereits der anderen Auflistung. Abbildung 154 zeigt das Ergebnis. Der Aufzählungs-Editor zeigt die Objekte korrekt an. Auch die Eigenschaften eines ausgewählten Objekts werden richtig dargestellt und können verändert werden. Einziger Wermutstropfen ist, dass über die Taste HINZUFÜGEN nur Instanzen der Klasse `Person`, nicht jedoch von `Controller` oder `Developer` erzeugt werden können.

Sofern Sie nicht mit Vererbungen arbeiten, sondern nur einen einzigen Datentyp in der Liste führen wollen, ist diese Lösung durchaus eine Überlegung wert, bietet sie doch auch andere Vorteile, wie den typsicheren Umgang mit der Liste beim Hinzufügen von Elementen sowie beim Zugriff über einen Index. Ist es für Sie jedoch wichtig, auch Instanzen anderer Typen hinzufügen zu können, oder haben Sie keine Möglichkeit, eine allgemeine durch eine typisierte Liste zu ersetzen, dann bietet sich die nachfolgend beschriebene Lösung an.

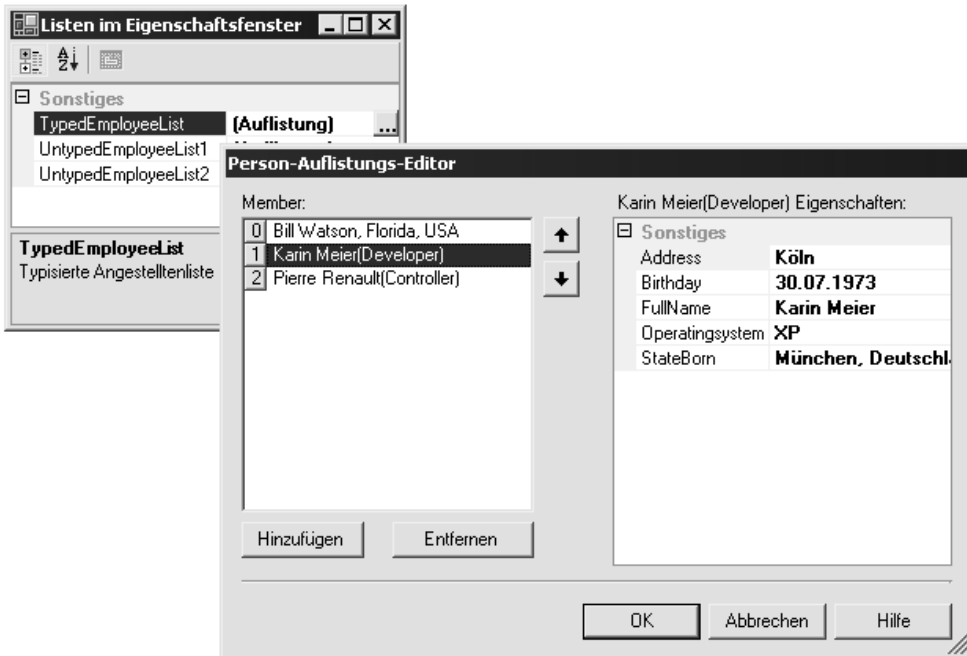


Abbildung 154: Typisierte Liste: Der Auflistungseitor zeigt die Eigenschaften korrekt an, bietet aber nur die Möglichkeit, Instanzen der Klasse `Person` hinzuzufügen

## Definition eines speziellen UIType-Editors für die eingesetzten Datentypen

Sie hatten es wahrscheinlich nicht anders vermutet: Auch hier kommen wieder Attribute zum Einsatz. Der Eigenschaft `UntypedEmployeeList2` (wieder vom Typ `ArrayList`) wird über das `Editor`-Attribut der spezialisierte Editor vom Typ `EmployeeListEditor` zugewiesen (Listing 288).

```
<System.ComponentModel.Description( _
    "Untypisierte Angestelltenliste mit EmployeeListEditor"), _
System.ComponentModel.Editor(GetType(EmployeeListEditor), _
GetType(System.Drawing.Design.UITypeEditor))> _
Public Property UntypedEmployeeList2() As ArrayList
    Get
        Return CUTEmployees2
    End Get
    Set(ByVal Value As ArrayList)
        CUTEmployees2 = Value
    End Set
End Property
```

*Listing 288: Eigenschaft `UntypedEmployeeList2` der Klasse `Company` mit Angabe des zu verwendenden Editors*

Der angegebene Editor ist eine Ableitung der Klasse `CollectionEditor`, die ihrerseits von der bereits beschriebenen Klasse `UITypeEditor` abgeleitet ist (Listing 289). Tabelle 25 zeigt die zu implementierenden Methoden.

```
Public Class EmployeeListEditor
    Inherits System.ComponentModel.Design.CollectionEditor

    Public Sub New(ByVal type As Type)
        MyBase.New(type)
    End Sub

    Protected Overrides Function CreateNewItemTypes() As System.Type()
        Return New Type() {GetType(Developer), GetType(Controller)}
    End Function

    Protected Overrides Function CreateCollectionItemType() _
        As System.Type

        Return GetType(Person)
    End Function

End Class
```

*Listing 289: Die Klasse `EmployeeListEditor` implementiert die Methoden, die der Editor des `PropertyGrids` für den Umgang mit unseren Datentypen benötigt.*

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Proper-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

| Methode                          | Bedeutung                                                                                                                                                                          |
|----------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| New(type)                        | Konstruktor, der ein Type-Objekt annimmt und an den Konstruktor der Basisklasse weiterleitet                                                                                       |
| CreateNewItemTypes As Type()     | Rückgabe der Typ-Objekte aller Datentypen, die über die Taste HINZUFÜGEN instanziiert werden sollen. Wird diese Methode nicht überschrieben, steht nur der Basistyp zur Verfügung. |
| CreateCollectionItemType As Type | Zu verwendender Basistyp                                                                                                                                                           |

Tabelle 25: Zu implementierende Methoden der Klasse *CollectionEditor*

Diese wenigen Definitionen reichen aus, um das in Abbildung 155 gezeigte gewünschte Ergebnis zu erzielen. Die HINZUFÜGEN-Taste bietet jetzt eine DropDown-Auswahl für die verwendbaren Datentypen. Mit dieser Variante können Sie den Auflistungs-Editor gestalten, ohne die eigentliche Liste verändern zu müssen.

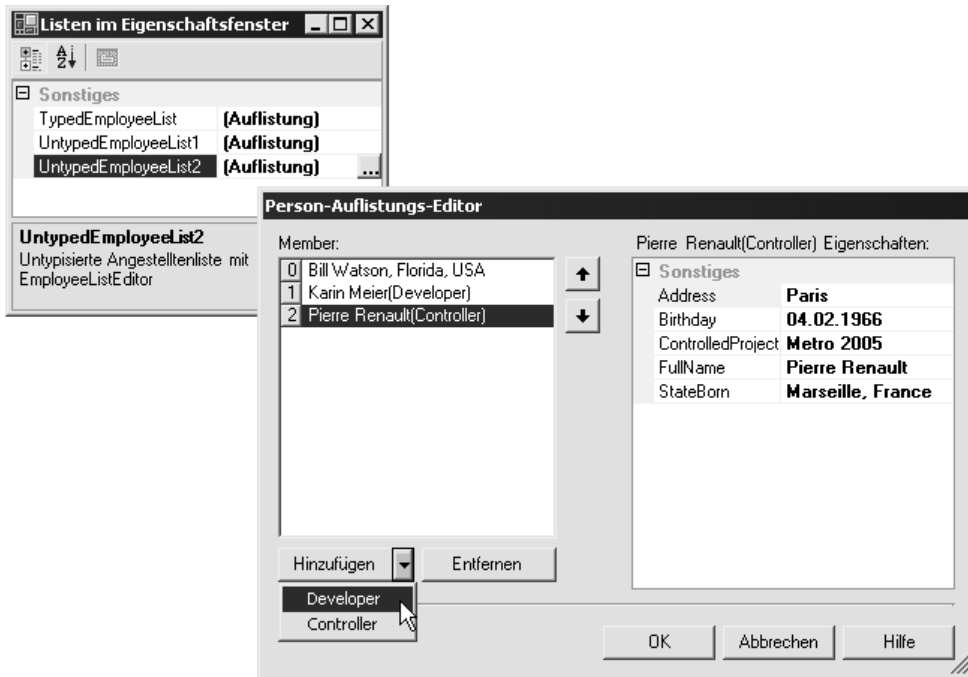


Abbildung 155: Korrekte Anzeige der Eigenschaften und Auswahl des Typs hinzuzufügender Objekte durch Bereitstellung von *EmployeeListEditor*

## Hinweis

Ein Kuriosum, dem ebenfalls Aufmerksamkeit geschenkt werden sollte, ist die Quelle, aus der die Informationen für die Bezeichnungen der in der linken Liste aufgeführten Elemente bezogen werden. Wie Sie vielleicht bemerkt haben, werden die Texte der obigen Beispiele mit Hilfe der `ToString`-Methode abgefragt. Das gilt allerdings nur, solange es keine Eigenschaft mit dem Namen `Name` gibt. Existiert eine solche Eigenschaft, wird sie zur Textbestimmung herangezogen.

## 161 Aktionen über Hyperlink-Tasten anbieten

Einige Steuerelemente wie z.B. das `DataGrid`- und das `TabControl` bieten im Kommandobereich des `Property-Browsers` `Hyperlink`-Schaltflächen an, über die der Anwender direkt mehr oder weniger komplexe Methoden aufrufen kann. Solche Schaltflächen sind dann besonders sinnvoll, wenn mehrere Eigenschaften gleichzeitig verändert werden sollen. Die Aktionstasten sind nicht an eine einzelne Eigenschaft gebunden, sondern stehen zur Verfügung, nachdem das Steuerelement ausgewählt worden ist.

Dieses Rezept ist die in der Kapitel-Einleitung erwähnte Ausnahme, was den Aufbau des Beispiels betrifft. Die Darstellung der `Hyperlink`-Tasten funktioniert (zumindest ohne großen Aufwand) leider nur im `Design-Modus`. Daher wird die Vorgehensweise an einem benutzerdefinierten Steuerelement, das im `Designer` der Entwicklungsumgebung bearbeitet wird, demonstriert.

Abbildung 156 zeigt die Bearbeitung des Steuerelements im `Designer`. Das Steuerelement gruppiert mehrere `CheckBox`s, über die eine Auswahl für eine gewünschte Fahrzeugausstattung getroffen werden kann. Wird das Steuerelement ausgewählt, zeigt das `PropertyGrid` zusätzlich die `Hyperlink`-Schaltflächen `STANDARD`, `LUXUS` und `SPECIAL OFFER` an. Ein Klick auf eine solche Schaltfläche genügt, um direkt alle `CheckBox`s gemäß einer vorgegebenen Kombination zu setzen bzw. zu löschen.

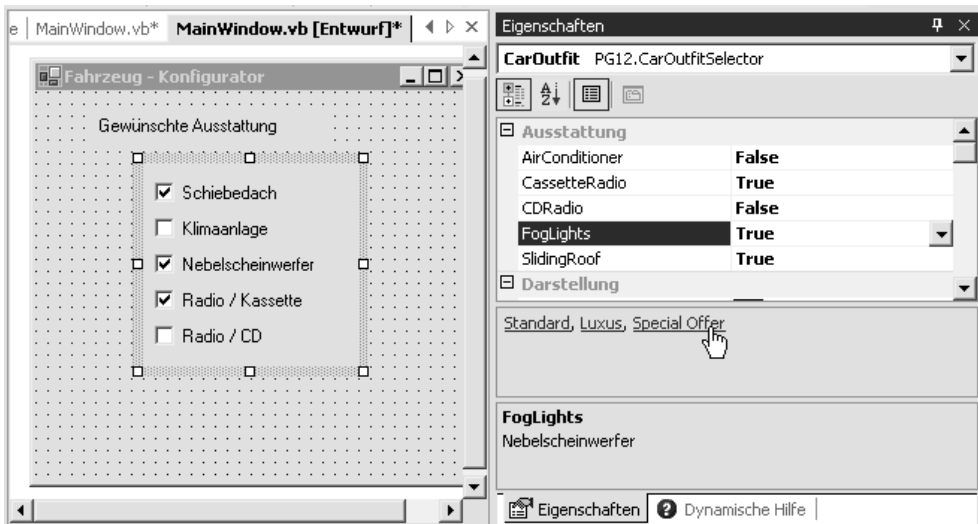


Abbildung 156: Hyperlink-Tasten für Aktionen anbieten

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Propor-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

Listing 290 zeigt zunächst den grundlegenden Aufbau des Steuerelementes (CarOutfitSelector). Für jede CheckBox werden Property-Set- und Get-Methoden definiert, die direkt auf die jeweilige Checked-Eigenschaft zugreifen. Diese Eigenschaften werden im Eigenschaftsfenster in der Kategorie AUSSTATTUNG angezeigt.

---

```
Public Class CarOutfitSelector
    Inherits System.Windows.Forms.UserControl

    #Region " Vom Windows Form Designer generierter Code "
    #End Region

    <Category("Ausstattung"), Description("Klimaanlage")> _
        Public Property AirConditioner() As Boolean
        Get
            Return CHKAirConditioner.Checked
        End Get
        Set(ByVal Value As Boolean)
            CHKAirConditioner.Checked = Value
        End Set
    End Property

    <Category("Ausstattung"), Description("Schiebedach")> _
        Public Property SlidingRoof() As Boolean
        ...
    End Property

    <Category("Ausstattung"), Description("Nebelscheinwerfer")> _
        Public Property FogLights() As Boolean
        ...
    End Property

    <Category("Ausstattung"), Description( _
        "Radio mit Kassettenrecorder")> _
        Public Property CassetteRadio() As Boolean
        ...
    End Property

    <Category("Ausstattung"), Description("Radio mit CD-Player")> _
        Public Property CDRadio() As Boolean
        ...
    End Property

End Class
```

---

*Listing 290: Das Steuerelement CarOutfitSelector*

Um die Aktionstasten im Eigenschaftsfenster anzuzeigen, muss der Klasse CarOutfitSelector ein Designer zugewiesen werden, der die Aktionen bekannt macht:

---

```
<Designer(GetType(CarOutfitSelector.CarOutfitDesigner))> _
Public Class CarOutfitSelector
    Inherits System.Windows.Forms.UserControl
```

---



---

```
...
End Class
```

---

Diese Designer-Klasse (CarOutfitDesigner) wiederum wird als innere Klasse von CarOutfitSelector implementiert (siehe Listing 291). Sie ist direkt abgeleitet von ControlDesigner und überschreibt lediglich die Get-Methode der Eigenschaft Verbs. Verbs gibt die Auflistung aller verfügbaren Aktionen in Form einer DesignerVerbCollection-Liste zurück und legt diese Liste beim ersten Aufruf an. Ein Element dieser Liste basiert auf dem Typ DesignerVerb und beinhaltet als Informationen den anzuzeigenden Text sowie eine Delegate-Referenz für die Methode, die aufgerufen werden soll, wenn der Benutzer auf die Schaltfläche klickt.

---

```
Public Class CarOutfitDesigner
    Inherits ControlDesigner

    ' Die Liste der Verben stellt die Befehlstasten im PropertyGrid
    ' bereit. Hier eine Referenz-Variable für Singleton-Pattern
    Private coVerbs As DesignerVerbCollection

    ' Einstellungen für Standard-Ausstattung
    Private Sub OnStandard(ByVal sender As Object, _
        ByVal e As EventArgs)

        Dim co As CarOutfitSelector
        co = DirectCast(Control, CarOutfitSelector)
        ChangeCheckBoxValue(co.CHKAirConditioner, False)
        ChangeCheckBoxValue(co.CHKCassetteRadio, False)
        ChangeCheckBoxValue(co.CHKCDRadio, False)
        ChangeCheckBoxValue(co.CHKFogLights, False)
        ChangeCheckBoxValue(co.CHKSlidingRoof, False)
    End Sub

    ' Einstellungen für Luxus-Ausstattung
    Private Sub OnLuxury(ByVal sender As Object, _
        ByVal e As EventArgs)

        Dim co As CarOutfitSelector
        co = DirectCast(Control, CarOutfitSelector)
        ChangeCheckBoxValue(co.CHKAirConditioner, True)
        ChangeCheckBoxValue(co.CHKCassetteRadio, False)
        ChangeCheckBoxValue(co.CHKCDRadio, True)
        ChangeCheckBoxValue(co.CHKFogLights, True)
        ChangeCheckBoxValue(co.CHKSlidingRoof, True)
    End Sub

    ' Einstellungen für Sonderangebot
    Private Sub OnSpecialOffer(ByVal sender As Object, _
        ByVal e As EventArgs)

        Dim co As CarOutfitSelector
```

---

Listing 291: Control-Designer zur Bereitstellung der Befehlsschaltflächen Standard, Luxus und Special Offer

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Prop-  
ertyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

---

```

co = DirectCast(Control, CarOutfitSelector)
ChangeCheckBoxValue(co.CHKAirConditioner, False)
ChangeCheckBoxValue(co.CHKCassetteRadio, True)
ChangeCheckBoxValue(co.CHKCDRadio, False)
ChangeCheckBoxValue(co.CHKFogLights, True)
ChangeCheckBoxValue(co.CHKSlidingRoof, True)
End Sub

Private Sub ChangeCheckBoxValue(ByVal cb As CheckBox, _
    ByVal value As Boolean)

    ' ComponentChangeService über Änderung informieren
    RaiseComponentChanged(TypeDescriptor.GetProperties( _
        Control).Find(cb.Name, True), cb.Checked, value)

    ' CheckBox setzen
    cb.Checked = value

End Sub

Public Overrides ReadOnly Property Verbs() As _
    System.ComponentModel.Design.DesignerVerbCollection

Get

    ' Wenn die Liste noch nicht existiert
    If coVerbs Is Nothing Then

        ' Neue Liste erstellen
        coVerbs = New DesignerVerbCollection

        ' Befehlstaste für Standard-Ausstattung
        coVerbs.Add(New DesignerVerb("Standard", _
            New EventHandler(AddressOf OnStandard)))

        ' Befehlstaste für Luxus-Ausstattung
        coVerbs.Add(New DesignerVerb("Luxus", _
            New EventHandler(AddressOf OnLuxury)))

        ' Befehlstaste für Sonderangebots-Ausstattung
        coVerbs.Add(New DesignerVerb("Special Offer", _
            New EventHandler(AddressOf OnSpecialOffer)))

    End If

    ' Liste zurückgeben
    Return coVerbs

End Get
End Property
End Class

```

---

Listing 291: Control-Designer zur Bereitstellung der Befehlsschaltflächen Standard, Luxus und Special Offer (Forts.)

Ausgeführt werden bei Klick auf die entsprechenden Schaltflächen die Methoden `OnStandard`, die alle `CheckBox`en löscht, `OnLuxus`, die die `CheckBox`en für die Luxus-Ausstattung setzt, sowie `OnSpecialOffer`, die die Kombination für das Sonderangebot auswählt. Der Zugriff auf die `CheckBox`en erfolgt in der Methode `ChangeCheckBoxValue`. Hier wird zusätzlich `RaiseComponentChanged` aufgerufen, um alle Designer und somit auch das Eigenschaftsfenster über die Änderung der Eigenschaft zu informieren. Nur so wird sichergestellt, dass nicht nur die `CheckBox`en auf dem Steuerelement richtig gesetzt werden, sondern auch die Anzeige im Eigenschaftsfenster aktualisiert wird.

Innerhalb der `OnXXX`-Methoden können Sie beliebige Aktionen programmieren. Sie können beispielsweise eigene Dialoge anzeigen, die dem Anwender helfen, komplexe Einstellungen vorzunehmen. Gängige Vorgehensweisen, die über die Änderung von Eigenschaften im `PropertyGrid` zu umständlich sind, können über derartige Aktionstasten gesteuert werden. Aktionstasten können dazu beitragen, die Konfiguration des Steuerelementes erheblich zu beschleunigen.

## 162 Eigenschaften dynamisch erstellen und hinzufügen

In den vorangegangenen Beispielen wurde es dem Eigenschaftsfenster überlassen, via Reflection die Eigenschaften der anzuzeigenden Objekte samt ihrer Attribute zu ermitteln. Alle Angaben zu Name, Beschreibung, Sichtbarkeit etc. wurden statisch über Attribute zur Compile-Zeit festgelegt. Oft stehen aber nicht alle Informationen schon vor Programmstart zur Verfügung. Dann muss dem `PropertyGrid` zur Laufzeit mitgeteilt werden, über welche Eigenschaften das Objekt verfügt und wie diese aufgebaut sind. Für den Anwender ist nicht erkennbar, ob die Eigenschaften dynamisch zur Laufzeit definiert oder über Reflection ermittelt worden sind (Abbildung 157).



Abbildung 157: Anzeige zur Laufzeit definierter Eigenschaften

Als Beispiel dient eine kleine `ToDoList`-Klasse, die im Wesentlichen aus einem `String`-Array besteht (Listing 292):

```
Public Class ToDoList
    ' Liste der Tätigkeiten
```

Listing 292: Beispielklasse mit Aufgabenliste als `String`-Array

---

```

Public ToDoItems() As String

' Konstruktor, der eine Liste übernimmt
Public Sub New(ByVal items() As String)
    ToDoItems = items
End Sub

End Class

```

---

*Listing 292: Beispielklasse mit Aufgabenliste als String-Array (Forts.)*

Die Klasse besitzt keine öffentlichen Properties, die direkt im Eigenschaftsfenster angezeigt werden könnten. Um dem PropertyGrid-Control dennoch die Möglichkeit zur Ermittlung anzuzeigender Eigenschaften zu geben, muss die Klasse das Interface `ICustomTypeDescriptor` implementieren (Listing 293). Hierzu müssen eine Reihe von Methoden implementiert werden, von denen die meisten Standardwerte zurückgeben oder die Aufgaben an andere Methoden delegieren.

---

```

Imports System.ComponentModel

Public Class ToDoList
    Implements ICustomTypeDescriptor

' Liste der Tätigkeiten
Public ToDoItems() As String

' Konstruktor, der eine Liste übernimmt
Public Sub New(ByVal items() As String)
    ToDoItems = items
End Sub

' Notwendige zu implementierende Funktionen,
' die Standardwerte zurückgeben
Public Function GetAttributes() As _
    System.ComponentModel.AttributeCollection _
    Implements _
    System.ComponentModel.ICustomTypeDescriptor.GetAttributes
    Return TypeDescriptor.GetAttributes(Me, True)
End Function

Public Function GetClassName() As String Implements _
    System.ComponentModel.ICustomTypeDescriptor.GetClassName

    Return TypeDescriptor.GetClassName(Me, True)
End Function

Public Function GetComponentName() As String Implements _
    System.ComponentModel.ICustomTypeDescriptor.GetComponentName

    Return TypeDescriptor.GetComponentName(Me, True)

```

---

*Listing 293: Klasse `ToDoList` implementiert das Interface `ICustomTypeDescriptor`*

---

End Function

```
Public Function GetConverter() As _
    System.ComponentModel.TypeConverter _
    Implements _
        System.ComponentModel.ICustomTypeDescriptor.GetConverter
```

```
Return TypeDescriptor.GetConverter(Me, True)
```

End Function

```
Public Function GetDefaultEvent() As _
    System.ComponentModel.EventDescriptor _
    Implements _
        System.ComponentModel.ICustomTypeDescriptor.GetDefaultEvent
```

```
Return TypeDescriptor.GetDefaultEvent(Me, True)
```

End Function

```
Public Function GetDefaultProperty() As _
    System.ComponentModel.PropertyDescriptor Implements _
        System.ComponentModel.ICustomTypeDescriptor.GetDefaultProperty
```

```
Return TypeDescriptor.GetDefaultProperty(Me, True)
```

End Function

```
Public Function GetEditor(ByVal editorBaseType As System.Type) _
    As Object Implements _
        System.ComponentModel.ICustomTypeDescriptor.GetEditor
```

```
Return TypeDescriptor.GetEditor(Me, editorBaseType, True)
```

End Function

```
Public Overloads Function GetEvents() As _
    System.ComponentModel.EventDescriptorCollection Implements _
        System.ComponentModel.ICustomTypeDescriptor.GetEvents
```

```
Return TypeDescriptor.GetEvents(Me, True)
```

End Function

```
Public Overloads Function GetEvents1(ByVal attributes() As _
    System.Attribute) As _
    System.ComponentModel.EventDescriptorCollection Implements _
        System.ComponentModel.ICustomTypeDescriptor.GetEvents
```

```
Return TypeDescriptor.GetEvents(Me, attributes, True)
```

End Function

```
Public Overloads Function GetProperties() As _
    System.ComponentModel.PropertyDescriptorCollection Implements _
        System.ComponentModel.ICustomTypeDescriptor.GetProperties
```

---

Basics

---

Datum/  
Zeit

---

Anwen-  
dungen

---

Zeich-  
nen

---

Bild-  
bearbei-  
tung

---

Win-  
dows  
Forms

---

Controls

---

Prop-  
ertyGrid

---

Datei-  
system

---

Netz-  
werk

---

System

---

Daten-  
banken

---

XML

---

Wissen-  
schaft

---

Ver-  
schiede-  
nes

---

```

    GetProperties(Nothing)
End Function

Public Function GetPropertyOwner(ByVal pd As _
    System.ComponentModel.PropertyDescriptor) As Object Implements _
    System.ComponentModel.ICustomTypeDescriptor.GetPropertyOwner

    Return Me
End Function

' Definition der Eigenschaften
Public Overloads Function GetProperties(ByVal attributes() As _
    System.Attribute) As _
    System.ComponentModel.PropertyDescriptorCollection Implements _
    System.ComponentModel.ICustomTypeDescriptor.GetProperties

    ' Neue Beschreibungsliste anlegen
    Dim pdc As New PropertyDescriptorCollection(Nothing)

    ' Descriptor für Anzahl der Einträge hinzufügen
    pdc.Add(New TDLCountPropertyDescriptor)

    ' Für jeden Array-Eintrag eine Eigenschaft hinzufügen
    For i As Integer = 0 To ToDoItems.GetUpperBound(0)
        pdc.Add(New TDLEntryPropertyDescriptor(i))
    Next

    ' Rückgabe der Liste
    Return pdc

End Function

End Class

```

---

*Listing 293: Klasse ToDoList implementiert das Interface ICustomTypeDescriptor (Forts.)*

Lediglich die Methode `GetProperties` muss implementiert werden. Sie erzeugt eine Liste vom Typ `PropertyDescriptorCollection`, fügt Einträge vom Typ `PropertyDescriptor` hinzu und gibt die Liste als Funktionswert zurück. In dieser Methode können Sie selbst die Eigenschaften, die das `PropertyGrid` darstellen soll, zusammenstellen. Sie sind in keinsten Weise an die `Properties` gebunden, die die Klasse zur Verfügung stellt, sondern können nach eigenem Ermessen neue Eigenschaften definieren.

Da `PropertyDescriptor` eine abstrakte (MustInherit) Klasse ist, muss für jeden Eigenschaftstyp, der verwendet werden soll, eine abgeleitete Klasse angelegt werden. Listing 294 zeigt die Ableitung zur Anzeige der Texteinträge (Klasse `TDLEntryPropertyDescriptor`), Listing 295 die Ableitung zur Anzeige der Anzahl der Einträge (Klasse `TDLCountPropertyDescriptor`).

```

' Beschreibung einer ToDoItem-Eigenschaft
Class TDLEntryPropertyDescriptor
Inherits PropertyDescriptor

' Index beim Anlegen merken, damit später auf das Element
' zugegriffen werden kann
Private index As Integer

' Konstruktor übernimmt zusätzlich noch den Index
Public Sub New(ByVal index As Integer)

    MyBase.New("Aufgabe " & index, New Attribute() _
        {CategoryAttribute.Data, _
        New DescriptionAttribute( _
            "Zeigt eine Aufgabe an oder legt sie fest"), _
        New CategoryAttribute("Aufgaben")})

    Me.index = index
End Sub

' Kein Reset möglich
Public Overrides Function CanResetValue( _
    ByVal component As Object) As Boolean

    Return False
End Function

' Typ der Komponente ist ToDoList
Public Overrides ReadOnly Property ComponentType() As System.Type
    Get
        Return GetType(ToDoList)
    End Get
End Property

' Wert über gespeicherten Index ermitteln
Public Overrides Function GetValue(ByVal component As Object) _
    As Object

    Return DirectCast(component, ToDoList).ToDoItems(index)
End Function

' Eigenschaft kann auch geändert werden
Public Overrides ReadOnly Property IsReadOnly() As Boolean
    Get
        Return False
    End Get
End Property

' Typ ist String
Public Overrides ReadOnly Property PropertyType() As System.Type

```

Listing 294: Klasse `TDLEntryPropertyDescriptor` definiert den Aufbau der Eigenschaften, die die Texteinträge repräsentieren sollen

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Proper-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

---

```

    Get
        Return GetType(String)
    End Get
End Property

' Wird hier nicht gebraucht, muss aber überschrieben werden
Public Overrides Sub ResetValue(ByVal component As Object)
End Sub

' Setzen des Eintrags
Public Overrides Sub SetValue(ByVal component As Object, _
    ByVal value As Object)

    ' Zugriff über gespeicherten Index
    DirectCast(component, ToDoList).ToDoItems(index) = CStr(value)
End Sub

' Wird hier nicht gebraucht, muss aber überschrieben werden
Public Overrides Function ShouldSerializeValue( _
    ByVal component As Object) As Boolean

    Return False
End Function

End Class

```

---

*Listing 294: Klasse `TDLEntryPropertyDescriptor` definiert den Aufbau der Eigenschaften, die die Texteinträge repräsentieren sollen (Forts.)*

Für jeden Eintrag wird eine neue Instanz der Klasse `TDLEntryPropertyDescriptor` erzeugt. Der Konstruktor übernimmt den Index, der die Position des Elementes im Array angibt, und speichert ihn für den späteren Zugriff. Dem Konstruktor der Basisklasse werden der zusammengesetzte Name sowie verschiedene Attribute übergeben. Es handelt sich dabei um die Attribute wie `DescriptionAttribute`, `CategoryAttribute` oder `BrowsableAttribute`, die in den vorherigen Beispielen zur Compile-Zeit direkt der Daten-Klasse zugeordnet worden sind. Hier werden zur Laufzeit dynamisch Instanzen von diesen Attributklassen erzeugt und alle gemeinsam als Array dem Basis-klassen-Konstruktor übergeben.

Durch die Ableitung von `PropertyDescriptor` wird die Überschreibung einiger Methoden und Properties erzwungen. Zwei Property-Get-Methoden benutzt das `PropertyGrid-Control` für die Ermittlung der Datentypen:

1. `ComponentType`
2. `PropertyType`

Erstere muss den Typ der Komponente, also z.B. des Steuerelementes zurückgeben (hier `ToDoList`), die zweite den Typ der Eigenschaft (hier `String` für die Texteinträge). `GetValue` wird aufgerufen, um den Wert der Eigenschaft zu ermitteln. Hier wird der im Konstruktor gespeicherte Index benutzt, um den betreffenden Wert aus dem Array zu lesen. Analog dazu wird in der Methode `SetValue` ein neuer Wert in das Array eingetragen und somit der alte überschrieben.

`TDLCountPropertyDescriptor` ist genauso aufgebaut. Die Klasse beschreibt die Eigenschaft `Anzahl` Einträge, die die Anzahl der Array-Einträge anzeigen soll. Sie ist nicht schreibgeschützt und kann



vom Anwender geändert werden. Bei einer Änderung wird ein neues Array der benötigten Größe angelegt und die bisherigen Daten kopiert. Dieser Vorgang ist in der Methode `SetValue` (Listing 295) implementiert. Das zusätzlich im Konstruktor übergebene Attribut `RefreshPropertiesAttribute.All` führt dazu, dass die Eigenschaftsliste automatisch neu aufgebaut wird, wenn die Anzahl der Listenelemente geändert worden ist (siehe Abbildung 158).

```
' Property-Descriptor für "Anzahl Einträge"
Class TDLCountPropertyDescriptor
    Inherits PropertyDescriptor

    ' Konstruktor muss Konstruktor der Basisklasse aufrufen
    ' und die notwendigen Parameter übergeben
    Public Sub New()
        MyBase.New("Anzahl Einträge", New Attribute() _
            {CategoryAttribute.Data, RefreshPropertiesAttribute.All, _
            New DescriptionAttribute("Zeigt die Anzahl der " & _
            "angelegten Einträge an oder legt sie fest"), _
            New CategoryAttribute("Verwaltung")})

    End Sub

    ' Kein Reset möglich
    Public Overrides Function CanResetValue( _
        ByVal component As Object) As Boolean

        Return False
    End Function

    ' Typ der Komponente ist ToDoList
    Public Overrides ReadOnly Property ComponentType() As System.Type
        Get
            Return GetType(ToDoList)
        End Get
    End Property

    ' Anzahl der Array-Elemente lesen
    Public Overrides Function GetValue(ByVal component As Object) _
        As Object

        Return DirectCast(component, ToDoList).ToDoItems.Length
    End Function

    ' Eigenschaft kann auch geändert werden
    Public Overrides ReadOnly Property IsReadOnly() As Boolean
        Get
            Return False
        End Get
    End Property

    ' Typ ist Integer
```

Listing 295: Klasse `TDLCountPropertyDescriptor` definiert den Aufbau der Eigenschaft »Anzahl Einträge«

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Proper-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

---

```

Public Overrides ReadOnly Property PropertyType() As System.Type
    Get
        Return GetType(Integer)
    End Get
End Property

' Wird hier nicht gebraucht, muss aber überschrieben werden
Public Overrides Sub ResetValue(ByVal component As Object)
End Sub

' Setzen des Wertes ändert die Array-Größe
' Vorhandene Elemente werden kopiert
Public Overrides Sub SetValue(ByVal component As Object, _
    ByVal value As Object)

    ' Neue Anzahl
    Dim n As Integer = CType(value, Integer)

    ' Die ToDo-Liste
    Dim tdl As ToDoList = DirectCast(component, ToDoList)

    ' Sicherheitscheck
    If n < 0 Then Throw New ArgumentOutOfRangeException( _
        "Werte kleiner Null sind nicht zulässig")

    ' Neues Array anlegen
    Dim newList(n - 1) As String

    ' Werte kopieren
    Array.Copy(tdl.ToDoItems, newList, _
        Math.Min(tdl.ToDoItems.Length, n))

    ' Ab jetzt neues Array verwenden
    tdl.ToDoItems = newList

End Sub

' Wird hier nicht gebraucht, muss aber überschrieben werden
Public Overrides Function ShouldSerializeValue( _
    ByVal component As Object) As Boolean
    Return False
End Function

End Class

```

---

*Listing 295: Klasse TDLCountPropertyDescriptor definiert den Aufbau der Eigenschaft »Anzahl Einträge« (Forts.)*



Abbildung 158: Bei Änderung der Eigenschaft »Anzahl Einträge« werden neue Eigenschaften in der Aufgabenliste hinzugefügt

### 163 Eigenschaften in unterschiedlichen Sprachen anzeigen (Lokalisierung)

Im vorangegangenen Rezept haben Sie gesehen, wie die Eigenschaften dynamisch zur Laufzeit generiert werden können. Alle Bezeichnungen, die im PropertyGrid dargestellt werden, werden dynamisch erstellt. Diese Technik können Sie auch dazu einsetzen, die Anzeigen im Eigenschaftsfenster an die im Betriebssystem eingestellte Sprache anzupassen.

Hierfür definieren Sie für die Texte Ressource-Dateien in den gewünschten Sprachen und laden diese über den ResourceManager zur Laufzeit. Tabelle 26 zeigt die für das Beispiel benötigten Bezeichner, Abbildung 159 und Abbildung 160 die Daten-Ansichten der Ressource-Dateien in Visual Studio. Im Beispielprogramm lassen sich die Sprachen zur Laufzeit umschalten (siehe Abbildung 161 und Abbildung 162).

| Bezeichner            | Bedeutung                                                 |
|-----------------------|-----------------------------------------------------------|
| EntryCountName        | Bezeichnung des Eintrags für die Anzahl der Einträge      |
| EntryCountDescription | Beschreibung zu EntryCountName                            |
| EntryName             | Bezeichnung für einen einzelnen Eintrag der Aufgabenliste |
| EntryNameDescription  | Beschreibung zu EntryName                                 |
| Head1                 | Kategorienbezeichnung für Aufgabenliste                   |
| Head2                 | Kategorienbezeichnung für EntryCount                      |

Tabelle 26: Verwendung der Bezeichner für die Text-Ressourcen

Da Deutsch die Voreinstellung für die Projekteinstellung des Beispielprojektes ist, werden die Einträge für die deutschen Texte in der Default-Ressource-Datei, hier *PropertyStringtable.resx* angelegt, während die englischen Texte in der allgemeinen englischen Ressource-Datei, hier *PropertyStringtable.en.resx*, platziert werden. Zugriffen wird auf die Ressourcen über eine Instanz der Klasse *ResourceManager*, deren Referenz in der öffentlichen Member-Variablen *ResMan* der Klasse *ToDoList* bereitgestellt wird:

|                           |
|---------------------------|
| Basics                    |
| Datum/<br>Zeit            |
| Anwen-<br>dungen          |
| Zeich-<br>nen             |
| Bild-<br>bearbei-<br>tung |
| Win-<br>dows<br>Forms     |
| Controls                  |
| Prop-<br>ertyGrid         |
| Datei-<br>system          |
| Netz-<br>werk             |
| System                    |
| Daten-<br>banken          |
| XML                       |
| Wissen-<br>schaft         |
| Ver-<br>schiede-<br>nes   |

```
Public Shared ResMan As New System.Resources.ResourceManager _
    ("PG14.PropertyStringtable", _
    System.Reflection.Assembly.GetAssembly(GetType(ToDoList)))
```

Datentabell Daten:

| data      | Daten für data        |                                                                |         |        |          |
|-----------|-----------------------|----------------------------------------------------------------|---------|--------|----------|
| resheader | name                  | value                                                          | comment | type   | mimetype |
| ▶         | EntryCountName        | Anzahl Einträge                                                | (Null)  | (Null) | (Null)   |
|           | EntryCountDescription | Zeigt die Anzahl der angelegten Aufgaben an oder legt sie fest | (Null)  | (Null) | (Null)   |
|           | EntryName             | Aufgabe                                                        | (Null)  | (Null) | (Null)   |
|           | EntryNameDescription  | Zeigt eine Aufgabe an oder legt sie fest                       | (Null)  | (Null) | (Null)   |
|           | Head1                 | Aufgaben                                                       | (Null)  | (Null) | (Null)   |
|           | Head2                 | Verwaltung                                                     | (Null)  | (Null) | (Null)   |
| *         |                       |                                                                |         |        |          |

☐ XML ☒ Daten

Abbildung 159: Deutsche Texte für die Darstellung im Eigenschaftsfenster

Datentabellen: Daten:

| data      | Daten für data        |                                   |         |        |          |
|-----------|-----------------------|-----------------------------------|---------|--------|----------|
| resheader | name                  | value                             | comment | type   | mimetype |
| ▶         | EntryCountName        | Number of Entries                 | (Null)  | (Null) | (Null)   |
|           | EntryCountDescription | Shows or sets the Number of Tasks | (Null)  | (Null) | (Null)   |
|           | EntryName             | Task                              | (Null)  | (Null) | (Null)   |
|           | EntryNameDescription  | Shows or sets the Task to do      | (Null)  | (Null) | (Null)   |
|           | Head1                 | Tasks                             | (Null)  | (Null) | (Null)   |
|           | Head2                 | Administration                    | (Null)  | (Null) | (Null)   |
| *         |                       |                                   |         |        |          |

☐ XML ☒ Daten

Abbildung 160: Englische Texte für die Darstellung im Eigenschaftsfenster

**Eigene PropertyGrid - Tabs**

☐ Englisch
 ☒ Deutsch

☐ Aufgaben
 

|          |                  |
|----------|------------------|
| Aufgabe0 | Kapitel 5 lesen  |
| Aufgabe1 | Elke gratulieren |
| Aufgabe2 | Rasen mähen      |

☐ Verwaltung
 

|                 |   |
|-----------------|---|
| Anzahl Einträge | 3 |
|-----------------|---|

**Anzahl Einträge**  
 Zeigt die Anzahl der angelegten Aufgaben an oder legt sie fest

Abbildung 161: Aufgabenliste mit Sprachauswahl, hier Deutsch

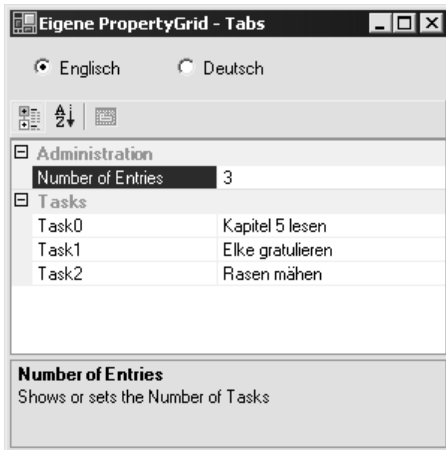


Abbildung 162: Aufgabenliste mit Sprachauswahl, hier Englisch

Ansonsten ist der Aufbau der Klasse `ToDoList` identisch mit Listing 293. In der Klasse `TDLCountPropertyDescriptor` (vgl. Listing 295) werden im Konstruktor die Bezeichnungen nicht statisch angelegt, sondern mit Hilfe des `ResourceManagers` aus der jeweils gültigen Ressource-Datei gelesen (siehe Listing 296).

```
Public Sub New()
    MyBase.New( _
        ToDoList.ResMan.GetString("EntryCountName"), New Attribute() _
        {CategoryAttribute.Data, RefreshPropertiesAttribute.All, _
        New DescriptionAttribute( _
            ToDoList.ResMan.GetString("EntryCountDescription")), _
        New CategoryAttribute(ToDoList.ResMan.GetString("Head2"))})

End Sub
```

Listing 296: Ermitteln der Attributtexte in der Klasse `TDLCountPropertyDescriptor`

Analog dazu erfolgt die Ermittlung der anzuzeigenden Texte im Konstruktor der Klasse `TDLEntryPropertyDescriptor` (siehe Listing 297).

```
Public Sub New(ByVal index As Integer)

    MyBase.New(ToDoList.ResMan.GetString("EntryName") & index, _
        New Attribute() _
        {CategoryAttribute.Data, _
        New DescriptionAttribute( _
            ToDoList.ResMan.GetString("EntryDescription")), _
        New CategoryAttribute(ToDoList.ResMan.GetString("Head1"))})

    Me.index = index
End Sub
```

Listing 297: Konstruktor der Klasse `TDLEntryPropertyDescriptor`

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Prop-  
ertyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

Die Methode `GetString` ermittelt den zum angegebenen Schlüssel gehörigen Text aus der Resource-Datei, die der aktuellen Ländereinstellung (`UICulture`) zugeordnet ist. Um die Texte zur Laufzeit austauschen zu können, muss daher lediglich die Ländereinstellung geändert und das Eigenschaftsfenster aufgefrischt werden. In Listing 298 sehen Sie die entsprechenden Aufrufe, die in den Click-Ereignissen der beiden `RadioButtons` ausgeführt werden.

---

```
Private Sub RBEnglish_CheckedChanged(ByVal sender As _
    System.Object, ByVal e As System.EventArgs) _
    Handles RBEnglish.CheckedChanged

    Thread.CurrentThread.CurrentUICulture = _
        New CultureInfo("en-US")

    PGTest.Refresh()

End Sub

Private Sub RBGerman_CheckedChanged(ByVal sender As _
    System.Object, ByVal e As System.EventArgs) _
    Handles RBGerman.CheckedChanged

    Thread.CurrentThread.CurrentUICulture = _
        New CultureInfo("de")

    PGTest.Refresh()

End Sub
```

---

*Listing 298: Umschalten der Ländereinstellung und Auffrischen der Anzeigentexte im Eigenschaftsfenster*

## 164 Neue Tab-Flächen hinzufügen

Vielleicht ist Ihnen schon einmal aufgefallen, dass die Entwicklungsumgebung (Visual Studio) bei C#-Programmen anders mit den Ereignissen der Steuerelemente umgeht. Alle Ereignisse werden ebenfalls im Eigenschaftsfenster angezeigt. Um eine optische und logische Trennung herzustellen, wird das Eigenschaftsfenster um eine Kartenreiter-Schaltfläche (Tab) erweitert. Sie zeigt einen kleinen Blitz und öffnet ein neues Tab-Fenster, auf dem nur die Ereignisse zu sehen sind (Abbildung 163).

Diese Technik steht Ihnen selbstverständlich auch zur Verfügung, wenn Sie beispielsweise eine Reihe von Eigenschaften von den Basiseigenschaften eines Steuerelementes getrennt anzeigen möchten, um dem Anwender mehr Übersicht zu bieten. Abbildung 164 zeigt die Eigenschaften der `ToDo`-Liste auf einer eigenen Tab-Seite.

Über das Attribut `PropertyTabAttribut`, das der Klasse `ToDoList` zugeordnet wird (Listing 299), erfährt das Eigenschaftsfenster, dass eine zusätzliche Tab-Seite anzuzeigen ist. Dem Attribut wird der Typ einer von `PropertyTab` abgeleiteten Klasse (`ToDoPropertyTab`) mitgegeben, die zur Anzeige der neuen Seite instanziiert wird.

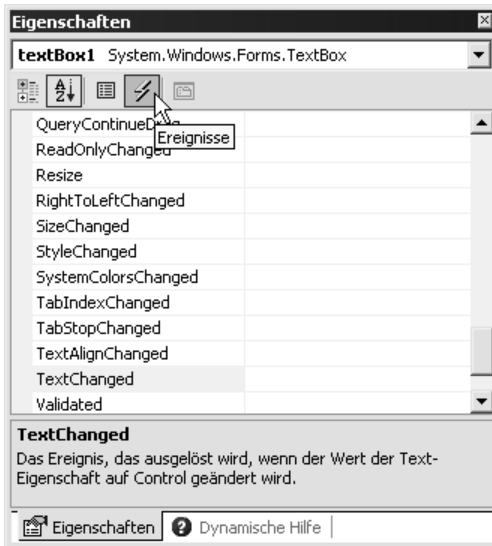


Abbildung 163: Die Entwicklungsumgebung in C# zeigt die Ereignisse eines Steuerelementes auf einer eigenen Tab-Seite an

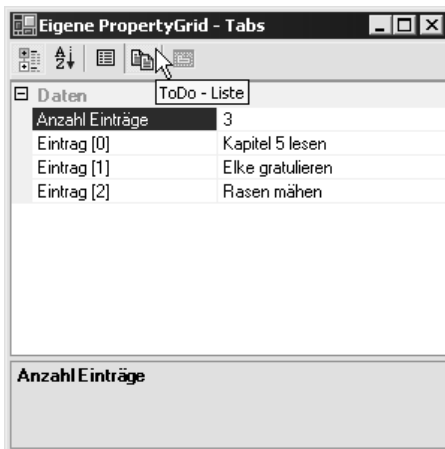


Abbildung 164: Anzeige der Eigenschaften in einem neuen Tab-Fenster

```
<PropertyTab(GetType(ToDoPropertyTab), _
PropertyTabScope.Component)> _
Public Class ToDoList

    ' Liste der Tätigkeiten
    Public ToDoItems() As String

    ' Konstruktor, der eine Liste übernimmt
    Public Sub New(ByVal items() As String)
```

Listing 299: Zuordnung einer zusätzlichen Tab-Seite über das Attribut PropertyTab

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Prop-  
ertyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

```

    ToDoItems = items
End Sub

```

```

End Class

```

---

*Listing 299: Zuordnung einer zusätzlichen Tab-Seite über das Attribut PropertyTab (Forts.)*

Durch die Ableitung von der abstrakten Klasse `PropertyTab` werden wiederum einige Methoden- und Property-Überschreibungen erzwungen (siehe Listing 300). So wird beispielsweise `TabName` überschrieben, um dem Tab einen Namen zuzuordnen, der als `ToolTip` angezeigt werden kann. Auch wenn es syntaktisch nicht unbedingt notwendig wäre, muss dennoch unbedingt die `Get`-Methode der Eigenschaft `Bitmap` überschrieben werden. Nur dann zeigt das Eigenschaftsfenster auch tatsächlich die Schaltfläche an. Ohne Angabe eines gültigen `Bitmaps` bleibt der gesamte Code zur Anzeige der Tab-Seite wirkungslos.

---

```

Public Class ToDoPropertyTab
    Inherits PropertyTab

    ' Überschreibung erforderlich
    ' Ruft die Methode mit drei Parametern auf
    Public Overloads Overrides Function GetProperties( _
        ByVal component As Object, _
        ByVal attributes() As System.Attribute) _
        As System.ComponentModel.PropertyDescriptorCollection

        Return GetProperties(Nothing, component, attributes)

    End Function

    ' Text, der als ToolTip angezeigt werden soll
    Public Overrides ReadOnly Property TabName() As String
        Get
            Return "ToDo-Liste"
        End Get
    End Property

    ' Anzuzeigendes Bitmap
    ' Diese Methode muss!!! überschrieben werden, sonst wird gar
    ' nichts angezeigt
    Public Overrides ReadOnly Property Bitmap() _
        As System.Drawing.Bitmap

        Get
            Return New Bitmap("../copy.bmp")
        End Get

    End Property

    ' Definition der anzuzeigenden Eigenschaften
    Public Overloads Overrides Function GetProperties( _

```

---

*Listing 300: Klasse `ToDoPropertyTab` definiert den Aufbau der zusätzlichen Seite im `PropertyGrid`*



```

ByVal context As System.ComponentModel.ITypeDescriptorContext, _
ByVal component As Object, _
ByVal attributes() As System.Attribute) _
As System.ComponentModel.PropertyDescriptorCollection

' Standardrückgabe, wenn es nicht um Objekte vom Typ ToDoList
' geht
If Not TypeOf component Is ToDoList Then
    Dim tc As TypeConverter = _
        TypeDescriptor.GetConverter(component)

    If tc Is Nothing Then
        Return TypeDescriptor.GetProperties(component, attributes)
    Else
        Return tc.GetProperties(context, attributes)
    End If
End If

' Neue Beschreibungsliste anlegen
Dim pdc As New PropertyDescriptorCollection(Nothing)

' Descriptor für Anzahl der Einträge hinzufügen
pdc.Add(New TDLCountPropertyDescriptor(Me))

' Für jeden Array-Eintrag eine Eigenschaft hinzufügen
Dim tdl As ToDoList = DirectCast(component, ToDoList)
For i As Integer = 0 To tdl.ToDoItems.GetUpperBound(0)
    pdc.Add(New TDLEntryPropertyDescriptor(Me, i))
Next

' Rückgabe der Liste
Return pdc

End Function

End Class

```

Listing 300: Klasse *ToDoPropertyTab* definiert den Aufbau der zusätzlichen Seite im *PropertyGrid*

Kern der Klasse ist jedoch die Überschreibung von *GetProperties*. Hier wird wie in den beiden vorangegangenen Beispielen eine Liste vom Typ *PropertyDescriptorCollection* erzeugt. Zusätzlich muss aber eine Sicherheitsabfrage vorgesehen werden, damit die benötigten Eigenschaften nicht versehentlich für artfremde Objekte erzeugt werden.

Die beiden Klassen, deren Instanzen der zurückzugebenden Eigenschaftsliste hinzugefügt werden, wurden bereits beschrieben und sollen hier nicht noch einmal erläutert werden (*TDLCountPropertyDescriptor* siehe Listing 295, *TDLEntryPropertyDescriptor* siehe Listing 294).

Für die Darstellung der Eigenschaften auf der neuen Tab-Seite gelten die gleichen Regeln und Vorgehensweisen wie für die Darstellung auf der Hauptseite. Auch hier können Sie wieder Attribute zuordnen, Eigenschaften gruppieren und spezielle Editoren einsetzen.

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Proper-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes



# Dateisystem

Das Dateisystem ist eine der tragenden Säulen des Betriebssystems. Verzeichnisse und Dateien sind die Grundlage jeder Datenorganisation. Auch bei der Programmierung spielen sie eine wichtige Rolle.

Wir zeigen Ihnen typische, oft benötigte Vorgehensweisen für den Umgang mit Dateien und Verzeichnissen. Sie finden hier sowohl Rezepte, die über das Framework realisierbar sind, als auch solche, die Zugriffe über verschiedene Betriebssystemfunktionen benötigen.

## 165 System-Verzeichnisse mit .NET

Eines der vordringlichen Ziele eines Betriebssystems ist die Verwaltung von Dateien, damit der Benutzer eines Computersystems dies möglichst einfach bewerkstelligen kann. Wer die Anfänge zum Beispiel mit den Desktop-Betriebssystemen CP/M und MS DOS 2.0 erlebt hat, weiß die Vorzüge der modernen Betriebssysteme zu schätzen. Um ein wenig Ordnung in die Menge der Dateien zu bringen, gibt es die Möglichkeit, Verzeichnisse/Ordner zu erstellen. Einigen dieser Verzeichnisse kommt eine fest definierte Bedeutung zu. Bei diesen Ordnern handelt es sich um vorgegebene Verzeichnisse des Betriebssystems.

Einige der Verzeichnisse sind allerdings auch abhängig von dem Benutzer, der sich gerade am System angemeldet hat. Um einem Programmierer nun die Möglichkeit an die Hand zu geben, diese Verzeichnisse in seinem Programm ermitteln zu können, gibt es im Namensraum `System.Environment` Unterstützung.

Leider gibt es keine Methode in diesem Namensraum, um das Windows-Verzeichnis zu ermitteln. Dieses hat zwar je nach Windows-Betriebssystem-Version einen Vorgabennamen, aber man sollte sich nicht darauf verlassen, dass der Anwender diesen bei der Installation so belassen hat. Abgesehen davon müsste man die Version des Betriebssystems feststellen (siehe Rezept [11.32](#) ((Informationen zum installierten Betriebssystem))).

```
Public Function GetWinDir() As String
    Dim TempPath As String
    Dim TempDirInfo As System.IO.DirectoryInfo
    Dim SysPath As String

    ' System32-Verzeichnis
    TempPath = System.Environment.SystemDirectory
    TempDirInfo = New System.IO.DirectoryInfo(TempPath)

    ' Windows-Verzeichnis eine Ebene über System32
    SysPath = TempDirInfo.Parent().ToString
    SysPath = TempDirInfo.Root.ToString + SysPath

    Return SysPath
End Function
```

Listing 301: Funktion `GetWinDir()` zur Ermittlung des Windows-Verzeichnisses

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung  
Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

Um nun das Windows-Verzeichnis zu ermitteln, kann man einen Umweg über die Ermittlung des System-Verzeichnisses nehmen. Dieses Verzeichnis ist unter dem Windows-Betriebssystem unterhalb des Windows-Verzeichnisses angesiedelt. Wie dies unter anderen Betriebssystemen sein wird, die .NET in Zukunft unterstützen sollte, steht zurzeit noch nicht fest. Eine Möglichkeit, das Windowsverzeichnis über diesen Umweg zu ermitteln, ist in Listing 301 zu sehen. Über `System.Environment` wird das System-Verzeichnis und anschließend das Parent-Verzeichnis ermittelt. Dieses wird dann als Windows-Verzeichnis zurückgegeben.

Eine andere Möglichkeit besteht in der Abfrage des Environments nach der Umgebungsvariablen `WINDIR`. In dieser Umgebungsvariablen wird beim Anmeldevorgang des Benutzers das Windows-Verzeichnis hinterlegt. Eine entsprechende Implementation ist in Listing 302 zu sehen.

---

```
Public Function GetWinDirEnv() As String
    Return System.Environment.GetEnvironmentVariable("WINDIR")
End Function
```

---

*Listing 302: Das Windows-Verzeichnis aus der Umgebungsvariablen WINDIR*

Alle weiteren Pfade kann man mit der Methode `GetFolderPath` aus dem `System.Environment`-Namensraum ableiten. Hierfür gibt es die Enumeration `SpecialFolder`, die alle entsprechenden Systempfade enthält. Wie diese Pfade hiermit ermittelt werden, ist in Listing 303 an einigen Beispielen zu sehen. Einzige Ausnahme in Listing 303 ist der Systempfad, wie er auch in Listing 301 benutzt wurde.

---

```
' Pfad zum Verzeichnis Programme
Public Function GetProgramPath() As String
    Return System.Environment.GetFolderPath( _
        Environment.SpecialFolder.ProgramFiles)
End Function

' Pfad zum System-Verzeichnis
Public Function GetSystemPath() As String
    Return System.Environment.SystemDirectory
End Function

' Pfad zum Verzeichnis der Favoriten
Public Function GetFavoritesPath() As String
    Return System.Environment.GetFolderPath( _
        Environment.SpecialFolder.Favorites)
End Function

'
Public Function GetCommonProgPath() As String
    Return System.Environment.GetFolderPath( _
        Environment.SpecialFolder.CommonProgramFiles)
End Function

'
Public Function GetMyComputerPath() As String
    Return System.Environment.GetFolderPath( _
```

---

*Listing 303: Ermittlung einiger ausgewählter Pfade mit der .NET-Methode `GetFolderPath()`*

```
        Environment.SpecialFolder.MyComputer)
End Function

'
Public Function GetCommonAppPath() As String
    Return System.Environment.GetFolderPath( _
        Environment.SpecialFolder.CommonApplicationData)
End Function

'
Public Function GetIECachePath() As String
    Return System.Environment.GetFolderPath( _
        Environment.SpecialFolder.InternetCache)
End Function

'
Public Function GetApplicationDataPath() As String
    Return System.Environment.GetFolderPath( _
        Environment.SpecialFolder.ApplicationData)
End Function

'
Public Function GetCookiePath() As String
    Return System.Environment.GetFolderPath( _
        Environment.SpecialFolder.Cookies)
End Function

'
Public Function GetDesktopPath() As String
    Return System.Environment.GetFolderPath( _
        Environment.SpecialFolder.Desktop)
End Function

'
Public Function GetLocalAppDataPath() As String
    Return System.Environment.GetFolderPath( _
        Environment.SpecialFolder.LocalApplicationData)
End Function

'
Public Function GetPersonalPath() As String
    Return System.Environment.GetFolderPath( _
        Environment.SpecialFolder.Personal)
End Function
```

Listing 303: Ermittlung einiger ausgewählter Pfade mit der .NET-Methode GetFolderPath() (Forts.)

Das Ergebnis des Beispielprogramms, welches Sie auch auf der CD finden, ist in Abbildung 165 zu sehen. Die Leerzeile in der siebten Zeile resultiert aus einer leeren Zeichenkette, die von der Funktion GetMyComputerPath zurückgegeben wurde.

|                           |
|---------------------------|
| Basics                    |
| Datum/<br>Zeit            |
| Anwen-<br>dungen          |
| Zeich-<br>nen             |
| Bild-<br>bearbei-<br>tung |
| Win-<br>dows<br>Forms     |
| Controls                  |
| Propert-<br>yGrid         |
| Datei-<br>system          |
| Netz-<br>werk             |
| System                    |
| Daten-<br>banken          |
| XML                       |
| Wissen-<br>schaft         |
| Ver-<br>schiede-<br>nes   |

## 166 Anwendungs-/Bibliotheksname des laufenden Prozesses

Will man den Namen und/oder den Pfad einer gerade aktiven Anwendung ermitteln, so kann man dies aus der entsprechenden Anwendung, bzw. aus der entsprechenden Bibliothek (DLL) heraus erfragen. Hierzu muss man die Hilfsmittel des Namensraumes `System.Reflection` in Anspruch nehmen. Innerhalb dieses Namensraumes ist die Klasse `Assembly` unter anderem für das Laden und Definieren von Assemblies und das Abfragen von Typen innerhalb dieser Assemblies zuständig. Mit Hilfe der `GetExecutingAssembly`-Methode kann man Informationen zur laufenden Assembly holen, hier den Pfad zur Assembly mit der Eigenschaft `Location`. Diese enthält den Assembly-Namen einschließlich des gesamten Pfades. Will man also den Pfad ermitteln, geht dies am einfachsten über `FileInfo` des `System.IO`- Namensraumes. Die so aufgebaute Funktion `GetAppDLLPath` ist in Listing 304 zu sehen.

---

```
Public Function GetAppDLLPath() As String
    Dim mFileInfo As System.IO.FileInfo

    ' FileInfo der DLL/Exe holen, die diese Funktion enthält
    mFileInfo = New System.IO.FileInfo( _
        System.Reflection.Assembly.GetExecutingAssembly.Location)

    ' Verzeichnis zurückgeben
    Return mFileInfo.DirectoryName
End Function
```

---

Listing 304: Pfad der aktiven Anwendung/der aktiven DLL

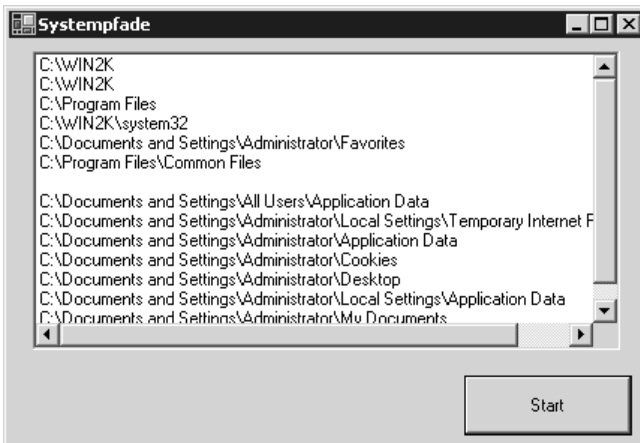


Abbildung 165: Beispiel für die Ausgabe der Systempfade

Analog kann man vorgehen, will man nur den Namen der Anwendung bzw. der Bibliothek (DLL) ermitteln. Die entsprechende Funktion `GetAppDLLFileName` ist in Listing 305 zu sehen.

```

Public Function GetAppDLLFileName() As String
    Dim mFileInfo As System.IO.FileInfo

    ' FileInfo der DLL/Exe holen, die diese Funktion enthält
    mFileInfo = New System.IO.FileInfo( _
        System.Reflection.Assembly.GetExecutingAssembly.Location)

    ' Dateinamen zurückgeben
    Return mFileInfo.Name
End Function

```

Listing 305: Name der aktiven Anwendung/Bibliothek (DLL)

Der vollständige Name, in der Hilfe auch Anzeigename genannt, enthält neben dem Anwendungsnamen ohne Erweiterung noch zusätzlich die Informationen zur Version, den Kulturkreis und den öffentlichen Schlüssel.

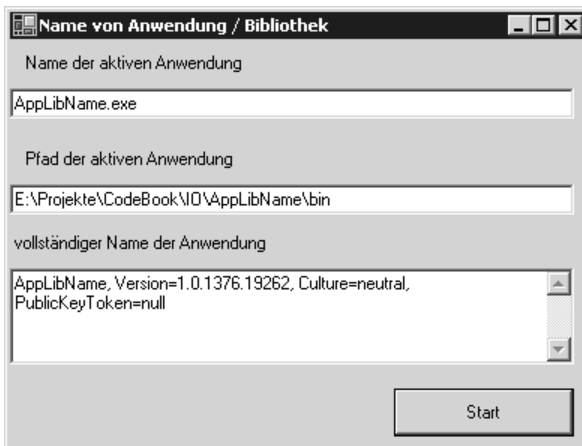


Abbildung 166: Beispiel für die Ermittlung des Anwendungs-, bzw. Bibliotheksnamens

Die entsprechende Funktion findet sich in Listing 306.

```

Public Function GetAppDLLFullName() As String

    Return _
        System.Reflection.Assembly.GetExecutingAssembly.FullName
End Function

```

Listing 306: Anzeigename/vollständiger Name der Anwendung/Bibliothek (DLL)

## 167 Existenz eines Verzeichnisses

Setzt ein Programm das Vorhandensein eines bestimmten Verzeichnisses voraus, tut man gut daran, die Existenz dieses Verzeichnisses vor Lese- oder Schreibvorgängen zu überprüfen. Man weiß ja nie, was auf einem Anwender-PC alles installiert wurde und abläuft. Um diese Abfrage kümmert sich die Funktion `ExistsDirectory` aus Listing 307.

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tung  
Win-  
dows  
Forms

Controls

Propor-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

---

```

Public Function ExistsDirectory(ByVal dir As String, _
    Optional ByVal Extended As Boolean = False) As Boolean

    Dim DirInfo As System.IO.DirectoryInfo

    Dim Exists As Boolean

    Try
        ' DirInfo kann auch erstellt werden, sollte Verzeichnis
        ' nicht existieren
        DirInfo = New System.IO.DirectoryInfo(dir)
    Catch ex As ArgumentNullException
        Throw New ApplicationException( _
            "ExistsDirectory: Übergabeparameter ist Nothing", ex)
    Catch ex As ArgumentException
        Throw New ApplicationException( _
            "ExistsDirectory: Übergabeparameter ist leer", ex)
    Catch ex As Exception
        Throw New ApplicationException( _
            "ExistsDirectory: Generelle Ausnahme", ex)
    End Try

    ' Rückgabe von DirInfo.Exists kann evtl. nicht reichen
    If DirInfo.Exists = True Then
        Exists = True
    Else
        Exists = False
    End If

    ' Eine Datei mit diesem Namen vorhanden?
    If Extended = True Then
        If DirInfo.Attributes >= 0 Then
            Exists = True
        End If
    End If

    Return Exists
End Function

```

---

Listing 307: Existenz eines Verzeichnisses überprüfen

Der erste Übergabeparameter ist der Name des Verzeichnisses, das überprüft werden soll. Der zweite, optionale Parameter erweitert die Funktion für den Fall, dass das abgefragte Verzeichnis erstellt werden soll, wenn es noch nicht existieren sollte.

Es wird in der Funktion zuerst versucht, mit dem Verzeichnisparameter ein `DirectoryInfo`-Objekt zu erstellen. Bei groben Fehlern werden entsprechende Ausnahmen generiert. Ist dies nicht der Fall, kann mit der `Exists`-Eigenschaft geprüft werden, ob dieses Verzeichnis existiert.

Dieses Ergebnis sagt aber nichts darüber aus, ob man ein entsprechendes Verzeichnis erstellen kann. Sollte eine Datei gleichen Namens existieren, schlägt das Erzeugen eines solchen Unterverzeichnisses fehl. Hier führen aber die Eigenschaften von `DirInfo` weiter. Sollte es eine Datei gleichen Namens geben, enthält das `DirInfo`-Objekt die Attribut- und Datumswerte dieser Datei.



Gibt es eine solche Datei nicht, ist die Eigenschaft `Attributes` kleiner Null und die Datumswerte lauten auf das Jahr 1601. Dies wird in der Funktion aus Listing 307 dazu ausgenutzt, festzustellen, ob eine solche Datei existiert. Wird der Funktion im zweiten Parameter der Wert `True` übergeben, wird zusätzlich die `Attributes`-Eigenschaft getestet und die Funktion liefert einen entsprechend modifizierten Wert zurück. So kann man sichergehen, auch ein Unterverzeichnis mit dem getesteten Namen erstellen zu können.

Die abgefangenen unterschiedlichen Ausnahmen werden einheitlich als `ApplicationException` dem aufrufenden Programm übergeben. So spart man sich im aufrufenden Programm mehrere `Catch`-Zweige.

## 168 Verzeichnis erstellen

Hat man mit der Funktion `ExistsDirectory` aus Listing 307 festgestellt, dass man ein Verzeichnis mit dem gewünschten Namen erstellen kann, so kann anschließend mit der Funktion `CreateDirectory` aus Listing 308 dieses Verzeichnis erstellt werden.

```
Public Function CreateDirectory(ByVal dir As String) As Boolean

    Dim DirInfo As System.IO.DirectoryInfo

    Try
        DirInfo = New System.IO.DirectoryInfo(dir)
    Catch ex As ArgumentNullException
        Throw New ApplicationException( _
            "CreateDirectory: Verzeichnisangabe ist Nothing.", ex)
    Catch ex As ArgumentException
        Throw New ApplicationException( _
            "CreateDirectory: Verzeichnisangabe ist falsch.", ex)
    End Try

    Try
        DirInfo.Create()
    Catch ex As IO.IOException
        Throw New ApplicationException( _
            "CreateDirectory: Verzeichnis kann nicht erstellt werden.", _
            ex)
    End Try

    Return True
End Function
```

Listing 308: Erstellen eines Verzeichnisses

Der Funktion wird der Name des Verzeichnisses als Zeichenkette übergeben. Hier kann ein beliebiger Pfad angegeben werden. Sollten mehrere Unterverzeichnisse erstellt werden müssen, um das eigentliche Verzeichnis erstellen zu können, wird dies von der `Create`-Methode mit erledigt. Gibt es also beispielsweise unterhalb von `C:\TEMP` kein weiteres Unterverzeichnis, so kann mit der Funktion `CreateDirectory` direkt das Verzeichnis `C:\TEMP\t_1\t_2\t_3` erstellt werden. Man sollte bei der Nutzung dieser Funktion entsprechend Vorsicht walten lassen, da man sonst bei einem kleinen Schreibfehler einen komplett neuen Verzeichnisbaum aufmachen kann.

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Propert-  
yGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

Die Funktion versucht mit dem übergebenen Pfad ein `DirectoryInfo`-Objekt zu erstellen. Sollte die Verzeichnisangabe leer sein oder gar nicht existieren, werden entsprechende Ausnahmen ausgelöst. Konnte das Objekt `DirInfo` erstellt werden, wird mit der `Create`-Methode versucht, dieses Verzeichnis zu erstellen. Sollte es hierbei zu Problemen kommen, wird eine `IOException` aufgefangen und mit einem Kommentar versehen an das aufrufende Programm zurückgegeben. Auch hier spart man sich im Hauptprogramm durch die einheitliche Behandlung der unterschiedlichen Ausnahmen in der Funktion mehrere `Catch`-Zweige im aufrufenden Programm.

## 169 Verzeichnis löschen

Das Löschen von Verzeichnissen geschieht analog zum Erstellen der Verzeichnisse. Es werden das zu löschende Verzeichnis und ein logischer Wert für die Unterverzeichnisse der Funktion `DeleteDirectory` aus Listing 309 übergeben.

---

```
Public Function DeleteDirectory(ByVal dir As String, _
    ByVal SubDirs As Boolean) As Boolean

    Dim DirInfo As System.IO.DirectoryInfo

    Try
        DirInfo = New System.IO.DirectoryInfo(dir)
    Catch ex As ArgumentNullException
        Throw New ApplicationException( _
            "DeleteDirectory: Verzeichnisangabe ist Nothing.", ex)
    Catch ex As ArgumentException
        Throw New ApplicationException( _
            "DeleteDirectory: Verzeichnisangabe ist falsch.", ex)
    End Try

    Try
        DirInfo.Delete(SubDirs)
    Catch ex As IOException
        Throw New ApplicationException( _
            "DeleteDirectory: Verzeichnis kann nicht gelöscht werden.", _
            ex)
    End Try

    Return True
End Function
```

---

*Listing 309: Löschen eines Verzeichnisses*

Der logische Parameter `SubDirs` ist bewusst ohne Default-Wert angegeben, um eine ausdrückliche Entscheidung des Programmierers für diesen Wert zu erzwingen. Neben den Unterverzeichnissen werden auch alle Dateien ohne weitere Nachfrage gelöscht, sollte dieser Wert auf `True` gesetzt sein.

## 170 Verzeichnis umbenennen/verschieben

Will man ein Verzeichnis umbenennen bzw. ein Verzeichnis einschließlich aller Unterverzeichnisse im Dateisystem verschieben, kann die Funktion `MoveDirectory` aus Listing 310 benutzt werden. Dieser Funktion werden der komplette umzubennende/zu verschiebende Pfad und der

Zielpfad übergeben. Hierbei ist darauf zu achten, dass der Zielpfad nicht schon existiert. Hierzu kann die Funktion aus Listing 307 benutzt werden. Um sicherzustellen, dass die Übergabeparameter nicht vollkommen falsch sind, könnte wieder ein einfacher Try ... Catch-Block für diese Aktion angewandt werden. In Listing 310 wird ein etwas anderer Weg eingeschlagen. Statt hierfür ein DirectoryInfo-Objekt zu erstellen, werden die Angaben zu den Quell- und Zielverzeichnissen mit klassischen String-Operationen auf Gültigkeit getestet und mit der Shared-Methode Move der Klasse Directory aus dem Namensraum System.IO verschoben/umbenannt.

```
Public Function MoveDirectory(ByVal Src As String, _
    ByVal Dest As String) As Boolean

    ' Hier ginge auch ArgumentNullException im Try-Block
    If Src Is Nothing Then
        Throw New ApplicationException( _
            "Quellverzeichnis fehlt im Aufruf.")
    End If

    ' Hier ginge auch ArgumentException im Try-Block
    If Src.Equals(String.Empty) Then
        Throw New ApplicationException( _
            "Quellverzeichnis ist leer.")
    End If

    If Dest Is Nothing Then
        Throw New ApplicationException( _
            "Zielverzeichnis fehlt im Aufruf.")
    End If

    If Dest.Equals(String.Empty) Then
        Throw New ApplicationException( _
            "Zielverzeichnis ist leer.")
    End If

    Try
        System.IO.Directory.Move(Src, Dest)
    Catch ex As System.Security.SecurityException
        Throw New ApplicationException("Zugriff verweigert.", ex)
    Catch ex As System.IO.IOException
        Throw New ApplicationException( _
            "Anderes Laufwerk oder Verzeichnis existiert schon", ex)
    End Try

    Return True
End Function
```

Listing 310: Listing umbenennen/verschieben

Bei dieser Aktion können unter anderem zwei Ausnahmen auftreten: das Verzeichnis existiert schon, ein Fehler, der mit den oben genannten Maßnahmen nicht auftreten sollte, oder es liegt keine ausreichende Berechtigung für das Verschieben/Umbenennen vor. Diese Fehler werden abgefangen und dem aufrufenden Programm als `ApplicationException` übermittelt.

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Proper-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

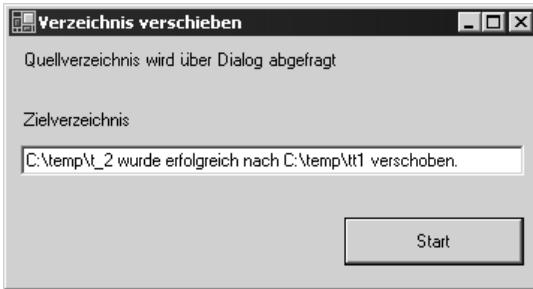


Abbildung 167: Verzeichnis umbenennen/verschieben

In Abbildung 167 sieht man das Ergebnis des Testprogramms. Der Einfachheit halber wird das Ergebnis in die TextBox für das Zielverzeichnis geschrieben.

## 171 Verzeichnis kopieren

Neben dem Erstellen, dem Verschieben und Umbenennen von Verzeichnissen kann man ebenfalls einen Verzeichnisbaum kopieren. Hierzu gibt es im .NET Framework weder in der `Directory` noch in der `DirectoryInfo`-Klasse entsprechende Methoden. Aber es gibt eine Lösung über die Scripting-Engine von Microsoft.

---

```
Public Function CopyDir(ByVal src As String, ByVal dest _
    As String, ByVal WriteOver As Boolean) As Boolean

    Dim fso As FileSystemObject = New FileSystemObject
    Dim FSODir As Folder

    ' Ist src ein Verzeichnis?
    Try
        FSODir = fso.GetFolder(src)
    Catch
        Throw New ApplicationException _
            ("Quellpfad <" + src + "> ist kein Verzeichnis.")
    End Try

    ' Alle Unterverzeichnisse
    src += "\*"

    ' Ist dest ein Verzeichnis?
    Try
        FSODir = fso.GetFolder(dest)
    Catch
        Throw New ApplicationException _
            ("Zielpfad <" + dest + "> ist kein Verzeichnis.")
    End Try

    Try
        fso.CopyFolder(src, dest, WriteOver)
    Catch ex As Exception
```

---

Listing 311: Verzeichnis kopieren

```

Throw New ApplicationException _
    ("CopyDir: Kopieren schlug fehl.", ex)
End Try

' Okay, verschieben
' fso.MoveFolder(src, dest)
End Function

```

Listing 311: Verzeichnis kopieren (Forts.)

Hierzu muss über PROJEKT / VERWEIS EINFÜGEN ... ein Verweis auf die *scrnun.dll* in das Projekt eingefügt werden (siehe Abbildung 168). Ist dieser Verweis vorhanden, muss mit Imports Scripting der hiermit erstellte Namensraum dem Programm bekannt gemacht werden.

Damit stehen dem Programm/der Funktion alle Methoden der *Scripting Runtime* zur Verfügung, unter anderem auch die *File System Objects* (FSO).

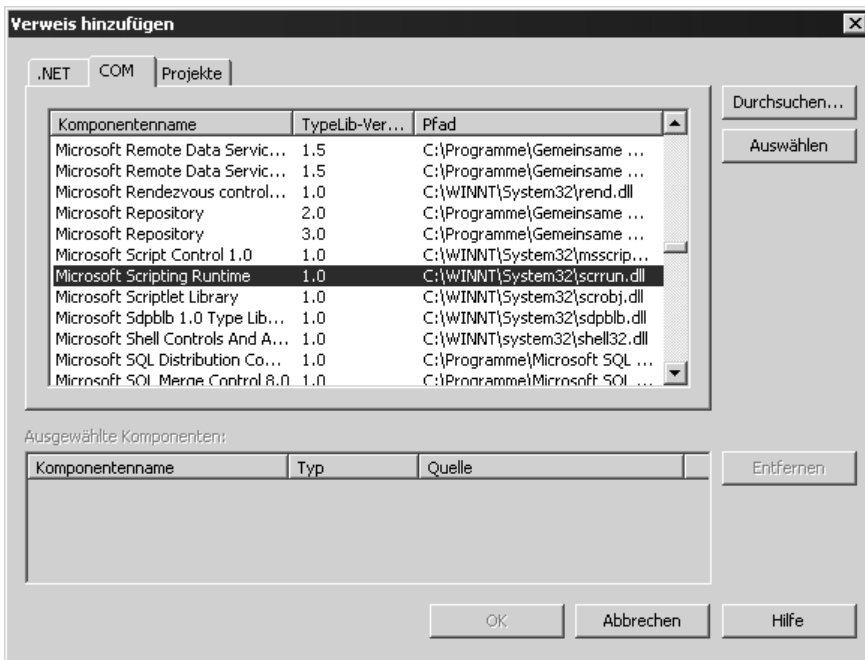


Abbildung 168: Verweis auf die Scripting Runtime einbinden

Die Funktion `CopyDir` aus Listing 311 nutzt diese Methoden. Es wird ein Objekt `fso` der Klasse `FileSystemObject` definiert und ein Objekt `FSODir` der Klasse `Folder` deklariert. Anschließend wird versucht, aus dem Quellpfad das `Folder`-Objekt zu erstellen. Dies wird als Test benutzt, ob der übergebene Quellpfad überhaupt ein Verzeichnis ist. Sollte dies nicht der Fall sein, wird die Funktion mit einer Ausnahme verlassen.

Da auch eventuell vorhandene Unterverzeichnisse kopiert werden sollen, wird der Quellpfad um die Zeichenkette `»\*«` erweitert. Für den Zielpfad wird ebenfalls dieser Test durchgeführt und nötigenfalls mit einer Ausnahme aus der Funktion heraus gesprungen. Wie ein Anwendungsprogramm eine solche Ausnahme zu spüren bekommt, ist in Abbildung 169 zu sehen. Sind diese

Tests bestanden, wird mit der Methode `CopyFolder` des `FileSystemObject`-Objektes der Verzeichnisbaum kopiert. Hierbei ist der letzte Parameter des Aufrufes interessant, dessen Wert auch der Funktion `CopyDir` mit dem Parameter `WriteOver` übergeben wird. Dieser Parameter gibt an, ob ein bereits vorhandenes Verzeichnis mit diesem Namen überschrieben werden soll (`True`) oder nicht (`False`).

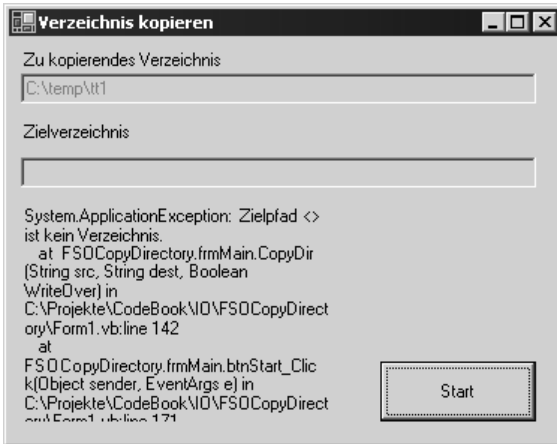


Abbildung 169: Verzeichnis kopieren mit Fehlermeldung

Wie man dem letzten Kommentar aus Listing 311 entnehmen kann, muss nur der Aufruf der Methode `CopyFolder` in `MoveFolder` umgeändert werden, um mit dieser Methode einen Verzeichnisbaum auch zu verschieben (siehe auch Kapitel [9.6](#)).

## 172 Verzeichnisgröße mit Unterverzeichnissen

In größeren Netzwerken ist es manchmal sehr nützlich, wenn man den verbrauchten Speicherplatz in den einzelnen Verzeichnissen, zum Beispiel auf einem Server, feststellen kann. Leider bietet Windows von Haus aus keine solche Funktion an. Es gibt zwei Möglichkeiten, dieses Problem zu beheben. Nachschauen, ob schon jemand anderes dieses Problem für uns gelöst hat, oder selber programmieren. Da es sich hier um ein Buch für Programmierer handelt, bleibt nur noch eine Lösungsmöglichkeit übrig. Das Ergebnis dieser Bemühungen ist in Abbildung 170 zu sehen. Um dieses Ziel zu erreichen, müssen die Verzeichnisse und die jeweiligen Unterverzeichnisse nacheinander abgearbeitet werden. Hierzu bietet sich ein rekursives Aufrufen einer entsprechenden Funktion an.

In Listing 312 ist der Einsprungspunkt für diese Bearbeitung aufgelistet. Der Funktion `DirSize` wird das Verzeichnis übergeben, ab dem die Größe der Unterverzeichnisse ermittelt werden soll. Zusätzlich wird eine Variable vom Typ `Hashtable` erzeugt, die das Ergebnis der Berechnungen aufnehmen soll.

Nach einem Check des Verzeichnisnamens wird die eigentliche Routine aufgerufen: `ProcessDirectory`. Diesem Unterprogramm wird neben der Verzeichnisangabe die soeben erstellte `Hashtable` übergeben.

```

Public Function DirSize(ByVal Dir As String) As Hashtable
    Dim path As String
    Dim Table As Hashtable = New Hashtable

    ' Wo nichts ist, kann man nichts berechnen
    If Dir Is Nothing Then
        Throw New ApplicationException("Verzeichnis = Nothing.")
    End If

    If Dir.Equals(String.Empty) = True Then
        Throw New ApplicationException("Verzeichnis ist leer.")
    End If

    If Directory.Exists(Dir) Then
        ProcessDirectory(Dir, Table)
    End If
    Return Table
End Function

```

Listing 312: Funktion *DirSize*, Einstiegspunkt in die rekursive Bearbeitung

Wie man im Listing 313 erkennen kann, wird diese Tabelle als Referenz übergeben. Der Grund liegt in der nicht kalkulierbaren Größe der erstellten Tabelle. Und jeder Aufruf der Routine müsste bei einer Übergabe per Wert die gesamte Tabelle kopieren. Dies kann sehr viel Zeit bedeuten, abgesehen von den vielen Kopien der Tabelle im Hauptspeicher.

In dieser Routine werden die Dateien innerhalb des aktuellen Verzeichnisses über die *Shared-Methode GetFiles* der Klasse *Directory* ermittelt und in ein Zeichenkettenarray abgelegt. Analog werden die Verzeichnisse mit Hilfe der Methode *GetDirectories* der gleichen Klasse abgefragt.

```

Public Sub ProcessDirectory _
    (ByVal Dest As String, _
    ByRef Table As Hashtable)

    Dim Subdirectory As String
    Dim FileName As String
    Dim Size As Long = 0

    ' Dateien im aktuellen Verzeichnis
    Dim fileEntries As String() = _
        Directory.GetFiles(Dest)

    ' Verzeichnisse im aktuellen Verzeichnis
    Dim Subdirectories As String() = _
        Directory.GetDirectories(Dest)

    For Each FileName In fileEntries
        Size += ProcessFile(FileName)
    Next FileName

    Table.Add(Dest, Size)

```

Listing 313: Rekursive Verzeichnisbearbeitung

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

```

For Each Subdirectory In Subdirectories
    ProcessDirectory(Subdirectory, Table)
Next Subdirectory

```

```
End Sub
```

---

*Listing 313: Rekursive Verzeichnisbearbeitung (Forts.)*

Für jede gefundene Datei wird die Funktion `ProcessFile` aufgerufen, die die aktuelle Datei bearbeitet. Diese Funktion liefert die Größe der Datei in Bytes zurück, siehe Listing 314. Eine ausführlichere Version zur Ermittlung von Dateigrößen ist in Kapitel 9.14 (Listing 323) erklärt. In der Schleife über alle Dateien wird so die Gesamtgröße des im Verzeichnis verbrauchten Plattenplatzes durch die Dateien errechnet. Das Ergebnis wird mit dem Namen des Verzeichnisses als Schlüssel und der Gesamtgröße des Verzeichnisses in der Hash-Tabelle abgespeichert. Schlussendlich ruft sich die Routine für jedes gefundene Verzeichnis selber auf. Auf diese Weise durchläuft die Funktion jedes Unterverzeichnis und kann so den Verbrauch an Plattenplatz feststellen.

---

```

Public Function ProcessFile(ByVal FileName As String) As Long

    Dim mFile As FileInfo = New FileInfo(FileName)

    Return mFile.Length

End Function

```

---

*Listing 314: Informationsermittlung für eine Datei*

Die Funktion `ProcessFile` liefert in Listing 314 die Größe der übergebenen Datei in Bytes zurück. Da Hash-Tabellen nicht so bekannt sind wie gewöhnliche Arrays, ist in Listing 315 das Beispielprogramm aufgelistet, welches zur Abbildung 170 führt.

---

```

Private Sub btnStart_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnStart.Click

    ' Nimmt die Verzeichnis-Einträge auf
    Dim Table As Hashtable
    ' Zum Durchlaufen der Hash-Tabelle
    Dim var As IDictionaryEnumerator

    ' Dialog für Verzeichnis öffnen
    If fb.ShowDialog = DialogResult.OK Then
        Table = DirSize(fb.SelectedPath)
    End If

    var = Table.GetEnumerator

    ' Schleife über alle Einträge der Hash-Tabelle
    While var.MoveNext

```

---

*Listing 315: Beispielprogramm für die Ermittlung von Verzeichnisgrößen*



```

        blList.Items.Add(var.Key.ToString + " : " + var.Value.ToString)
    End While

End Sub
End Class

```

Listing 315: Beispielprogramm für die Ermittlung von Verzeichnisgrößen (Forts.)

Durch Auslösen des START-Buttons wird die Routine in Listing 315 ausgelöst, die als Erstes eine Hash-Tabelle deklariert, die das Ergebnis aufnehmen soll. Um diese Tabelle durchlaufen zu können, wird die Variable `var` vom Typ `IDictionaryEnumerator` benötigt, die ebenfalls deklariert wird. Das Verzeichnis, ab dem die kumulierte Dateigröße angezeigt werden soll, wird durch einen `FolderBrowserDialog` ausgewählt und der Funktion `DirSize` übergeben.

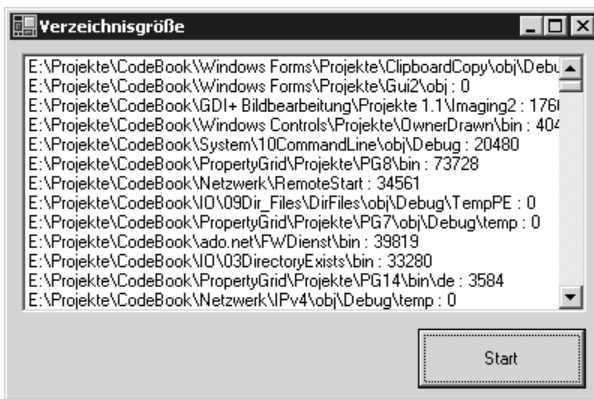


Abbildung 170: Ausgabe der Funktion zur Verzeichnisgröße

Um die Tabelle nun durchlaufen zu können, wird die Variable `var` mit dem Resultat aus dem Aufruf der `GetEnumerator`-Methode der Hash-Tabelle belegt. Damit steht die Variable `var` vor dem ersten gültigen Wert innerhalb der Hash-Tabelle. Eine Schleife über alle Einträge dieser Tabelle muss also mit dem Aufruf von `MoveNext` der Variablen beginnen. Dieser Aufruf liefert so lange `True` zurück, bis das Ende der Tabelle erreicht ist. Mit dieser Eigenschaft kann man sehr gut eine Schleife über alle Einträge erstellen, wie dies am Ende von Listing 315 geschieht.

## 173 Existenz einer bestimmten Datei

Will man das Vorhandensein einer bestimmten Datei überprüfen, so kann dies recht einfach mit den Mitteln des .NET Frameworks getestet werden. Oft wird auch noch die Information benötigt, ob diese Datei eine gewisse Größe unter- oder überschreitet. Dieser Test kann mit der Funktion aus Listing 316 durchgeführt werden.

Der Funktion wird der zu überprüfende Dateiname und optional die erforderliche Dateigröße übergeben. Nach der Variablendeklaration wird versucht, ein Objekt der Klasse `FileInfo` zu erzeugen. Werden hierbei keine der bekannten Ausnahmen generiert, ist das Objekt vorhanden und kann auf sein Vorhandensein im Dateisystem geprüft werden.

---

```

Public Function ExistsFile(ByVal FileName As String, _
    Optional ByVal GreaterSize As Long = -1) As Boolean

    ' Datei-Objekt
    Dim mFileInfo As System.IO.FileInfo

    ' Datei vorhanden
    Dim Exists As Boolean

    ' Erzeugen des Datei-Objektes und grober Test auf Gültigkeit
    Try
        mFileInfo = New System.IO.FileInfo(FileName)
    Catch ex As ArgumentNullException
        Debug.WriteLine(ex.ToString)
    Catch ex As ArgumentException
        Debug.WriteLine(ex.ToString)
    End Try

    If mFileInfo.Exists = True Then
        ' Spezialisierte Einschränkung der Existenz
        If mFileInfo.Length > GreaterSize Then
            Exists = True
        End If
    Else
        Exists = False
    End If

    Return Exists
End Function

```

---

Listing 316: Vorhandensein einer Datei überprüfen

Nach dem Check mit der Methode `Exists` des Objektes wird überprüft, ob die Datei eine entsprechende Größe überschreitet. Wurde keine Dateigröße beim Aufruf der Funktion übergeben, wird als Default-Wert `-1` festgelegt, was ein üblicher Defaultwert ist und zudem mit dieser Wahl in der entsprechenden Abfrage (`>`) eine Dateigröße von Null Bytes erlaubt.

## 174 8.3 Dateinamen

Das Format 8.3 für Dateinamen stammt noch aus der Zeit des DOS-Betriebssystems. Aber es gibt immer noch eine relativ große Menge an Programmen, die nur dieses Format unterstützen. Diese haben sich aus der DOS/Win 3.11-Zeit bis heute herübergerettet. Um eine Datei in diesem Format zu erzeugen, eventuell auch mit entsprechendem Pfad, dient die Funktion `Get83Name` aus Listing 317. Dieser Funktion werden 4 Parameter übergeben: der lange Dateiname einschließlich Pfad, so wie er in den neueren Betriebssystemen durchaus erlaubt ist, einem logischen Parameter, der darüber entscheidet, ob die Funktion mit oder ohne 8.3-Pfad zurückgegeben wird, einem logischen Parameter, der das Speichern in der Zwischenablage regelt, und zu guter Letzt einem logischen Parameter, der darüber entscheidet, ob der Eintrag auch nach Beendigung des Programms in der Zwischenablage verbleibt.

```

Public Function Get83Name(ByVal Filename As String, _
    Optional ByVal CompletePath As Boolean = False, _
    Optional ByVal InClipboard As Boolean = False, _
    Optional ByVal StayInClipboard As Boolean = False) As String

    Dim FI As System.IO.FileInfo

    ' aus der Scripting Runtime
    Dim fso As Scripting.FileSystemObject
    Dim fsoFile As Scripting.File
    Dim ShortName As String

    Try
        FI = New System.IO.FileInfo(Filename)
    Catch ex As System.IO.IOException
        Throw New ApplicationException("Get83Name IO: ", ex)
    Catch ex As ArgumentNullException
        Throw New ApplicationException("Get83Name: " + _
            Filename + " ist Nothing.", ex)
    Catch ex As ArgumentException
        Throw New ApplicationException("Get83Name: " + _
            Filename + " ist leer.", ex)
    Catch ex As Exception
        Throw New ApplicationException("Get83Name: ", ex)
    End Try

    fso = New Scripting.FileSystemObject
    fsoFile = fso.GetFile(Filename)

    ShortName = ""
    If CompletePath = True Then
        ShortName = fsoFile.ShortPath
    Else
        ShortName = fsoFile.ShortName
    End If

    ' In die Zwischenablage?
    If InClipboard = True Then
        Clipboard.SetDataObject(ShortName, StayInClipboard)
    End If

    Return ShortName
End Function

```

Listing 317: Erzeugen von 8.3 Dateinamen/Pfaden

Um einen 8.3-Dateinamen zu erzeugen muss auf die Scripting Runtime zurückgegriffen werden, da .NET eine solche Namenskonvertierung nicht unterstützt. Zur Einbindung der Scripting Runtime sind einige Anmerkungen in Kapitel 9.7 zu finden. Zu Beginn der Funktion wird versucht, ein Objekt der Klasse `FileInfo` zu erzeugen. Dies dient einem groben Test auf die Gültigkeit des Dateinamens, indem entsprechende Ausnahmen abgefangen werden und dem aufrufenden Programm einheitlich als `ApplicationException` gemeldet werden.

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Propor-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

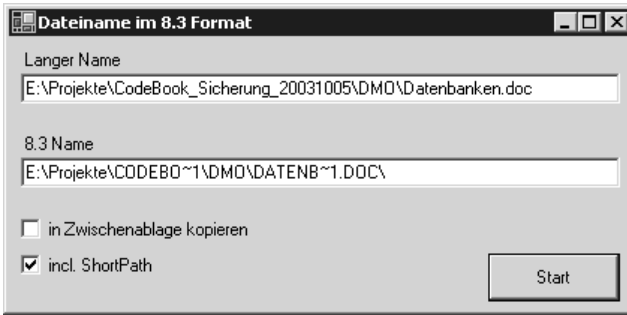


Abbildung 171: Erzeugen eines 8.3 Dateinamens/Pfad

Mit der Erzeugung des Objektes `fso` der Klasse `FileSystemObject` wird die Grundlage für das File-Objekt gelegt. Dieses wird mit der Methode `GetFile` des `fso`-Objektes ermittelt. Soll der komplette Pfad ausgegeben werden, wird dieser nun über die Eigenschaft `ShortPath` abgefragt. Zum Schluss wird den Übergabeparametern entsprechend dieser Kurzname in die Zwischenablage kopiert. Hierzu wird die `Clipboard`-Klasse aus dem Namensraum `Systems.Windows.Forms` benutzt, die bei Windows-Forms-Programmen standardmäßig eingeblendet ist. Der zweite Parameter der Routine `SetDataObject` legt fest, ob der Eintrag nach Beendigung des Programms in der Zwischenablage verbleibt. Ein Beispiel für den Aufruf dieser Funktion ist in Abbildung 171 zu sehen.

## 175 Datei umbenennen/verschieben

Das Umbenennen bzw. das Verschieben einer Datei basiert auf der gleichen Grundlage, da in der Praxis beide Aktionen identisch sind. Eine entsprechende Funktion ist in Listing 318 mit dem Namen `RenMoveFile` realisiert. Dieser Funktion werden alter und neuer Name als Parameter übergeben.

---

```
Public Function RenMoveFile(ByVal OldName As String, _
    ByVal NewName As String) As Boolean

    If OldName Is Nothing Then
        Throw New ApplicationException(OldName + " ist Nothing.")
    End If

    If OldName.Equals(String.Empty) Then
        Throw New ApplicationException(OldName + " ist leer.")
    End If

    If NewName Is Nothing Then
        Throw New ApplicationException(NewName + " ist Nothing.")
    End If

    If NewName.Equals(String.Empty) Then
        Throw New ApplicationException(NewName + " ist leer.")
    End If
```

---

Listing 318: Datei umbenennen/verschieben

```

Try
    System.IO.File.Move(OldName, NewName)
Catch ex As System.Security.SecurityException
    Throw New ApplicationException("Zugriff verweigert.", ex)
Catch ex As System.IO.IOException
    Throw New ApplicationException( _
        "Andere Datei existiert schon", ex)
Catch ex As Exception
    Throw New ApplicationException(OldName + " kann nicht In " _
        + NewName + "gewandelt werden.")
End Try

Return True
End Function

```

*Listing 318: Datei umbenennen/verschieben (Forts.)*

Nach einem Check der Namen wird mit der `Move`-Methode des `File`-Objektes aus dem `System.IO`-Namensraum versucht, diese Datei entsprechend zu verschieben. Ausnahmen, die auftreten könnten, wie zum Beispiel die Ausnahme wegen Berechtigungsproblemen, werden abgefangen und dem aufrufenden Programm einheitlich als `ApplicationException` gemeldet.

## 176 Datei kopieren

Für das Kopieren einer Datei existiert eine entsprechende Methode in der Klasse `File` des `System.IO`-Namensraumes. Um das Abfangen der unterschiedlichen Ausnahmen zu vereinheitlichen, kann man auch diese Methode in einer eigenen Funktion kapseln. Dies geschieht mit der Funktion `CopyFile` aus Listing 319. Dieser Funktion werden der Quellname und der Zielname der Datei übergeben. Der dritte optionale Parameter legt fest, ob eine eventuell schon vorhandene Datei überschrieben werden kann oder nicht. Dieser Parameter wird auf `False` gesetzt, sollte vom aufrufenden Programm dieser Parameter nicht gesetzt werden.

```

Public Function CopyFile(ByVal OldName As String, _
    ByVal NewName As String, Optional ByVal over As Boolean = False) _
    As Boolean

    ' Ist die Zeichenkette für den alten Namen überhaupt initialisiert?
    If OldName Is Nothing Then
        Throw New ApplicationException(OldName + " ist Nothing.")
    End If

    ' Ist die Zeichenkette für den alten Namen leer?
    If OldName.Equals(String.Empty) Then
        Throw New ApplicationException(OldName + " ist leer.")
    End If

    ' Ist die Zeichenkette für den neuen Namen überhaupt initialisiert?
    If NewName Is Nothing Then
        Throw New ApplicationException(NewName + " ist Nothing.")
    End If

```

*Listing 319: Kopieren einer Datei*

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Proper-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

---

```

' Ist die Zeichenkette für den neuen Namen leer?
If NewName.Equals(String.Empty) Then
    Throw New ApplicationException(NewName + " ist leer.")
End If

Try
    ' Kopierversuch
    System.IO.File.Copy(OldName, NewName, over)
Catch ex As System.Security.SecurityException
    Throw New ApplicationException("Zugriff verweigert.", ex)
Catch ex As System.IO.IOException
    ' Man kann zwar überschreiben, aber falls Überschreiben nicht
    ' gewünscht, kann eine Ausnahme auftreten
    Throw New ApplicationException( _
        "Andere Datei existiert schon", ex)
Catch ex As Exception
    Throw New ApplicationException(OldName + " kann nicht nach" _
        + NewName + " kopiert werden.")
End Try

Return True
End Function

```

---

Listing 319: Kopieren einer Datei (Forts.)

Nach dem Test auf eine verwertbare Zeichenkette für Quell- und Zielname der Datei wird mit der `Shared`-Methode `Copy` der Klasse `File` versucht, die Datei zu kopieren. Auch bei dieser Methode entscheidet der dritte Parameter über das Überschreiben einer bereits mit gleichem Namen vorhandener Datei. Hierbei auftretende Probleme, wie zum Beispiel Berechtigungsprobleme, werden als `ApplicationException` an das aufrufende Programm gemeldet.

## 177 Dateiversion feststellen

Jeder .NET-Anwendung werden Informationen zur Version und zu den Eigentümerrechten der Assembly mit gegeben. Auf diese Informationen kann man Einfluss nehmen, wenn man entsprechende Angaben in der Datei `AssemblyInfo.vb` hinterlässt. Diese wird vom VISUAL STUDIO jedem Projekt automatisch beigelegt. Die Assembly-Datei für das Testprogramm zu diesem Kapitel ist in Listing 320 abgedruckt. Die Angaben sind so weit vorbereitet, dass man nur noch die entsprechenden Angaben ausfüllen muss.

---

```

Imports System
Imports System.Reflection
Imports System.Runtime.InteropServices

' Allgemeine Informationen über eine Assembly werden über die folgende
' Attributgruppe gesteuert. Ändern Sie diese Attributwerte, um Informationen,
' die mit einer Assembly verknüpft sind, zu bearbeiten.

' Die Werte der Assemblyattribute überprüfen

```

---

Listing 320: Die Datei `AssemblyInfo.vb` zum Testprogramm der Funktion `GetFileVersionInfo`

```

<Assembly: AssemblyTitle("Killerapplication")>
<Assembly: AssemblyDescription("The best Killerapplication ;-")>
<Assembly: AssemblyCompany("Barchfeld")>
<Assembly: AssemblyProduct("Killerfunction")>
<Assembly: AssemblyCopyright("2003 by Barchfeld")>
<Assembly: AssemblyTrademark("no Trademark")>
<Assembly: CLSCompliant(True)>

' Die folgende GUID ist für die ID der Typbibliothek, wenn dieses
' Projekt in COM angezeigt wird
<Assembly: Guid("C31AC426-7B81-40CE-870E-F01F0C8C99E0")>

' Versionsinformationen für eine Assembly bestehen aus den folgenden
' vier Werten:
'
'     Hauptversion
'     Nebenversion
'     Buildnummer
'     Revisionsnummer
'
' Sie können alle Werte angeben oder auf die standardmäßigen Build-
' und Revisionsnummern zurückgreifen, indem Sie '*' wie unten
' angezeigt verwenden:

<Assembly: AssemblyVersion("1.0.*")>

```

Listing 320: Die Datei AssemblyInfo.vb zum Testprogramm der Funktion GetFileVersionInfo (Forts.)

Die so abgelegten Informationen können aus der kompilierten Version des Programms oder der Bibliothek wieder extrahiert werden. Die Funktion GetFileVersionInfo aus Listing 321 erledigt diese Aufgabe, indem eine ArrayList mit den entsprechenden Angaben gefüllt wird.

```

Public Function GetFileVersionInfo(ByVal name As String) _
    As ArrayList

    Dim Liste As ArrayList = New ArrayList
    Dim Info As FileVersionInfo

    ' Informationen über FileVersionInfo-Objekt
    Try
        Info = FileVersionInfo.GetVersionInfo(name)
    Catch ex As ArgumentNullException
        Throw New ApplicationException("GetFileVersionInfo: " + _
            name + " ist Nothing.", ex)
    Catch ex As ArgumentException
        Throw New ApplicationException("GetFileVersionInfo: " + _
            name + " ist leer.", ex)
    Catch ex As Exception
        Throw New ApplicationException("GetFileVersionInfo: " + _
            name + " Ausnahme: ", ex)

```

Listing 321: Funktion zur Ermittlung von Datei-Versions-Informationen

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Proper-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

---

End Try

```
Liste.Add("Firma = " + Info.CompanyName)
Liste.Add("Produktversion = " + Info.ProductVersion)
Liste.Add("Sprache = " + Info.Language)
Liste.Add("(c) = " + Info.LegalCopyright)
Liste.Add("Name = " + Info.FileName)
Liste.Add("Build - Name = " + Info.OriginalFilename)
Liste.Add("Debug - Version = " + Info.IsDebug.ToString)
Liste.Add("Dateibeschreibung = " + Info.Comments)
Liste.Add("Dateiversion = " + Info.FileVersion)
```

```
Return (Liste)
```

```
End Function
```

---

*Listing 321: Funktion zur Ermittlung von Datei-Versions-Informationen (Forts.)*

Hierzu wird ein Objekt der Klasse `FileVersionInfo` aus dem Namensraum `System.Diagnostics` erstellt. Dem Konstruktor der Klasse wird der Name der Datei übergeben. Sollten hierbei Ausnahmen generiert werden, werden diese dem aufrufenden Programm als `ApplicationException` gemeldet.



*Abbildung 172: Ausgabebeispiel zu Datei-Versions-Informationen*

Dem so erstellten Objekt `Info` können die Assembly-Informationen über die entsprechenden Eigenschaften des Objektes abgerufen werden. Interessant hierbei ist, dass die in der `Assembly-Info.vb`-Datei in der Eigenschaft `AssemblyDescription` abgespeicherten Daten nicht über die Eigenschaft `FileDescription`, sondern über `Comments` abgerufen werden müssen.

Das Ergebnis dieses Funktionsaufrufs für das Testprogramm zu dieser Funktion ist in Abbildung 172 zu sehen.

## 178 Dateigröße

Die Größe einer Datei wird vom .NET in Bytes zurückgeliefert. Ein Beispiel für eine solche Ermittlung ist in Listing 314 zu sehen. Oft benötigt man allerdings die Angabe der Dateigröße in einer anderen Einheit als Byte. Um die Umrechnung in eine Funktion auszulagern, die mehrfach angewandt werden kann, ist die Funktion `GetFileSize` aus Listing 323 bestimmt. Dieser Funktion wird neben dem Dateinamen (einschließlich Pfad) ein Enumerations-Wert für die Einheit über-



geben, in der die Dateigröße zurückgeliefert werden soll. Diese Enumeration ist in Listing 322 abgedruckt.

```
Public Enum FileSizeUnit As Integer
    B = 0
    kB
    MB
    GB
End Enum
```

Listing 322: Enumeration zur Festlegung der Dateigrößenangaben

Der optionale Parameter `SizeUnit` aus Listing 323 wird auf `Byte` gesetzt, um als Default-Wert die Angabe des .NET Frameworks zu reproduzieren.

```
Public Function GetFileSize(ByVal Filename As String, _
    Optional ByVal SizeUnit As FileSizeUnit = FileSizeUnit.B) As Long

    Dim FI As System.IO.FileInfo
    Dim Length As Long

    Try
        FI = New System.IO.FileInfo(Filename)
        ' Länge in Bytes
        Length = FI.Length
    Catch ex As System.IO.IOException
        Throw New ApplicationException _
            ("GetFileSize IO fehlgeschlagen", ex)
    Catch ex As ArgumentNullException
        Throw New ApplicationException _
            ("GetFileSize: " + Filename + " ist Nothing.", ex)
    Catch ex As ArgumentException
        Throw New ApplicationException _
            ("GetFileSize: " + Filename + " ist leer.", ex)
    End Try

    Select Case SizeUnit
        Case FileSizeUnit.kB
            Length /= 1024
        Case FileSizeUnit.MB
            Length /= (1024 * 1024)
        Case FileSizeUnit.GB
            Length /= (1024 * 1024 * 1024)
    End Select

    Return Length
End Function
```

Listing 323: Ermittlung der Dateigröße

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Propor-  
tyGrid

**Datei-  
system**

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

Mit dem Dateinamen wird über die `FileInfo`-Klasse das Objekt `FI` erzeugt, über das die Dateigröße mit der Eigenschaft `Length` abgefragt werden kann. Sollte dies ohne Ausnahmen zu generieren möglich sein, wird entsprechend der gewünschten Einheit die Dateigröße berechnet und dem aufrufenden Programm zurückgeliefert.

## 179 Dateivergleich

Dateien zu vergleichen kann ein beliebig umfangreiches und beliebig kompliziertes Unterfangen werden. Es hängt stark von den gewünschten Anforderungen an ein solches Programm oder einer solchen Funktion ab. Als Beispiel sei auf das Programm *WinDiff.exe* aus dem PLATFORM SDK verwiesen. Nun kann man sich aber auch eine Funktion programmieren, die man in eigene Programme einbinden kann. Hat man ein Netzwerk von mehreren hunderttausend Dateien, ist es mit Vergleichen von Dateiname, Dateigröße und unterschiedlichen Datumsangaben von Dateien nicht getan. Was zählt, ist der Inhalt. Eine Möglichkeit besteht nun darin, die Dateien Byte für Byte zu vergleichen. An dieser Stelle ist auch zu entscheiden, ob Leerzeichen, Tabulatoren für den Vergleich eine Rolle spielen, und wenn ja, an welcher Stelle. Spielt die Groß-/ Kleinschreibung bei Textdateien eine Rolle? Nimmt man auf solche »Feinheiten« keine Rücksicht, gibt es eine recht schnelle Methode, zwei Dateien miteinander zu vergleichen. Die Umsetzung ist in Listing 324 zu sehen.

Der Funktion werden die beiden zu vergleichenden Dateien als Zeichenkette übergeben. Nach der Deklaration einiger Hilfsvariablen werden die beiden Dateien als `FileStream` geöffnet, indem für jede Datei jeweils ein Objekt vom Typ `FileStream` erzeugt wird.

---

```
Public Function CompareFile(ByVal SrcFile As String, _
    ByVal CompFile As String) As Boolean

    Dim Res As Boolean
    Dim SrcLength As Long
    Dim CompLength As Long
    Dim FSSrc As System.IO.FileStream
    Dim FSComp As System.IO.FileStream
    Dim Src64 As String
    Dim Comp64 As String
    Dim SrcHash As Integer
    Dim CompHash As Integer

    Try
        ' FileStream der Ausgangsdatei
        FSSrc = New System.IO.FileStream(SrcFile, IO.FileMode.Open, _
            IO.FileAccess.Read)

        ' FileStream der zu testenden Datei
        FSComp = New System.IO.FileStream(CompFile, IO.FileMode.Open, _
            IO.FileAccess.Read)

        ' Dateigrößenberechnung beider Dateien
        SrcLength = GetFileSize(SrcFile)
        CompLength = GetFileSize(CompFile)
```

---

Listing 324: Vergleich zweier Dateien

```

' Die Arrays mit der richtigen Größe erstellen
Dim ByteSrc(SrcLength) As Byte
Dim ByteComp(CompLength) As Byte

' Gesamte Datei einlesen
FSSrc.Read(ByteSrc, 0, SrcLength)
FSComp.Read(ByteComp, 0, CompLength)

' Wird nicht mehr benötigt, schließen
FSSrc.Close()
FSComp.Close()

' Nach Base64 konvertieren
Src64 = Convert.ToBase64String(ByteSrc)
Comp64 = Convert.ToBase64String(ByteComp)

' Hash berechnen
SrcHash = Src64.GetHashCode
CompHash = Comp64.GetHashCode

If SrcHash.Equals(CompHash) Then
    Res = True
Else
    Res = False
End If

Catch ex As ArgumentNullException
    Throw New ApplicationException _
        ("CompareFile: Dateiname leer", ex)
Catch ex As ArgumentException
    Throw New ApplicationException _
        ("CompareFile: Dateiname ist Nothing", ex)
Catch ex As System.IO.FileNotFoundException
    Throw New ApplicationException _
        ("CompareFile: Dateiname nicht gefunden", ex)
Catch ex As System.IO.IOException
    Throw New ApplicationException _
        ("CompareFile: IO-Fehler", ex)
Catch ex As Exception
    Throw New ApplicationException("CompareFile fehlgeschlagen", ex)
End Try

Return Res
End Function

```

Listing 324: Vergleich zweier Dateien (Forts.)

Die Dateigröße wird mit der Funktion `GetFileSize` aus Listing 323 ermittelt. Diese Dateigröße wird genutzt, um die Dateien über die `Read`-Methode komplett einzulesen. Nachdem die Objekte mit dem Inhalt der jeweiligen Datei gefüllt sind, können die `FileStream`-Objekte geschlossen werden. Sie werden in der Folge der Funktion nicht mehr benötigt. Um alle Dateiformen vergleichen zu können, wird der Inhalt der Dateien in das Base64-Format konvertiert. Diese Konvertierung ist eindeutig und wandelt auch binäre Dateien in eine als Zeichenkette darstellbare Form um.

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Proper-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

Diese Base64-Zeichenketten werden in Variablen des Typs `String` abgespeichert. Die `String`-Klasse enthält eine Methode `GetHashCode`, die für die Zeichenkette einen eindeutigen Hash-Code erzeugt. Vergleicht man also den Hash-Code zweier Zeichenketten, hat man eine vor allem bei langen Zeichenketten schnelle Methode des Vergleiches. Da in den Zeichenketten der Dateiinhalt als Zeichenkette abgespeichert wurde, kann somit der Dateiinhalt verglichen werden. Diese Vorgehensweise ist schneller, als die Dateien Byte für Byte auf Gleichheit zu testen.

Nachteil dieser Methode ist, dass beide Dateien in den Hauptspeicher geladen werden müssen. Bei Dateien im Gigabyte-Bereich dürfte dies zurzeit in den meisten Fällen zu Schwierigkeiten führen. Für diese Fälle kann man sich aber sehr schnell eine Funktion selber schreiben, die diese Dateien Byte für Byte durchgeht.

## 180 Temporäre Dateinamen

Für jeden Benutzer eines Rechners wird ein Verzeichnis angelegt, in welchem die benutzerspezifischen Einstellungen abgespeichert werden. Hierzu zählt auch das Verzeichnis für temporäre Dateien. Eine kleine Wrapper-Funktion um die `Shared`-Methode `GetTempPath` ist in Listing 325 dargestellt. Auf allen getesteten Rechnern lieferte die Funktion den Pfad in der 8.3-Darstellung zurück. Der gleiche Effekt tritt auf, wenn man die Umgebungsvariable `%TEMP%` abfragt.

---

```
Public Function GetTempDir() As String
```

```
    Return System.IO.Path.GetTempPath
```

```
End Function
```

---

*Listing 325: Verzeichnis für temporäre Dateien*

Um den Namen einer temporären Datei zu bekommen, kann die Methode `GetTempFileName` der Klasse `System.IO.Path` benutzt werden. Einen entsprechenden Aufruf enthält Listing 326. Das »Problem« der Methode `GetTempFileName` ist, dass diese Methode einen gültigen Dateinamen ausprobiert, indem sie versucht, Dateien mit der Größe Null Bytes im temporären Verzeichnis zu erstellen. Gelingt dies, ist der Dateiname gültig und wird zurückgeliefert. Die erzeugte Datei der Größe Null existiert damit, ist aber nicht für die weitere Verarbeitung geöffnet. Will man in diese Datei schreiben, darf man dies keinesfalls mit einer Erstellung der Datei verknüpfen. Zudem hat sich in der Praxis gezeigt, dass die Suche wohl nicht immer richtig gelingt. Es werden jedenfalls im temporären Verzeichnis im Laufe der Zeit viele Dateien der Länge Null angesammelt.

Einige Programmierer benutzen diese Funktion auch zur Erzeugung eindeutiger Namen, ohne je die Absicht zu haben, in die erstellte Datei zu schreiben. In solchen Fällen kann die Funktion `GetTempFile` aus Listing 326 benutzt werden. Dieser Funktion wird mit einem logischen Parameter mitgeteilt, ob die erzeugte temporäre Datei nach der Erzeugung direkt wieder gelöscht werden soll oder nicht.

---

```
Public Function GetTempFile(ByVal delete As Boolean) As String
```

```
    Dim FileName As String
```

```
    Dim FI As System.IO.FileInfo
```

---

*Listing 326: Temporärer Dateiname mit .NET*

---

```

' Temporärer Dateiname aus dem System
FileName = System.IO.Path.GetTempFileName

If delete = True Then
    Try
        ' Löschen über FileInfo-Objekt
        FI = New System.IO.FileInfo(FileName)
        FI.Delete()
    Catch ex As System.IO.IOException
        Throw New ApplicationException _
            ("GetTempFileName: Löschen fehlgeschlagen", ex)
    End Try
End If

Return FileName
End Function

```

---

Listing 326: Temporärer Dateiname mit .NET (Forts.)

Eine andere Möglichkeit besteht darin, sich per Funktion selber einen eindeutigen Dateinamen zu generieren. Der in Listing 327 vorgestellte Weg hat sogar den Vorteil, dass die so erzeugte Datei mit anderen Rechnern austauschbar wäre, da der Name auch dann eindeutig ist. Das Hilfsmittel hierfür liegt in den Global Unique Identifiers (GUIDs). Im Namensraum `System` existiert eine entsprechende Klasse `GUID`, mit der eine solche eindeutige Kennung erzeugt werden kann.

---

```

Public Function GetTempFileNameGUID() As String
    Dim mGUID As Guid
    Dim mGUIDString As String

    ' Neue GUID vom Typ GUID erstellen
    mGUID = Guid.NewGuid
    mGUIDString = mGUID.ToString

    ' GUID-Zeichenketten haben Trennstriche
    ' diese werden hier entfernt
    mGUIDString = mGUIDString.Replace("-", "")

    ' Dateinamen-Erweiterung an Dateinamen anhängen
    mGUIDString += ".tmp"

    Return mGUIDString
End Function

```

---

Listing 327: eindeutiger temporärer Dateiname

In Listing 327 wird mit der `Shared`-Methode `NewGuid` dieser Klasse ein Objekt genau dieser Klasse erzeugt. Mit der bekannten Methode `ToString` kann diese GUID in eine Zeichenkette umgewandelt werden. Diese Zeichenkette hat noch den Nachteil – aber das ist sicherlich eine Geschmacksfrage –, Bindestriche zu enthalten, da eine GUID nach Bildungsgruppen getrennt in der Zeichenkette gespeichert werden soll(?). Diese Bindestriche werden mit der `String`-Methode `Replace` durch nichts ersetzt. Der so generierten Zeichenkette wird die Endung `«.tmp»` angehängt und der so gebildete Dateiname dem aufrufenden Programm übermittelt.

---

Basics

---

Datum/  
Zeit

---

Anwen-  
dungen

---

Zeich-  
nen

---

Bild-  
bearbei-  
tung

---

Win-  
dows  
Forms

---

Controls

---

Propor-  
tyGrid

---

Datei-  
system

---

Netz-  
werk

---

System

---

Daten-  
banken

---

XML

---

Wissen-  
schaft

---

Ver-  
schiede-  
nes

---

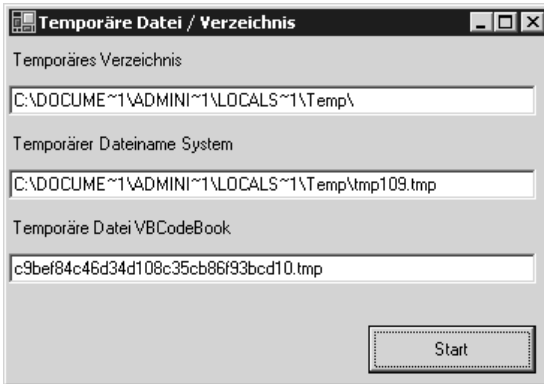


Abbildung 173: Temporärer Dateinamen und Verzeichnis

Beispiel für die oben geschilderten Funktionen sind in Abbildung 173 zu sehen.

## 181 Datei in mehreren Verzeichnissen suchen am Beispiel der Verzeichnisse von PATH

Kennt man die Verzeichnisse, in denen eine Datei zu finden sein sollte, die man sucht, ist eine Funktion hilfreich, die eine Liste von Verzeichnissen durchsuchen kann, die man dieser Funktion übergibt. Eine mögliche Realisierung solch einer Funktion ist in Listing 329 abgedruckt.

Ein konkretes Beispiel für eine solche Verzeichnisliste stellt der Inhalt der Umgebungsvariablen PATH dar. In dieser Variablen sind mehrere Verzeichnisse als semikolongetrennte Liste enthalten. Nun bietet sich eine solche Liste nicht unbedingt zum direkten Durchsuchen an. Die Funktion `GetEnvPaths` aus Listing 328 liefert die einzelnen Pfade als Zeichenketten-Array zurück. Über die Methode `GetEnvironmentVariable` der Klasse `Environment` aus dem Namensraum `System` werden die Verzeichnisse aus der Umgebung des Benutzers geholt. Die `Split`-Methode der `String`-Klasse überführen die Zeichenkette in das `String-Array` `Paths`.

---

```
Public Function GetEnvPaths() As String()

    Dim EnvPath As String
    Dim Paths() As String

    ' Holen der Umgebungsvariablen PATH
    EnvPath = System.Environment.GetEnvironmentVariable("PATH")

    ' Aufteilen der Zeichenkette am <;> In ZK-Array ablegen
    Paths = EnvPath.Split(CType(";", Char))

    Return Paths
End Function
```

---

Listing 328: PATH-Variable in String-Array überführen

Die so generierte Liste, aber natürlich auch jede andere Verzeichnisliste in Form eines String-Arrays, kann der Funktion `IsFileInPath` neben dem zu suchenden Dateinamen übergeben werden. Die Funktion überprüft als Erstes, ob der Dateiname und die Verzeichnisliste überhaupt einen Inhalt haben.

```
Public Function IsFileInPath(ByVal FileName As String, _
    ByVal Paths() As String) As String

    Dim FullFilePath As String
    Dim i As Integer

    If FileName Is Nothing Then
        Throw New ApplicationException(FileName + " ist Nothing")
    End If

    If FileName.Equals(String.Empty) Then
        Throw New ApplicationException("FileName ist leer")
    End If

    If Paths.Length = 0 Then
        Throw New ApplicationException("Verzeichnisarray ist leer")
    End If

    For i = 0 To Paths.Length - 1
        ' Kompletten Dateinamen erzeugen
        FullFilePath = Paths(i) + "\" + FileName
        If System.IO.File.Exists(FullFilePath) = True Then
            Exit For
        End If
        FullFilePath = ""
    Next

    Return FullFilePath
End Function
```

Listing 329: Datei in einem Verzeichnis suchen

Sind die Übergabeparameter nicht leer, wird in einer Schleife über alle Einträge der Liste der Dateiname an den jeweils aktuellen Pfad angehängt und mit der `Exists`-Methode der `File`-Klasse auf das Vorhandensein dieser Datei getestet. Liefert diese Funktion `True` zurück, kann die Schleife vorzeitig mit `Exit For` verlassen werden. In der Schleife wird die Variable `FullFilePath` am Ende eines Durchlaufs geleert. Damit kann das aufrufende Programm unterscheiden, ob die Datei gefunden wurde (Variable hat einen Inhalt) oder nicht (Variable ist leer). Am Beispiel der Abbildung 174 kann man sehen, dass die gesuchte Datei *feder.bmp* im Verzeichnis `C:\WINNT` gefunden wurde. Alle Verzeichnisse der Umgebungsvariablen `PATH` sind in der darunter angesiedelten `ListBox` zu erkennen.

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

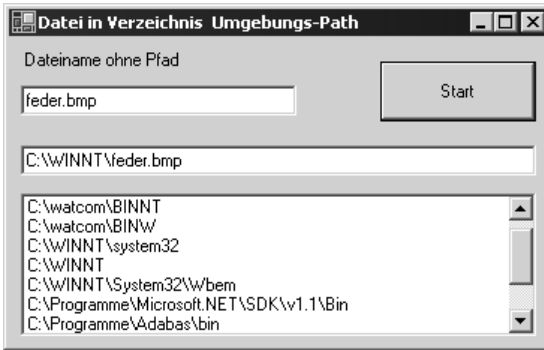


Abbildung 174: Dateisuche in einer Verzeichnisliste

## 182 Dateiinformatoren mit File System Object

Im .NET FRAMEWORK existiert die Klasse `FileInfo`, über die man sehr viele Informationen über eine Datei ermitteln kann. Einige Anwendungen dieser Klasse können in den vorhergehenden Kapiteln eingesehen werden. Neben der Anwendung dieser Klasse kann man aus Kompatibilitätsgründen, oder um sanft nach .NET zu konvertieren, die Scripting Runtime einbinden. Dies wurde unter anderem in Listing 317 angewandt.

Welche Informationen zusätzlich noch mit den File System Objects (FSO) der Scripting Runtime erfragt werden können, ist in Listing 331 zu sehen. Die ermittelten Informationen werden mit Hilfe einer Strukturvariablen (Listing 330) dem aufrufenden Programm zurückgegeben.

---

```
Public Structure FSFileInfoStruc
    Dim Path As String
    Dim LastAccess As Date
    Dim LastModified As Date
    Dim Created As Date
    Dim ShortName As String
    Dim Size As Long
End Structure
```

---

Listing 330: Strukturvariable für Dateiinformatoren

Die Funktion `GetFSFileInfo` aus Listing 331 liefert alle Informationen zurück, die mit Hilfe der FSO ermittelt werden können. Vergleicht man dies mit den Informationen, die `FileInfo` zur Verfügung stellt, fällt die Wahl bei einer Neuprogrammierung nicht schwer.

---

```
Public Function GetFSFileInfo(ByVal Name As String) As _
    FSFileInfoStruc

    Dim fso As FileSystemObject = New FileSystemObject
    Dim FSFile As File
    Dim FSInfo As FSFileInfoStruc

    FSFile = fso.GetFile(Name)
```

---

Listing 331: Dateiinformatoren mit FSO



```
' Pfadangabe per FSO
FSOInfo.Path = FSOFile.Path

' Zeitpunkt des letzten Zugriffs
FSOInfo.LastAccess = FSOFile.DateLastAccessed

' Zeitpunkt der letzten Änderung
FSOInfo.LastModified = FSOFile.DateLastModified

Zeitpunkt der Erstellung
FSOInfo.Created = FSOFile.DateCreated

' kurzer Dateiname
FSOInfo.ShortName = FSOFile.ShortName

' Dateigröße
FSOInfo.Size = CType(FSOFile.Size, Long)

Return FSOInfo
End Function
```

Listing 331: Dateiinformationen mit FSO (Forts.)

Damit die Funktion in der dargestellten Weise funktioniert, muss die Scripting Runtime in das Projekt eingebunden werden. Wie dies geschieht, kann Abbildung 175 und dem dazugehörigen Text entnommen werden. Zusätzlich muss durch die Anweisung `Imports Scripting` der entsprechende Namensraum der Funktion bekannt gemacht werden.



Abbildung 175: Dateiinformationen mittels FSO

Zu Beginn des Programms wird ein Objekt der Klasse `FileSystemObject` erstellt. Dieses Objekt dient als Ausgangsbasis für das Objekt `FSOFile`, welches mit dem Namen der zu untersuchenden Datei initialisiert wird. Anschließend werden die einzelnen Dateiinformationen der Struktur übergeben, die schlussendlich dem aufrufenden Programm zurückgegeben wird.

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Propert-  
yGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

## 183 Laufwerksinformationen mit FSO

Informationen über Dateisysteme/Laufwerke kann man nicht nur über WMI (siehe Rezepte 11.14 ff. ((Logische Laufwerke mit WMI)) in Erfahrung bringen. Begnügt man sich mit etwas weniger Informationen, so geht dies auch über die File System Objects der Scripting Runtime Engine.

Um die Eigenschaften eines Laufwerkes für verschiedene Gerätetypen ermitteln zu können, kann eine Enumeration der verschiedenen Gerätetypen benutzt werden. Die in diesem Kapitel benutzte Enumeration ist in Listing 332 zu sehen.

---

```
Public Enum DriveType
    Unknown = 0
    Removable
    Fixed
    Network
    CDROM
    RAMDisk
End Enum
```

---

*Listing 332: Enumeration der Datenträgertypen*

Die für jedes Laufwerk zurückgegebenen Informationen werden in einer Strukturvariablen gesammelt. Die Typdefinition für diese Strukturvariable ist in Listing 333 abgedruckt. Sie enthält den Namen des Laufwerkes, die Gesamtkapazität in Gigabyte, die noch freie Kapazität in Gigabyte, den Laufwerkstyp und die Seriennummer des Laufwerkes.

---

```
Public Structure DriveStruc
    Dim Name As String
    Dim TotalGB As Integer
    Dim FreeGB As Integer
    Dim Type As DriveType
    Dim SN As String
End Structure
```

---

*Listing 333: Strukturvariable für die Rückgabe der Ergebnisse*

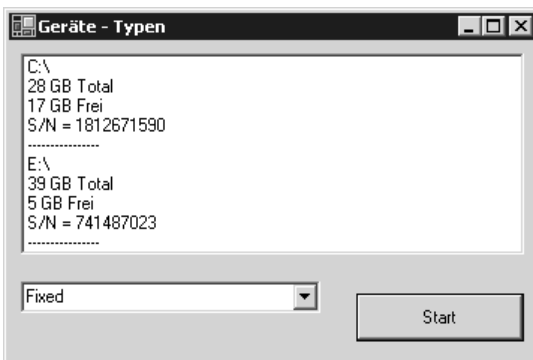


*Abbildung 176: Dateisystemeigenschaften von Festplatten eines Testrechners*

Mit diesen Vorarbeiten kann die Funktion `GetDriveTypeInfo` aus Listing 334 realisiert werden, die die Ergebnisse für die angeforderten Laufwerkstypen als Hash-Tabelle zurückliefert.

Mit der Methode `GetLogicalDrives` der Klasse `Environment` werden alle logischen Laufwerke des Rechners ermittelt und deren Bezeichnung in dem String-Array `Drives` abgespeichert.

```
Public Function GetDriveTypeInfo(ByVal DrvType As DriveType) _
    As Hashtable

    Dim Table As Hashtable = New Hashtable
    Dim Drives() As String
    Dim Info As DriveStruc
    Dim i As Integer
    Dim fso As FileSystemObject = New FileSystemObject
    Dim fsoType As DriveType
    Dim fsoDrive As Drive

    ' Alle Laufwerke ermitteln, die dem System bekannt sind
    Drives = System.Environment.GetLogicalDrives

    ' Schleife über alle bekannten Laufwerke
    For i = 0 To Drives.GetUpperBound(0) - 1
        fsoDrive = fso.GetDrive(Drives(i))

        ' Bei Wechselplattenmedien / Disks notwendig
        If fsoDrive.IsReady = True Then
            fsoType = fsoDrive.DriveType
            If fsoType = DrvType Then
                Info.Name = Drives(i)
                Info.TotalGB = fsoDrive.TotalSize / (1024 * 1024 * 1024)
                Info.FreeGB = fsoDrive.FreeSpace / (1024 * 1024 * 1024)
                Info.SN = fsoDrive.SerialNumber.ToString

                Table.Add(Drives(i), Info)
            End If
        End If
    Next
    Return Table
End Function
```

Listing 334: Funktion zur Ermittlung von Laufwerkseigenschaften

In einer Schleife über alle logischen Laufwerke wird dann für das jeweils aktuelle Laufwerk ein Objekt der Klasse `Drive` erstellt. Für dieses Objekt muss festgestellt werden, ob das dazugehörige Laufwerk bereit ist. Ein Diskettenlaufwerk ohne eingelegte Diskette ist beispielsweise nicht bereit und würde zu einer Ausnahme führen, wenn Informationen über ein solches Laufwerk abgefragt würden. Da nur Angaben über Laufwerke eines bestimmten Typs gewünscht sind, wird die Abfrage mittels einer weiteren `if`-Abfrage nur für diese Typen durchgeführt. Die so mit Werten gefüllte Strukturvariable wird als `Value` der Hash-Tabelle angehängt, wobei die Bezeichnung des logischen Laufwerks als `Key` benutzt wird.

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Proper-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

## 184 Delimited-Dateien nach XML transformieren

Zurzeit werden noch viele Daten als Delimited-Dateien aus Programmen exportiert oder zwischen Anwendern ausgetauscht. Ob die einzelnen Felder der Datensätze dann mit Komma, Semikolon oder einem »|« getrennt werden, spielt dabei eher eine untergeordnete Rolle. Wollen Sie die Daten als XML weiterverarbeiten, benötigen Sie eine entsprechende Konvertierungsfunktion. Eine mögliche Lösung ist in Listing 335 zu sehen. Dieser Funktion werden fünf Parameter übergeben. Die ersten beiden Parameter beziehen sich auf die Quelldatei, die letzten drei auf die zu erstellende XML-Datei. XML wird in Kapitel 13 ((XML)) behandelt, daher wird an dieser Stelle nicht näher darauf eingegangen.

---

```
Public Function Del2XML(ByVal strDel As String, _
    ByVal FileName As String, _
    ByVal XMLDatasetName As String, _
    ByVal XMLNamespace As String, _
    ByVal XMLTableName As String) As String

    Dim TextFileSR As IO.StreamReader
    Dim mDS As DataSet = New DataSet
    Dim mTable As DataTable = New DataTable
    Dim mRow As DataRow
    Dim mField As String
    Dim mKopf As String = "<?xml version=""1.0""?>"
    Dim i As Integer = 0

    Try
        ' Datei wird mit StreamReader eingelesen
        TextFileSR = New IO.StreamReader(FileName)
    Catch ex As ArgumentException
        Throw New ApplicationException _
            ("Del2XML: Dateiname ist leer", ex)
    Catch ex As ArgumentNullException
        Throw New ApplicationException _
            ("Del2XML: Dateiname ist Nothing", ex)
    Catch ex As System.IO.FileNotFoundException
        Throw New ApplicationException _
            ("Del2XML: Dateiname konnte nicht gefunden werden", ex)
    Catch ex As System.IO.IOException
        Throw New ApplicationException _
            ("Del2XML: allgemeine IO-Ausnahme", ex)
    Catch ex As Exception
        Throw New ApplicationException _
            ("Del2XML: allgemeine Ausnahme", ex)
    End Try

    ' Dataset mit Inhalt versorgen
    mDS.DataSetName = XMLDatasetName
    mDS.Namespace = XMLNamespace
    mDS.Tables.Add(XMLTableName)

    ' Tabellenfelder aufbauen
```

---

Listing 335: Umwandlung von Delimited-Dateien in XML-Dateien

```

TextFileSR.BaseStream.Seek(0, IO.SeekOrigin.Begin)
For Each mField In TextFileSR.ReadLine.Split(strDel)
    mDS.Tables(0).Columns.Add(mField)
Next

' Alle Zeilen der Datei einlesen
While (TextFileSR.Peek() > -1)
    mRow = mDS.Tables(0).NewRow
    For Each mField In TextFileSR.ReadLine.Split(strDel)

        ' Felder des Dataset mit Werten aus Dateizeile füllen
        mRow(i) = mField
        i += 1
    Next
    i = 0

    ' Zeile der Tabelle hinzufügen
    mDS.Tables(0).Rows.Add(mRow)
End While

TextFileSR.Close()

' XML-Zeichenkette des Dataset erzeugen
mField = mDS.GetXml

' fehlenden Kopf hinzufügen, siehe Variablendefinition
mField = mKopf + ControlChars.CrLf _
    + mField

Return mField
End Function

```

Listing 335: Umwandlung von Delimited-Dateien in XML-Dateien (Forts.)

Mit dem übergebenen Dateinamen wird ein `StreamReader`-Objekt erzeugt. Sollten dabei Ausnahmen auftreten, so werden diese einheitlich als `ApplicationException` an das aufrufende Programm gemeldet.

Um die XML-Datei zu erstellen, wird als Hilfe das `Dataset`-Objekt `mDS` erstellt. Diesem Objekt werden der übergebene `Dataset`-Name, der `Dataset`-Namensraum und der Tabellename übergeben. Damit die Konvertierungsfunktion die Feldnamen kennt, müssen in der ersten Zeile der Delimited-Datei die Feldnamen in der gleichen Reihenfolge wie die anschließenden Daten und ebenfalls mit Trennzeichen getrennt abgespeichert sein.

Mit dieser ersten Zeile werden nun die Felder der Tabelle erstellt. Anschließend wird für jede Zeile ein neuer Datensatz angelegt und mit den entsprechenden Daten der Delimited-Datei gefüllt. Sind alle Datensätze eingelesen, kann mit der `GetXml`-Methode das Dataset in einer Zeichenkette abgelegt werden. Diese Zeichenkette entspricht noch nicht einer wohlgeformten XML-Datei. Um diese Bedingung zu erfüllen muss noch eine Kopfzeile vorangestellt werden. Als Beispiel kann die Datei aus Listing 337 genommen werden. Die umgewandelte Datei im XML-Format ist in Listing 338 abgedruckt.

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Propert-  
yGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

---

```

Private Sub btnTransform_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnTransform.Click

    ' OpenFileDialog-Objekt erzeugen
    Dim ofd As OpenFileDialog = New OpenFileDialog

    Dim XMLString As String
    Dim FileName As String
    Dim TextFileSR As IO.StreamReader

    ' OpenFileDialog-Eigenschaften festlegen
    ofd.Filter = "Text Files|*.txt"
    ofd.Title = "Auswahl für Datei mit Trennzeichen"

    ' OpenFileDialog anzeigen und Dateinamen übernehmen
    If ofd.ShowDialog = DialogResult.OK Then
        FileName = ofd.FileName
    End If

    TextFileSR = New IO.StreamReader(FileName)
    txtDelFile.Text = TextFileSR.ReadToEnd
    TextFileSR.Close()

    XMLString = Del2XML(txtDelimiter.Text, FileName, _
        "Patient", "hospital.vbcodebook.com", _
        "PatientAddress")
    txtXMLFile.Text = XMLString
End Sub

```

---

*Listing 336: Aufrufbeispiel der Funktion zur Umwandlung Delimited – XML*

Um die Umwandlung an Hand des Beispiels besser verfolgen zu können, ist in Listing 336 das Programm abgedruckt, welches die Datei aus Listing 337 der Funktion `Del2XML` übergeben hat.

---

```

PatientID;Name;ChrName;Birthday;Street;Zip;City
2300003;Mustermann;Karl;01.01.1978;Beispielstr. 8;22001;Hamburg
2300010;Meier - Mayer;Ute;21.08.1948;Schillerweg 23;46486 Wesel

```

---

*Listing 337: Delimited-Datei*

---

```

<?xml version="1.0"?>
<Patient xmlns="hospital.vbcodebook.com">
  <PatientAddress>
    <PatientID>2300003</PatientID>
    <Name>Mustermann</Name>
    <ChrName>Karl</ChrName>
    <Birthday>01.01.1978</Birthday>
    <Street>Beispielstr. 8</Street>
    <Zip>22001</Zip>
    <City>Hamburg</City>
  </PatientAddress>
</Patient>

```

---

*Listing 338: Nach XML umgewandelte Datei*

---

```

</PatientAddress>
<PatientAddress>
  <PatientID>2300010</PatientID>
  <Name>Meier - Mayer</Name>
  <ChrName>Ute</ChrName>
  <Birthday>21.08.1948</Birthday>
  <Street>Schillerweg 23</Street>
  <Zip>46486 Wesel</Zip>
</PatientAddress>
</Patient>

```

---

*Listing 338: Nach XML umgewandelte Datei (Forts.)*

## 185 Überwachung des Dateisystems

Das .NET Framework bietet mit seinen vielen Klassen Möglichkeiten, die vorher nur mit C/C++ oder Systemaufrufen möglich waren. Manche Dinge gingen unter VISUAL BASIC überhaupt nicht. Eine sehr praktische Möglichkeit ist die Überwachung des Dateisystems. Es ist in heutigen heterogenen Netzwerk- und Rechnerstrukturen durchaus noch üblich, dass Daten zwischen verschiedenen Systemen durch ASCII-Dateien ausgetauscht werden. Üblich ist dann das Lesen dieser Dateien zu einem fest vorgegebenen Zeitpunkt. Könnte man diese Datei lesen, wenn sie gerade erstellt oder verändert wurde, könnten die Daten schneller zwischen den Systemen ausgetauscht werden.

Wie man eine solche Überwachung für ein Verzeichnis aufbaut, ist in Listing 339 zu sehen. Es handelt sich um ein komplettes Modul einer Konsolen-Anwendung. Die grundlegende Erstellung eines Überwachungs-Objektes kann in einer Zeile abgehandelt werden. Im System.IO-Namensraum existiert exakt für diese Aufgabe die Klasse `FileSystemWatcher`. In Listing 339 wird das Objekt `fsw` durch Instanzierung erstellt. Dieses Objekt kann allerdings in diesem Zustand noch nicht sehr viel. Es verbraucht nur Speicherplatz. Die Funktionalität muss über Event-Handler zur Verfügung gestellt werden. Zusätzlich muss das Objekt natürlich wissen, was es überwachen soll.

In Listing 339 wird zuerst nach dem zu überwachenden Verzeichnis gefragt und der Variablen `Path2Watch` zugeordnet. Anschließend wird festgelegt, ob vorhandene Unterverzeichnisse ebenfalls mit überwacht werden sollen. Diese Rahmenbedingungen werden dem `FileSystemWatcher`-Objekt übergeben.

---

```

Imports System.IO

Module FileSystemWatcherProgram
  Private Path2Watch As String
  Private WithSubDirectories As Boolean
  Private fsw As FileSystemWatcher
  Private mEvt As WaitForChangedResult

  Sub Main()
    fsw = New FileSystemWatcher

    Console.WriteLine("Verzeichnis : ")
    Path2Watch = Console.ReadLine()

```

---

*Listing 339: Programm zur Überwachung des Dateisystems*

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Propert-  
yGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

```

Console.WriteLine("Unterverzeichnisse einschliessen (True/False): ")
WithSubDirectories = CType(Console.ReadLine(), Boolean)

fsw.Path = Path2Watch
fsw.IncludeSubdirectories = WithSubDirectories

AddHandler fsw.Created, _
    New FileSystemEventHandler(AddressOf IsCreated)

AddHandler fsw.Changed, _
    New FileSystemEventHandler(AddressOf IsChanged)

AddHandler fsw.Deleted, _
    New FileSystemEventHandler(AddressOf IsDeleted)

AddHandler fsw.Renamed, _
    New RenamedEventHandler(AddressOf IsRenamed)

Console.WriteLine("Überwachung wird gestartet!")
Console.WriteLine()

While True
    mEvt = fsw.WaitForChanged(WatcherChangeTypes.All)
End While

End Sub

Private Sub IsCreated(ByVal source As Object, _
    ByVal evt As FileSystemEventArgs)
    If WithSubDirectories Then
        Console.WriteLine(evt.FullPath + " wurde erstellt")
    Else
        Console.WriteLine(evt.Name + " wurde erstellt")
    End If
End Sub

Private Sub IsChanged(ByVal source As Object, _
    ByVal evt As FileSystemEventArgs)
    If WithSubDirectories Then
        Console.WriteLine(evt.FullPath + " wurde verändert")
    Else
        Console.WriteLine(evt.Name + " wurde verändert")
    End If
End Sub

Private Sub IsDeleted(ByVal source As Object, _
    ByVal evt As FileSystemEventArgs)
    If WithSubDirectories Then
        Console.WriteLine(evt.FullPath + " wurde gelöscht")
    Else
        Console.WriteLine(evt.Name + " wurde gelöscht")
    End If
End Sub

```

---



```

End If
End Sub

Private Sub IsRenamed(ByVal source As Object, _
    ByVal evt As RenamedEventArgs)
    If WithSubDirectories Then
        Console.WriteLine(evt.OldFullPath + " wurde In " + _
            evt.FullPath + " geändert")
    Else
        Console.WriteLine(evt.OldName + " wurde In " + _
            evt.Name + " geändert")
    End If
End Sub

End Module

```

Listing 339: Programm zur Überwachung des Dateisystems (Forts.)

Für jede Veränderungsart, die überwacht werden soll, muss ein entsprechender Event-Handler programmiert und dem Objekt mitgeteilt werden. Hierzu zählen das Erstellen einer Datei oder eines Verzeichnisses (Created), das Ändern (Changed), das Umbenennen (Renamed) und das Löschen (Deleted).

Die für diese Ereignisse erstellten Funktionen müssen einer definierten Aufrufsignatur folgen, wobei diese für fast alle Ereignisse identisch ist. Nur die Signatur für Renamed folgt einem anderen Schema. Die Namen der Funktionen spielen dabei keine Rolle. Im Listing 339 wurden die Funktionen nach dem Ereignis benannt und die Vorsilbe Is vorangestellt. Nachdem die Funktionen mit der Methode AddHandler dem FileSystemWatcher-Objekt angefügt wurden, wird in einer Endlosschleife auf Veränderungen geprüft. Je nach erfolgter Veränderung wird dann die entsprechende Funktion aufgerufen.



Abbildung 177: Verzeichnisüberwachung

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

Ein Beispiel für eine solche Überwachung ist in Abbildung 177 zu sehen. Die ersten sechs Einträge sind durch das Erstellen einer leeren Visio-Datei entstanden! Hieran kann man erkennen, dass man die Eigenarten der Programme berücksichtigen muss, die diejenigen Dateien erstellen, die man überwachen möchte. Nur das Löschen der leeren Visio-Datei erzeugt ein Ereignis J. Auch das Erstellen eines Verzeichnisses mit dem Windows-Explorer produziert zwei Ereignisse. Zuerst wird ein neuer Ordner (»new folder« wegen des amerikanischen Windows) angelegt, der dann umbenannt wird. Dies entspricht auch der sichtbaren Vorgehensweise des Windows-Explorers. In der letzten Zeile erkennt man, dass für das Löschen eines Verzeichnisses einschließlich aller Unterverzeichnisse nur ein Ereignis ausgelöst wird. Und nicht, wie man vermuten könnte, für jedes gelöschte Verzeichnis und jede gelöschte Datei ein separates Ereignis.

## 186 Datei-Attribute

Jeder Datei im Dateisystem werden bestimmte Informationen über den Status der Datei mitgegeben. Diese Informationen können natürlich mit dem .NET Framework ausgelesen werden. Aber auch die File System Objects kennen ein Verfahren, diese Attribute einer Datei zu ermitteln. Hierbei zeigen sich Unterschiede im Ergebnis, die unter anderem an den Ergänzungen der Attribute bei neueren Betriebssystemen liegen. Aus diesem Grund werden hier beide Verfahren vorgestellt.

In Listing 340 ist die Ermittlung der Attribute mit FSO abgedruckt. Zur Einbindung der FSO siehe Kapitel 9.7 Nach der Erstellung eines Objektes der Klasse `FileSystemObject` wird das Objekt der Klasse `File` für die zu untersuchende Datei über dieses Objekt erstellt. Die Attribute dieser Datei können dann über die Eigenschaft `Attributes` dieses Datei-Objektes ermittelt werden. Anschließend wird jedes mögliche Attribut abgefragt und als Klartext einer Zeichenkette zugeordnet. In dieser Zeichenkette werden die Attribute als semikolongetrennte Liste an das aufrufende Programm zurückgeliefert.

---

```
Public Function GetFSOAttributes(ByVal Name As String) As String
    Dim RetAttribute As String
    Dim Fso As FileSystemObject = New FileSystemObject
    Dim FsoFile As File
    Dim FSOFileAttribute As FileAttribute

    FsoFile = Fso.GetFile(Name)
    FSOFileAttribute = FsoFile.Attributes

    If (FSOFileAttribute And FileAttribute.Alias) = _
        FileAttribute.Alias Then

        RetAttribute += "Alias"
    End If

    If (FSOFileAttribute And FileAttribute.Archive) = _
        FileAttribute.Archive Then

        RetAttribute += ";Archiv"
    End If

    If (FSOFileAttribute And FileAttribute.Compressed) = _
        FileAttribute.Compressed Then
```

---

Listing 340: Dateiattribute mit den File System Objects

```

        RetAttribute += ";Komprimiert"
    End If

    If (FSOFileAttribute And FileAttribute.Directory) = _
        FileAttribute.Directory Then

        RetAttribute += ";Verzeichnis"
    End If

    If (FSOFileAttribute And FileAttribute.Hidden) = _
        FileAttribute.Hidden Then

        RetAttribute += ";Versteckt"
    End If

    If (FSOFileAttribute And FileAttribute.Normal) = _
        FileAttribute.Normal Then

        RetAttribute += ";Normal"
    End If

    If (FSOFileAttribute And FileAttribute.ReadOnly) = _
        FileAttribute.ReadOnly Then

        RetAttribute += ";NurLesen"
    End If

    If (FSOFileAttribute And FileAttribute.System) = _
        FileAttribute.System Then

        RetAttribute += ";System"
    End If

    If (FSOFileAttribute And FileAttribute.Volume) = _
        FileAttribute.Volume Then

        RetAttribute += ";Volume"
    End If
    Return RetAttribute
End Function

```

Listing 340: Dateiattribute mit den File System Objects (Forts.)

Ermittelt man die Datei-Attribute direkt mit den Klassen des .NET Frameworks, kann die Klasse `FileInfo` herangezogen werden. Nach der Erstellung eines Objektes für diese Klasse mit dem Namen der zu untersuchenden Datei wird jedes mögliche Attribut ermittelt und mit Semikolon getrennt in einer Zeichenkette abgespeichert. Diese Zeichenkette wird dem aufrufenden Programm zurück gegeben. Gegebenenfalls kann dort die Zeichenkette mit der `Split`-Methode in ein Zeichenketten-Array aufgelöst werden.

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

**Datei-  
system**

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schie-  
denes

---

```

Public Function GetAttributes(ByVal FileName As String) _
    As String

    Dim FileNameFI As System.IO.FileInfo
    Dim FileNameAttrib As System.IO.FileAttributes
    Dim AttributeString As String

    Try
        FileNameFI = New System.IO.FileInfo(FileName)
    Catch ex As ArgumentNullException
        Throw New ApplicationException _
            ("GetAttributes: Dateiname ist Nothing", ex)
    Catch ex As ArgumentException
        Throw New ApplicationException _
            ("GetAttributes: Dateiname ist leer", ex)
    Catch ex As System.IO.FileNotFoundException
        Throw New ApplicationException _
            ("GetAttributes: Datei konnte nicht gefunden werden", ex)
    Catch ex As System.IO.IOException
        Throw New ApplicationException _
            ("GetAttributes: allg. IO-Ausnahme", ex)
    Catch ex As Exception
        Throw New ApplicationException _
            ("GetAttributes: allg. Ausnahme", ex)
    End Try

    If (FileNameFI.Attributes And FileNameAttrib.Archive) = _
        IO.FileAttributes.Archive Then
        AttributeString = "Archiv"
    End If

    If (FileNameFI.Attributes And FileNameAttrib.Compressed) = _
        IO.FileAttributes.Compressed Then
        AttributeString += ";Komprimiert"
    End If

    If (FileNameFI.Attributes And FileNameAttrib.Device) = _
        IO.FileAttributes.Device Then
        AttributeString += ";Gerät"
    End If

    If (FileNameFI.Attributes And FileNameAttrib.Directory) = _
        IO.FileAttributes.Directory Then
        AttributeString += ";Verzeichnis"
    End If

    If (FileNameFI.Attributes And FileNameAttrib.Encrypted) = _
        IO.FileAttributes.Encrypted Then
        AttributeString += ";Verschlüsselt"
    End If

```

---

```
If (FileNameFI.Attributes And FileNameAttrib.Hidden) = _
    IO.FileAttributes.Hidden Then
    AttributeString += ";Versteckt"
End If

If (FileNameFI.Attributes And FileNameAttrib.Normal) = _
    IO.FileAttributes.Normal Then
    AttributeString += ";Normal"
End If

If (FileNameFI.Attributes And FileNameAttrib.NotContentIndexed) = _
    IO.FileAttributes.NotContentIndexed Then
    AttributeString += ";nicht im FileIndex-Verfahren"
Else
    AttributeString += ";im FileIndex-Verfahren"
End If

If (FileNameFI.Attributes And FileNameAttrib.Offline) = _
    IO.FileAttributes.Offline Then
    AttributeString += ";Offline"
End If

If (FileNameFI.Attributes And FileNameAttrib.ReadOnly) = _
    IO.FileAttributes.ReadOnly Then
    AttributeString += ";NurLesen"
End If

If (FileNameFI.Attributes And FileNameAttrib.ReparsePoint) = _
    IO.FileAttributes.ReparsePoint Then
    AttributeString += ";Analysepunkt vorhanden"
End If

If (FileNameFI.Attributes And FileNameAttrib.SparseFile) = _
    IO.FileAttributes.SparseFile Then
    AttributeString += ";Dünn besetzt"
End If

If (FileNameFI.Attributes And FileNameAttrib.System) = _
    IO.FileAttributes.System Then
    AttributeString += ";System"
End If

If (FileNameFI.Attributes And FileNameAttrib.Temporary) = _
    IO.FileAttributes.Temporary Then
    AttributeString += ";Temporär"
End If

Return AttributeString
End Function
```

Listing 341: Dateiattribute mit .NET (Forts.)

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Propert-  
yGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

Die Unterschiede beider Verfahren kann man in Abbildung 178 sehen. Das FSO-Verfahren erkennt nicht, dass die Datei am FileIndex-Verfahren des Betriebssystems teilnimmt. Dieses Verfahren gab es zur Entstehungszeit der FSO noch nicht. Interessant ist, dass FSO das Attribut `Normal` feststellt, während das .NET-Verfahren nicht dieser Meinung ist.

Bei der Entwicklung neuer Programme sollte man also genau abwägen, ob es sich noch lohnt, alte Verfahren einzusetzen, oder doch den Mehraufwand des Lernens »in Kauf« nimmt.

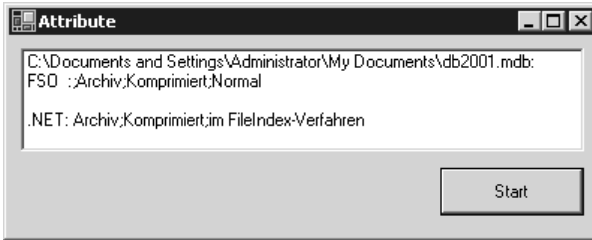


Abbildung 178: Dateiattribute mit FSO und .NET

## 187 Bestandteile eines Pfads ermitteln

Wenn Sie einen vorgegebenen Pfad in seine Bestandteile zerlegen müssen, dann finden Sie in der Klasse `Path` wertvolle Hilfestellungen. Eine Reihe statischer Methoden stellt die gewöhnlich benötigten Analysefunktionen bereit.

Mit Hilfe eines kleinen Testprogramms (Abbildung 179) können Sie schnell nachvollziehen, welche Ergebnisse die verschiedenen Methoden liefern. Listing 342 zeigt, wie die Informationen abgerufen werden. Z.B. gibt `Path.GetDirectoryName` den Verzeichnis-Teil eines Pfades zurück, während `Path.GetFileName` den Dateiname inklusive Erweiterung zurückgibt.

---

```
Private Sub TBPath_TextChanged(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles TBPath.TextChanged

    ' Der zu analysierende Pfad
    Dim p As String = TBPath.Text

    Try
        TBPath.ForeColor = Color.Black

        ' Pfad-Informationen ermitteln
        CBHasExtension.Checked = Path.HasExtension(p)
        CBIsAbsolute.Checked = Path.IsPathRooted(p)
        TBDirectory.Text = Path.GetDirectoryName(p)
        TBFilename1.Text = Path.GetFileName(p)
        TBFilename2.Text = Path.GetFileNameWithoutExtension(p)
        TBExtension.Text = Path.GetExtension(p)

    Catch ex As Exception
        ' Fehler abfangen, Text rot anzeigen
        TBPath.ForeColor = Color.Red
    End Try
End Sub
```

---

Listing 342: Ermitteln der Bestandteile eines in einer TextBox eingegebenen Pfades

---

End Try

End Sub

---

Listing 342: Ermitteln der Bestandteile eines in einer TextBox eingegebenen Pfades (Forts.)

The screenshot shows a Windows application window titled "Pfade". It contains the following elements:

- A text box labeled "Pfad" containing the string "C:\A\B\C\Text.txt".
- Two checkboxes: "Besitzt Dateierweiterung" (checked) and "Ist absolut" (checked).
- A text box labeled "Verzeichnis" containing the string "C:\A\B\C".
- A text box labeled "Dateiname" containing the string "Text.txt".
- A text box labeled "Dateiname ohne Erweiterung" containing the string "Text".
- A text box labeled "Erweiterung" containing the string ".txt".

Abbildung 179: Das kleine Testprogramm demonstriert, wie ein Pfad mit Hilfe der Klasse `Path` zerlegt werden kann

Beachten Sie jedoch bitte, dass die Methoden sich auf die String-Bearbeitung beschränken. Es wird nicht untersucht, ob die Verzeichnisse oder Dateien tatsächlich existieren. Ferner können die Methoden nicht unterscheiden, ob es sich bei dem Ausgangspfad um ein Verzeichnis oder eine Datei handelt. Bei der Analyse wird angenommen, dass der Text hinter dem letzten Backslash eine Dateiangabe ist.

## 188 Absolute und gekürzte (kanonische) Pfade ermitteln

Ein Pfad kann ohne Angabe eines Root-Knotens, also relativ definiert sein. Er bezieht sich dann auf das aktuelle Verzeichnis (`CurrentDirectory`). Für viele Dateioperationen ist es jedoch notwendig, einen absoluten Pfad anzugeben, der unabhängig vom aktuell eingestellten Verzeichnis ist.

Des Weiteren kann ein Pfad auch Verweise auf Elternknoten enthalten. Z.B. ist `X:\A\B\..\C` identisch mit `X:\A\C`. Solche Pfade entstehen oft bei der Zusammensetzung von Verzeichnissen und sollten möglichst in ihre kürzeste (kanonische) Form umgewandelt werden.

Während sich der absolute Pfad eines vorgegebenen Pfades leicht mit `Path.GetFullPath` herausfinden lässt, stellt das Framework zum Kürzen des Pfades bislang nichts zur Verfügung. Hierfür muss auf die API-Funktion `Path.Canonicalize` zurückgegriffen werden (Listing 343).

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Proper-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

---

Public Class API

```
'''
Public Declare Auto Function PathCanonicalize Lib "shlwapi.dll" _
    (ByVal dst As System.Text.StringBuilder, ByVal src As String) _
    As Boolean
```

End Class

Public Class ApiVBNet

```
'''
Public Shared Function GetCanonicalPath( _
    ByVal pathFrom As String) As String

    ' StringBuider-Instanz als Buffer für die API-Funktion
    Dim sb As New System.Text.StringBuilder(300)

    ' API-Methode aufrufen
    Dim b As Boolean = API.PathCanonicalize(sb, pathFrom)

    ' Bei Erfolg zusammengesetzten String zurückgeben
    If b Then Return sb.ToString()

    ' Ansonsten Leerstring zurückgeben
    Return ""
```

End Function

End Class

---

Listing 343: Wrapper-Funktion für den Aufruf der API-Methode PathCanonicalize

Auch hierfür steht Ihnen auf der Buch-CD ein kleines Testprogramm zur Verfügung, mit dessen Hilfe Sie das Verhalten der beiden Methoden nachvollziehen können (Abbildung 180). Der in der TextBox TBPath eingegebene Pfad wird umgewandelt und in zwei anderen TextBoxen dargestellt:

---

```
Dim p As String = TBPath.Text

' Pfad-Informationen ermitteln
TBAbsolutePath.Text = Path.GetFullPath(p)
TBCanonicalPath.Text = ApiVBNet.GetCanonicalPath(p)
```

---

## 189 Relativen Pfad ermitteln

Haben Sie z.B. von einem Standard-Dialog (OpenFileDialog / SaveFileDialog) einen absoluten Pfad erhalten, benötigen aber einen relativen, beispielsweise bezogen auf ein Projektverzeichnis, dann ist auch hier wieder der Aufruf einer API-Funktion (PathRelativePathTo) angesagt. Zwar gibt es auch die Möglichkeit, über die Klasse Uri relative Pfade generieren zu lassen, aber bei gewöhnlichen Verzeichnispfaden führt das oft zu unerwünschten Ergebnissen (insbesondere dann, wenn der Pfad Leerzeichen enthält).



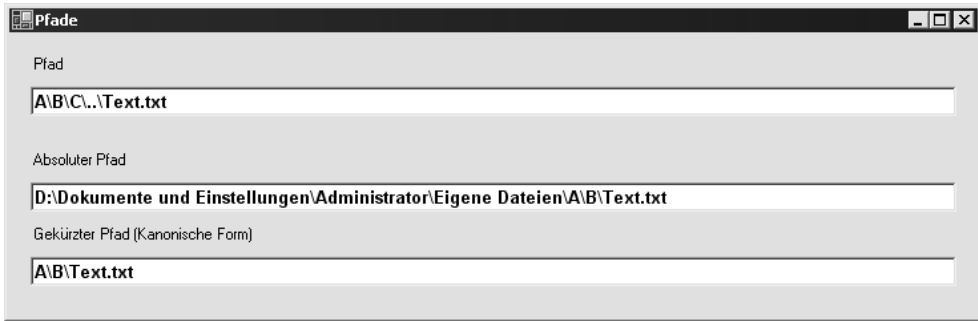


Abbildung 180: Ermitteln des absoluten und des gekürzten Pfads

Listing 344 zeigt die Implementierung der Wrapper-Funktion `GetRelativePath`. Sie bestimmt für den Parameter `pathTo` den Pfad, der, bezogen auf den Basispfad im Parameter `pathFrom`, den relativen Pfad darstellt (vergleiche auch Abbildung 181). Wird also PFAD 1 (`pathFrom`) als das aktuelle Verzeichnis eingestellt, dann ist der relative Pfad identisch mit PFAD 2 (`PathTo`).

Damit die Methode unterscheiden kann, ob ein Pfad ein Verzeichnis oder eine Datei adressiert, muss sowohl für `pathFrom` als auch für `pathTo` ein Flag angegeben werden (`pathFromIsDirectory` bzw. `pathToIsDirectory`). Nutzen Sie das kleine Testprogramm auf der Buch-CD, um die Auswirkungen verschiedener Einstellungen nachzuvollziehen.

```
Public Class API

    Public Declare Auto Function PathRelativePathTo _
        Lib "shlwapi.dll" (ByVal relPath As System.Text.StringBuilder, _
        ByVal pathFrom As String, ByVal attrFrom As Integer, _
        ByVal pathTo As String, ByVal attrTo As Integer) As Boolean
End Class

Public Class ApiVBNNet

    ...

    Public Shared Function GetRelativePath(ByVal pathFrom As String, _
        ByVal pathFromIsDirectory As Boolean, ByVal pathTo As String, _
        ByVal pathToIsDirectory As Boolean) As String

        ' StringBuilder für Textbearbeitung durch GetRelativePath
        Dim sb As New System.Text.StringBuilder(300)

        ' Attribute setzen
        Dim attrFrom, attrTo As API.FileAttribute
        If pathFromIsDirectory Then attrFrom = _
            API.FileAttribute.Directory
        If pathToIsDirectory Then attrTo = API.FileAttribute.Directory

        ' Methode aufrufen
        Dim b As Boolean = API.PathRelativePathTo(sb, pathFrom, _
```

Listing 344: Wrapper-Funktion für den Aufruf von `PathRelativePathTo`

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Propor-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

---

```

        attrFrom, pathTo, attrTo)

    ' Im Erfolgsfall erzeugten String zurückgeben, sonst Leerstring
    If b Then Return sb.ToString()
    Return " "

End Function

End Class

```

---

Listing 344: Wrapper-Funktion für den Aufruf von PathRelativePathTo (Forts.)

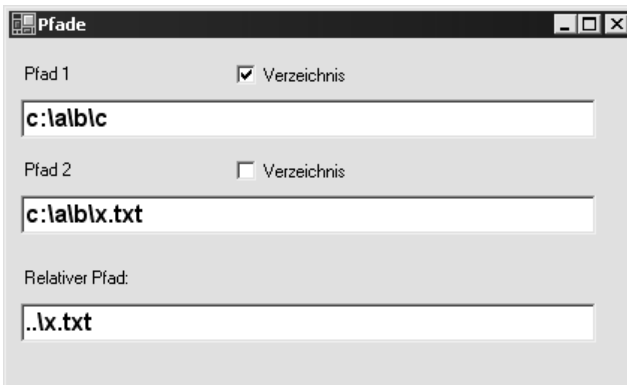


Abbildung 181: Ermitteln des relativen Pfads

## 190 Icons und Typ einer Datei ermitteln

Spezielle Datei-Informationen, die sehr spezifisch für Windows-Betriebssysteme sind, werden nicht über die Klasse `System.IO.FileInfo` zur Verfügung gestellt. Dazu gehören unter anderem der Dateityp, der aus der Dateierweiterung bestimmt und beispielsweise im Explorer angezeigt wird, sowie die Symbole, die einer Datei zugeordnet sind. Für die Ansicht im Explorer sind zwei Symbole von Interesse:

- ▶ `LargeIcon` für die Darstellung »Große Symbole« und
- ▶ `SmallIcon` für alle anderen Darstellungen.

Da die Informationen bislang nicht im Framework mit managed Code abrufbar sind, muss eine API-Funktion zu diesem Zweck bemüht werden. Windows stellt hierfür die Methode `SHGetFileInfo` zur Verfügung (Listing 345). Über den Parameter `uFlags` wird gesteuert, welche Informationen gewünscht werden, über die `SHFILEINFO`-Struktur, auf die der Parameter `psfi` zeigt, werden diese Informationen zurückgegeben.

---

```

Public Declare Auto Function SHGetFileInfo Lib "shell32.dll" ( _
    ByVal pszPath As String, _
    ByVal dwFileAttributes As Integer, _
    ByRef psfi As SHFILEINFO, _

```

---

Listing 345: Declare-Statement für die API-Funktion `SHGetFileInfo` zur Abfrage von Dateitypen, Icons usw.

```

ByVal cbFileInfo As Integer, _
ByVal uFlags As Integer) _
As IntPtr

```

*Listing 345: Declare-Statement für die API-Funktion SHGetFileInfo zur Abfrage von Dateitypen, Icons usw. (Forts.)*

Der Parameter `uFlags` kann eine Bitkombination entgegennehmen und so mehrere Abfragen gleichzeitig ermöglichen (z.B. `Dateityp` und `Icon`). Die wichtigsten Werte sind in Listing 346 als Enumeration (`SHGetFileInfoConstants`) definiert. Es gibt aber Bitkombinationen, die sich gegenseitig ausschließen (siehe MSDN). Listing 347 zeigt die Datenstruktur `SHFILEINFO`. Sie enthält zwei Char-Arrays, die über das Attribut `MarshalAs` mit einer festen Länge definiert werden müssen.

```

Public Enum SHGetFileInfoConstants
    SHGFI_TYPENAME = &H400
    SHGFI_ATTRIBUTES = &H800
    SHGFI_EXETYPE = &H2000
    SHGFI_LARGEICON = 0
    SHGFI_SMALLICON = 1
    SHGFI_ICON = &H100
End Enum

```

*Listing 346: Häufig benötigte Konstanten für SHGetFileInfoConstants*

```

<StructLayout(LayoutKind.Sequential, CharSet:=CharSet.Auto)> _
Public Structure SHFILEINFO
    Public hIcon As Int32
    Public iIcon As Int32
    Public dwAttributes As Int32
    <MarshalAs(UnmanagedType.ByValTStr, SizeConst:=260)> _
    Public szDisplayName As String
    <MarshalAs(UnmanagedType.ByValTStr, SizeConst:=80)> _
    Public szTypeName As String
End Structure

```

*Listing 347: Datenstruktur für den Aufruf von SHGetFileInfo*

Wird ein Icon einer Datei abgefragt, dann erfolgt die Rückgabe über `hIcon` in Form eines Handles. Der Aufrufer der Methode `SHGetFileInfo` ist selbst dafür verantwortlich, dieses Handle wieder freizugeben. Geschieht das nicht, gehen Windows-Ressourcen verloren. Für die Freigabe wird die Methode `DestroyIcon` benötigt (siehe Listing 348).

```

Public Declare Auto Function DestroyIcon _
    Lib "user32.dll" (ByVal hicon As IntPtr) As Boolean

```

*Listing 348: Benötigte Methode zur Freigabe eines Icon-Handles*

Damit Sie als Programmierer die Dateiinformationen nicht aus verschiedenen Quellen zusammensuchen müssen, soll hier die Klasse `FileInformation` definiert werden, die die Informationsbeschaffung zentralisiert. Die Klasse soll die folgenden Informationen bereitstellen:

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Proper-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

1. Dateinamen
2. Dateityp
3. Größe in Byte
4. Datum der letzten Änderung
5. Großes Symbol
6. Kleines Symbol

Für zusätzliche Informationen können Sie die Klasse leicht erweitern. Orientieren Sie sich dazu an der nachfolgenden Beschreibung.

Da die Klasse `FileInfo` bereits einen Teil dieser Informationen liefert (siehe auch Rezept 9.8 ff. ((Verzeichnisgröße mit Unterverzeichnissen))), wäre ein konsequenter Ansatz, die neue Klasse von `FileInfo` abzuleiten. Leider unterbindet das Framework dieses Vorhaben, da `FileInfo` versiegelt (`NotInheritable`) ist. So bleibt nur die Möglichkeit, alle benötigten Eigenschaften als Member-Variablen (oder alternativ als Properties) zu definieren. Listing 349 zeigt die Definition als schreibgeschützte Member-Variablen.

---

```
Public Class FileInformation
    Implements IDisposable

    ' Dateiname
    Public ReadOnly Name As String

    ' Dateityp, wie er im Explorer angezeigt wird
    Public ReadOnly Filetype As String

    ' Dateigröße
    Public ReadOnly Length As Long

    ' Datum der letzten Änderung
    Public ReadOnly LastChanged As DateTime

    ' Icons, wie sie im Explorer angezeigt werden
    Public ReadOnly LargeIcon As Bitmap
    Public ReadOnly SmallIcon As Bitmap
    ...
End Class
```

---

Listing 349: Eigenschaften der Klasse `FileInformation`

Beachten Sie bitte, dass die Icons als Bitmap gespeichert werden. Dadurch werden GDI+-Ressourcen belegt, die freigegeben werden müssen. Deswegen implementiert die Klasse die Schnittstelle `IDisposable` und gibt diese Ressourcen in der Methode `Dispose` (Listing 350) wieder frei.

---

```
' Entsorgung
Public Sub Dispose() Implements System.IDisposable.Dispose
    LargeIcon.Dispose()
    SmallIcon.Dispose()
End Sub
```

---

Listing 350: Freigabe belegter Ressourcen

Ein einziger öffentlicher Konstruktor, der als Parameter den Dateipfad übernimmt, steht für die Instanzierung zur Verfügung (Listing 351). Zunächst werden die Informationen abgerufen, die über `FileInfo` verfügbar sind. Anschließend wird `SHGetFileInfo` aufgerufen, um im ersten Schritt den Dateityp und das kleine Symbol der Datei zu erhalten. Aus dem zurückgegebenen Icon-Handle wird mit `Bitmap.FromHicon` ein Bitmap-Objekt erstellt. Danach wird das Handle wieder freigegeben.

Mit dem zweiten Aufruf von `SHGetFileInfo` wird das große Symbol (`LageIcon`) der Datei abgerufen. Auch hiervon wird wieder ein Bitmap-Objekt angelegt und das Handle anschließend freigegeben. Nach Beendigung des Konstruktors stehen alle Daten in den schreibgeschützten Member-Variablen bereit.

```
Public Sub New(ByVal path As String)

    ' Dateiinformationen über FileInfo-Klasse holen
    Dim fi As New System.IO.FileInfo(path)

    ' Informationen übernehmen
    Me.Name = fi.Name
    Me.Length = fi.Length
    Me.LastChanged = fi.LastWriteTime

    ' Datenstruktur für SHGetFileInfo
    Dim shfi As API.SHFILEINFO

    ' Dateityp und SmallIcon abfragen
    API.SHGetFileInfo(path, 0, shfi, Len(shfi), _
        API.SHGetFileInfoConstants.SHGFI_TYPENAME Or _
        API.SHGetFileInfoConstants.SHGFI_SMALLICON Or _
        API.SHGetFileInfoConstants.SHGFI_ICON)

    ' Dateityp übernehmen
    Me.Filetype = shfi.szTypeName

    ' Icon als Bitmap-Objekt anlegen und übernehmen
    Dim ip As New IntPtr(shfi.hIcon)
    Me.SmallIcon = Bitmap.FromHicon(ip)
    ' Handle freigeben!
    API.DestroyIcon(ip)

    ' LargeIcon abfragen
    API.SHGetFileInfo(path, 0, shfi, Len(shfi), _
        API.SHGetFileInfoConstants.SHGFI_LARGEICON Or _
        API.SHGetFileInfoConstants.SHGFI_ICON)

    ' Icon als Bitmap-Objekt anlegen und übernehmen
    ip = New IntPtr(shfi.hIcon)
    Me.LargeIcon = Bitmap.FromHicon(ip)
    ' Handle freigeben
    API.DestroyIcon(ip)

End Sub
```

Listing 351: Im Konstruktor der Klasse `FileInformation` werden die Informationen gesammelt

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung  
Win-  
dows  
Forms

Controls

Propor-  
tGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

Der Einsatz der Klasse gestaltet sich recht einfach, da lediglich eine Instanz angelegt werden muss und der einzige zu übergebende Parameter der Dateipfad ist. Mit dem folgenden Code

---

```
' Lesen der Informationen
Dim fi As New FileInformation(TXTFile.Text)

' Setzen der Steuerelemente
TXTName.Text = fi.Name
TXTTyp.Text = fi.Filetype
TXTLength.Text = fi.Length.ToString()
PBSmall.Image = fi.SmallIcon
PBLarge.Image = fi.LargeIcon
```

---

werden die Datei-Informationen abgerufen und die Steuerelemente der Beispielanwendung (Abbildung 182) mit den entsprechenden Werten besetzt.

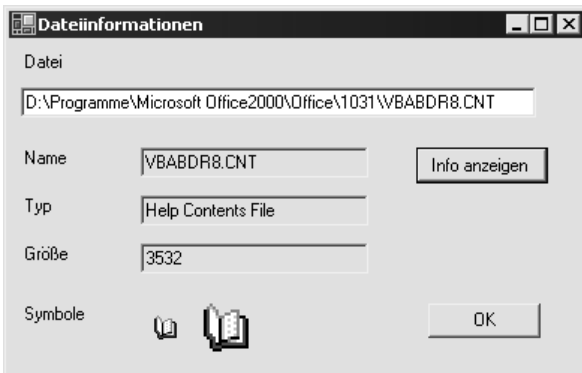


Abbildung 182: Die Klasse *FileInformation* stellt auch den Dateityp und die Symbole bereit

## 191 Dateien kopieren, verschieben, umbenennen und löschen mit SHFileOperation

Sicher kennen Sie das animierte Dialogfenster mit den fliegenden Blättern, das Windows anzeigt, wenn z.B. eine Reihe von Dateien kopiert werden. Während der Aktion werden die bearbeiteten Dateien und eine geschätzte Ausführungsdauer angezeigt. Beim Löschen von Dateien im Explorer werden diese normalerweise nicht wirklich gelöscht, sondern lediglich in den Papierkorb verschoben.

Die in den anderen Rezepten vorgestellten bzw. genutzten Dateioperationen greifen direkt auf das Dateisystem zu und nutzen nicht die Möglichkeiten der Windows-Shell. Uns ist auch nicht bekannt, dass im Framework eine direkte Unterstützung vorgesehen ist. Somit bleibt, wenn man die animierten Dialoge in .NET nutzen will, nur der Umweg über das API.

Listing 352 zeigt die Deklaration für die API-Funktion *SHFileOperation*. Die Methode benötigt einen Zeiger auf eine Struktur vom Typ *SHFILEOPSTRUCT*, die die notwendigen Informationen enthält. In *wFunc* wird die auszuführende Aktion angegeben. Hierfür wurden vier Konstanten als Enumeration (*SHFileOpConstants*) definiert. *pFrom* gibt die Datei bzw. die Dateien an, die als Quelle der Operation dienen sollen, *pTo* diejenigen, die als Ziel verwendet werden sollen, sofern die Operation eines benötigt.

## Hinweis

Der Aufruf von `SHFileOperation` bringt eine Reihe Besonderheiten und Gefahren mit sich. Beachten Sie bitte unbedingt die Hinweise, die zur Methode und zur Struktur `SHFILEOPSTRUCT` in der MSDN-Doku gegeben werden.

Während in `pFrom` Wildcards für Dateien verwendet werden dürfen, ist dies für `pTo` nicht erlaubt. Dateien sollten immer mit absoluten Pfaden angegeben werden. Mehrere Datei-Angaben werden mit einem Null-Character voneinander getrennt. Sowohl `pFrom` als auch `pTo` müssen mit einem doppelten Null-Character terminiert werden!

Über die Eigenschaft `fFlags` lässt sich das Verhalten der vier Operationen steuern. Die Eigenschaft wurde wiederum als Enumeration definiert, so dass die Konstanten alle als Enum-Werte zur Verfügung stehen. Alle Flags lassen sich bitweise kombinieren.

*' SHFileOperation für Dateioperationen mit Windows-Shell*

```
Public Declare Auto Function SHFileOperation Lib "shell32.dll" _
    (ByRef lpFileOp As SHFILEOPSTRUCT) As Integer
```

*' Unterstützte Kommandos für SHFileOperation*

```
Public Enum SHFileOpConstants
    Move = 1
    Copy = 2
    Delete = 3
    Rename = 4
End Enum
```

*' Struktur für SHFileOperation*

```
<StructLayout(LayoutKind.Sequential, Pack:=1, _
    CharSet:=CharSet.Auto)> _
Public Structure SHFILEOPSTRUCT
    Public hwnd As IntPtr
    Public wFunc As SHFileOpConstants
    Public pFrom As String
    Public pTo As String
    Public fFlags As SHFileOpFlagConstants
    Public fAnyOperationsAborted As Boolean
    Public hNameMappings As IntPtr
    Public lpszProgressTitle As String
End Structure
```

*' Flag-Definitionen für SHFileOperation*

```
<Flags()> Public Enum SHFileOpFlagConstants As Integer
    Multitestfiles = 1
    Confirmmouse = 2
    Silent = 4
    RenameOnCollision = &H8
    NoConfirmation = &H10
    WantMappingHandle = &H20
    AllowUndo = &H40
    FilesOnly = &H80
```

---

```

SimpleProgress = &H100
NoConfirmMakeDir = &H200
NoErrorUI = &H400
NoCopySecurityAttribs = &H800
End Enum

```

---

*Listing 352: API-Deklarationen für SHFileOperation und die benötigten Strukturen und Flags (Forts.)*

Aus Sicherheitsgründen werden die Aufrufe in der Klasse `ShellFileOperations` (Listing 353) gekapselt. Für die Datei bzw. Verzeichnislisten wird je eine Instanz der Klasse `StringCollection` angelegt. Die Referenzen der Listen sind über die `ReadOnly`-Eigenschaften `Sourcefiles` und `Destinationfiles` erreichbar.

Alle Flags werden über boolesche Properties gekapselt. In Listing 354 ist aus Platzgründen nur die Eigenschaft `NoConfirmation` vollständig abgedruckt. Alle anderen sind analog aufgebaut. Das Setzen oder Rücksetzen einer Eigenschaft ändert das zugehörige Bit in der Datenstruktur `shFileOpData`.

---

```

Imports System.Collections.Specialized

Public Class ShellFileOperations

    ' Liste der Quelldateien/-verzeichnisse
    Protected source As New StringCollection
    ' Liste der Zieldateien/-verzeichnisse
    Protected destination As New StringCollection

    ' Benötigte Datenstruktur
    Protected shFileOpData As API.SHFILEOPSTRUCT

    ' Handle des übergeordneten Fensters
    Public Property WindowHandle() As IntPtr
    Get
        Return shFileOpData.hwnd
    End Get
    Set(ByVal Value As IntPtr)
        shFileOpData.hwnd = Value
    End Set
End Property

    ' Abrufen der Quelldatei-Liste
    Public ReadOnly Property Sourcefiles() As StringCollection
    Get
        Return source
    End Get
End Property

    ' Abrufen der Zieldatei-Liste
    Public ReadOnly Property Destinationfiles() As StringCollection
    Get

```

---

*Listing 353: Die Eigenschaften der Klasse `ShellFileOperations` gestatten den Zugriff auf die für `SHFileOperation` benötigten Daten*



```

        Return destination
    End Get
End Property

' Flag NoConfirmation
Public Property NoConfirmation() As Boolean
    Get
        ' Flag abfragen
        Return (shFileOpData.fFlags And _
            API.SHFileOpFlagConstants.NoConfirmation) <> 0
    End Get

    Set(ByVal Value As Boolean)
        If Value Then
            ' Flag setzen
            shFileOpData.fFlags = shFileOpData.fFlags Or _
                API.SHFileOpFlagConstants.NoConfirmation
        Else
            ' Flag löschen
            shFileOpData.fFlags = shFileOpData.fFlags And Not _
                API.SHFileOpFlagConstants.NoConfirmation
        End If
    End Set
End Property

' Flag Multidestfiles
Public Property Multidestfiles() As Boolean
    ...
End Property

' Flag RenameOnCollision
Public Property RenameOnCollision() As Boolean
    ...
End Property

' Flag AllowUndo
Public Property AllowUndo() As Boolean
    ...
End Property

' Flag FilesOnly
Public Property FilesOnly() As Boolean
    ...
End Property

' Flag SimpleProgress
Public Property SimpleProgress() As Boolean
    ...
End Property

```

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Propor-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

Listing 353: Die Eigenschaften der Klasse *ShellFileOperations* gestatten den Zugriff auf die für *SHFileOperation* benötigten Daten (Forts.)

---

```

' Flag NoConfirmMakeDir
Public Property NoConfirmMakeDir() As Boolean
...
End Property

' Flag NoErrorUI
Public Property NoErrorUI() As Boolean
...
End Property

' Flag NoCopySecurityAttribs
Public Property NoCopySecurityAttribs() As Boolean
...
End Property

...

End Class

```

---

*Listing 353: Die Eigenschaften der Klasse ShellFileOperations gestatten den Zugriff auf die für SHFileOperation benötigten Daten (Forts.)*

Die vier Operationen (Kopieren, Löschen, Verschieben und Umbenennen) werden über öffentliche Methoden bereitgestellt (Listing 354). In jeder Methode wird das Strukturfeld `wFunc` gesetzt, die Dateilisten in Null-Character-getrennte Zeichenketten gewandelt und die Methode `ShFileOperation` aufgerufen. Deren Rückgabewert (0 steht für fehlerfrei) wird als boolescher Wert (hier steht `True` für fehlerfrei) zurückgegeben.

---

```

Public Class ShellFileOperations
...

' Kopieren von Dateien/Verzeichnissen
Public Function Copy() As Boolean

    ' Auszuführende Funktion
    shFileOpData.wFunc = API.SHFileOpConstants.Copy

    ' Quell- und Zielliste zusammensetzen
    shFileOpData.pFrom = CreateStringlist(source)
    shFileOpData.pTo = CreateStringlist(destination)

    ' Operation ausführen
    Return API.SHFileOperation(shFileOpData) = 0

End Function

' Löschen von Dateien/Verzeichnissen
Public Function Delete() As Boolean

```

---

*Listing 354: Die Methoden der Klasse ShFileOperations führen die gewünschten Datei-Operationen aus*

```

' Auszuführende Funktion
shFileOpData.wFunc = API.SHFileOpConstants.Delete

' Quellliste zusammensetzen
shFileOpData.pFrom = CreateStringlist(source)

' Operation ausführen
Return API.SHFileOperation(shFileOpData) = 0

```

End Function

```

' Verschieben von Dateien/Ordern
Public Function Move() As Boolean

' Auszuführende Funktion
shFileOpData.wFunc = API.SHFileOpConstants.Move

' Quell- und Zielliste zusammensetzen
shFileOpData.pFrom = CreateStringlist(source)
shFileOpData.pTo = CreateStringlist(destination)

' Operation ausführen
Return API.SHFileOperation(shFileOpData) = 0

```

End Function

```

' Umbenennen von Dateien und Ordnern
Public Function Rename() As Boolean

' Auszuführende Funktion
shFileOpData.wFunc = API.SHFileOpConstants.Rename

' Quell- und Zielliste zusammensetzen
shFileOpData.pFrom = CreateStringlist(source)
shFileOpData.pTo = CreateStringlist(destination)

' Operation ausführen
Return API.SHFileOperation(shFileOpData) = 0

```

End Function

```

Public Shared Function CreateStringlist(ByVal strings As _
StringCollection) As String

Dim list As String

' Alle Strings der Liste anhängen und mit Null-Char trennen
For Each s As String In strings
list &= s & ChrW(0)
Next

```

Listing 354: Die Methoden der Klasse *ShFileOperations* führen die gewünschten Datei-Operationen aus (Forts.)

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Propor-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

```
' Zusätzlichen Null-Char anhängen
list &= ChrW(0)
Return list
```

```
End Function
```

```
End Class
```

*Listing 354: Die Methoden der Klasse `ShFileOperations` führen die gewünschten Datei-Operationen aus (Forts.)*

Zur Konvertierung der String-Listen wird die Methode `CreateStringlist` (Listing 354) eingesetzt. Sie durchläuft eine Liste und hängt alle Einträge getrennt durch einen Null-Character aneinander. Abschließend wird ein zusätzlicher Null-Character angehängt, damit die API-Funktion das Listenende erkennen kann.

## Kopieren

Um Dateien zu kopieren, müssen die Pfade der Quelldateien der `Sourcefiles`-Auflistung hinzugefügt werden. Die `Destinationfiles`-Auflistung kann z.B. aus einem einzigen Pfad bestehen:

```
Dim FileOP As New ShellFileOperations
FileOP.Sourcefiles.Add("c:\found.000\file0000.chk")
FileOP.Sourcefiles.Add("c:\temp\*.msi")
FileOP.Destinationfiles.Add("c:\temp\test3")
```

```
FileOP.Copy()
```

Der Code kopiert die beiden Source-Dateien in das Verzeichnis `c:\temp\test3`. Während des Kopiervorgangs wird ein animierter Dialog wie in Abbildung 183 angezeigt. Existiert das Zielverzeichnis nicht, wird der Anwender gefragt, ob es angelegt werden soll (Abbildung 184). Die Abfrage lässt sich über das Flag `NoConfirmMakeDir` unterdrücken.

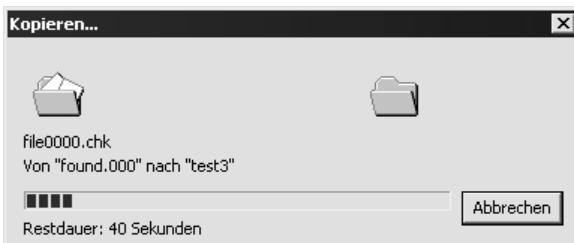


Abbildung 183: Animierter Dialog beim Kopieren von Dateien

Wenn im Zielordner bereits Dateien existieren, deren Namen mit denen der Quelldateien übereinstimmen, wird der ERSETZEN-Dialog wie in Abbildung 185 angezeigt. Auch dieser lässt sich unterdrücken, indem das Flag `NoConfirmation` gesetzt wird.

Wenn das Flag `Multidestfiles` gesetzt ist, können Sie für jede Quelldatei in der `Destinationfiles`-Auflistung einen Zielpfad angeben. So lassen sich z.B. mehrere Dateien unter anderem Namen mit einem Aufruf kopieren.

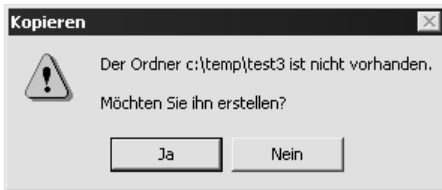


Abbildung 184: Automatische Abfrage, falls der Zielordner nicht existiert



Abbildung 185: Unterdrückbare Abfrage zum Überschreiben von Dateien

## Hinweis

Die animierten Dialoge werden nur angezeigt, wenn die Dateioperation einen längeren Zeitraum in Anspruch nimmt. Bei Operationen, die innerhalb einiger Sekunden ausgeführt werden, unterbleibt die Anzeige des Dialogfensters.

## Löschen

Über das Flag `AllowUndo` können Sie steuern, ob die Dateien in den Papierkorb verschoben (`True`) oder ob sie sofort gelöscht werden sollen (`False`):

```
Dim FileOP As New ShellFileOperations
FileOP.AllowUndo = True
FileOP.Sourcefiles.Add("c:\temp\test\*")
FileOP.Delete()
```

Auch hier werden wieder Abfrage-Dialoge (Abbildung 186) angezeigt, die über Flags unterdrückt werden können. Während des Löschens wird ein animierter Dialog wie in Abbildung 187 angezeigt. Die Operation benötigt keine Liste von Zieldateien.

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Propor-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

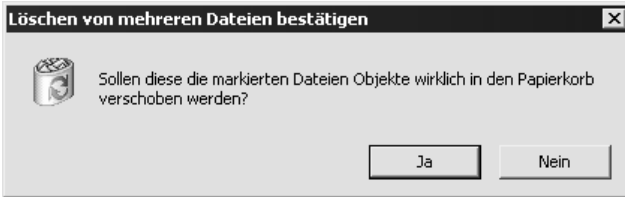


Abbildung 186: Sicherheitsabfrage vor dem Löschen der Dateien

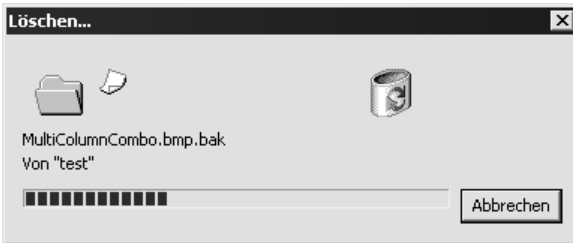


Abbildung 187: Animierung während des Löschvorgangs

## Verschieben

Das Verschieben von Dateien verläuft analog zum Kopieren:

---

```
Dim FileOP As New ShellFileOperations
FileOP.AllowUndo = True
FileOP.Sourcefiles.Add("c:\temp\test\*")
FileOP.Destinationfiles.Add("c:\temp\xxx")
FileOP.Move()
```

---

Während des Verschiebevorgangs (sofern die Zeit reicht) wird ein animierter Dialog wie in Abbildung 188 angezeigt.

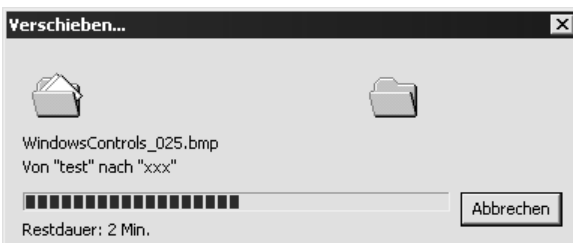


Abbildung 188: Animierter Dialog während des Verschiebens von Dateien

## Umbenennen

Als vierte und letzte Operation lassen sich Dateien mit `SHFileOperation` umbenennen. Aufgrund der geringen Ausführungsgeschwindigkeit wird es aber selten dazu kommen, dass ein animierter Dialog angezeigt wird. Uns jedenfalls ist es nicht gelungen, einen solchen zu provozieren. Das Umbenennen mehrere Dateien, insbesondere, wenn auch Verzeichnisse umbenannt werden sollen, lässt sich besser mit der Verschieben-Operation (`Move`) bewerkstelligen.

```
Dim FileOP As New ShellFileOperations
FileOP.Sourcefiles.Add("c:\temp\test.txt")
FileOP.Destinationfiles.Add("c:\temp\test.aaa")

FileOP.Rename()
```

|                           |
|---------------------------|
| Basics                    |
| Datum/<br>Zeit            |
| Anwen-<br>dungen          |
| Zeich-<br>nen             |
| Bild-<br>bearbei-<br>tung |
| Win-<br>dows<br>Forms     |
| Controls                  |
| Proper-<br>tyGrid         |
| <b>Datei-<br/>system</b>  |
| Netz-<br>werk             |
| System                    |
| Daten-<br>banken          |
| XML                       |
| Wissen-<br>schaft         |
| Ver-<br>schiede-<br>nes   |





# Netzwerk

In diesem Kapitel sind einige Rezepte zum Thema Intra- und Internet zusammengefasst. Es geht hier beispielsweise um die Bestimmung von IP-Nummern (IPv4 und IPv6) oder die Ermittlung der Domäne. Um zu zeigen, dass Web-Services nicht nur mit ASP.NET genutzt werden können, ist die Nutzung des Google-Web-Services in einem Rezept exemplarisch für diese Art des Netzwerkverkehrs aufgenommen worden.

Wer ein Thema vermisst, sollte sich in den anderen Kapiteln dieses Buches umschauen, da es durchaus sein kann, dass das entsprechende Thema dort behandelt wird. Ein Blick in den Index sollte hier weiter helfen. Wir haben in diesem Buch in den einzelnen Rezepten durchaus Techniken benutzt und beschrieben, die nutzbringend sind, aber keinen Eingang in ein eigenständiges Rezept gefunden haben.

## 192 IPv4-Adressen nach IPv6 umrechnen

Der Adressraum im Internet wird immer enger. Dieses Problem wurde schon vor Jahren erkannt und durch eine neue Netzwerk-Adressierung gelöst. Nach einem schleppenden Start gehen aber mittlerweile viele Provider und Hersteller von Netzwerk-Geräten dazu über, die neue Adressierung (Internet Protocol Version 6, kurz IPv6) neben der »alten« und mehrheitlich noch benutzten Adressierung der Version 4 (kurz IPv4) zu gestatten bzw. intern nur noch zu nutzen. Diese Adressierung fällt durch anfänglich recht kryptische Netzwerkadressen auf.

Als Hilfestellung kann die Funktion `IPv4ToIPv6` aus Listing 355 benutzt werden. Dieser Funktion wird eine gültige IP-Adresse im alten Format übergeben.

---

```
Public Function IPv4ToIPv6(ByVal IPAddress As String) As String
```

```
    Dim Parts() As String
    Dim p1 As Integer
    Dim p2 As Integer
    Dim p3 As Integer
    Dim p4 As Integer
```

```
    Dim IPv6 As String
```

```
    ' IPv4 am . auftrennen
```

```
    Try
```

```
        Parts = IPAddress.Split(CType(".", Char))
```

```
    Catch ex As ArgumentNullException
```

```
        Throw New ApplicationException _
            ("Adresse ist Nothing", ex)
```

```
    Catch ex As ArgumentException
```

```
        Throw New ApplicationException _
            ("Adresse ist leer", ex)
```

```
    Catch ex As Exception
```

---

*Listing 355: Umrechnung alter IPv4-Adressen in neue IPv6-Adressen*

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung  
Win-  
dows  
Forms

Controls

Propert-  
yGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

```

    Throw New ApplicationException _
        ("allg. Ausnahme", ex)
End Try

' IPv4 Adressen sind halt so lang
If Parts.Length <> 4 Then
    Throw New ApplicationException _
        ("keine IPv4-Adresse")
End If

p1 = CType(Parts(0), Integer)
p2 = CType(Parts(1), Integer)
p3 = CType(Parts(2), Integer)
p4 = CType(Parts(3), Integer)

IPv6 = String.Format _
    ("::ff:{0:x2}{1:x2}:{2:x2}{3:x2}", p1, p2, p3, p4)

Return IPv6
End Function

```

---

*Listing 355: Umrechnung alter IPv4-Adressen in neue IPv6-Adressen (Forts.)*

Die IP-Adresse wird mit Hilfe der `Split`-Funktion aus der `String`-Klasse an den Punkten aufgebrochen und als entsprechende einzelne Adressteile in den Variablen `pi` abgespeichert. Die »Umrechnung« in das neue Format gestaltet sich dadurch recht einfach, man muss diese Teile nur noch richtig ausgeben.

IPv4-Ziffern in der IPv6-Schreibweise sind an dem führenden »:ff« zu erkennen. Dieser Adressraum wurde den alten IP-Nummern zugeordnet. Anschließend folgen der erste und zweite Wert der alten IP-Adresse (von links betrachtet) in hexadezimaler Schreibweise. Der dritte und vierte Wert werden ebenfalls zusammengefasst und durch einen Doppelpunkt getrennt rechts angehängt.

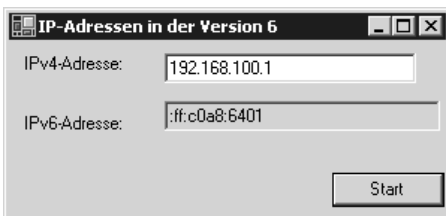


*Abbildung 189: Umwandlung einer IP-Adresse von v4 nach v6*

Das Ergebnis einer solchen Umrechnung ist in Abbildung 189 zu sehen. Die gezeigte Schreibweise ist eine Abkürzung. Wäre diese Umrechnung alles, hätte man nichts gewonnen. Die zusätzlichen Adressen liegen im Bereich links des hier führenden »:ff«. Somit ist auch klar, dass man neue Adressen nur unter bestimmten Bedingungen in die alten (jetzt noch gültigen) Adressen umrechnen kann.

## 193 IPv6-Adressen nach IPv4 umrechnen

Während die Adressen der Version 4 aus 32 Bit bestanden, ist die Adresslänge der Version auf 128 Bit erweitert worden. Damit stehen jetzt nicht 4.294.967.296 ( $2^{32}$ ) Adressen zur Verfügung, sondern  $2^{128} = 340.282.366.920.938.463.463.374.607.431.768.211.456$ . Geschrieben werden diese neuen Adressen in 8 Blöcken zu je 16 Bit Breite, durch einen Doppelpunkt getrennt. Die IP-Adressen der Version 4 finden sich in den 2 kleinsten Blöcken wieder. Zusätzlich wird eine alte Adresse mit einem führenden »:ff« gekennzeichnet.

Die Funktion IPv62IPv4 aus Listing 356 führt eine solche Umrechnung von Adressen durch.

```
Public Function IPv62IPv4(ByVal Address As String) As String
    Dim v6() As String
    Dim p1 As String
    Dim p2 As String
    Dim p3 As String
    Dim p4 As String
    Dim t As Int32

    ' IPv6 am : auftrennen
    Try
        v6 = Address.Split(CType(":", Char))
    Catch ex As ArgumentNullException
        Throw New ApplicationException _
            ("Adresse ist Nothing", ex)
    Catch ex As ArgumentException
        Throw New ApplicationException _
            ("Adresse ist leer", ex)
    Catch ex As Exception
        Throw New ApplicationException _
            ("allg. Ausnahme", ex)
    End Try

    ' IPv6 Adressen für IPv4 sind halt so lang
    If v6.Length <> 4 Then
        Throw New ApplicationException _
            ("keine IPv6-Adresse, oder IPv6 ist nicht nach v4 " + _
            "zu konvertieren")
    End If

    ' erste Hexzahl für IPv6 konvertierbare Adressen ist immer 0xff
    If v6(1).ToLower <> "ff" Then
        Throw New ApplicationException _
            ("keine IPv6-Adresse")
    End If

    ' Hex nach Int32 des ersten Hex-Words der v6-Adresse
    t = Int32.Parse(v6(2), _
        Globalization.NumberStyles.AllowHexSpecifier)

    ' Adressbytes 1 und 2
```

Listing 356: IP-Adressen der Version 6 nach v4 umrechnen

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung  
Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

```

p1 = CType(t \ 256, String)
p2 = CType(t Mod 256, String)

' Hex nach Int32 des zweiten Hex-Words der v6-Adresse
t = Int32.Parse(v6(3), _
    Globalization.NumberStyles.AllowHexSpecifier)

' Adressbytes 3 und 4
p3 = CType(t \ 256, String)
p4 = CType(t Mod 256, String)

' Rückgabe der v4-Adresse
Return p1 + "." + p2 + "." + p3 + "." + p4
End Function

```

---

*Listing 356: IP-Adressen der Version 6 nach v4 umrechnen (Forts.)*

Nach dem Check auf eine gültigen Adresse wird die Adresse an den Doppelpunkten aufgetrennt und in ein Zeichenkettenarray gespeichert. Da eine gültige IPv6-Adresse einen führenden Doppelpunkt hat, ist der Inhalt des Zeichenkettenarrays mit dem Index 0 leer. An der Position mit Index 1 muss ff stehen. Die eigentliche Adresse beginnt also erst ab dem Indexwert 2. Dort steht eine 16-Bit-Zahl, die für die alte Adressierung in zwei 8-Bit-Zahlen getrennt werden muss.

Hierfür wird die 16-Bit-Hexadezimalzahl in einen dezimalen Wert umgerechnet. Hierzu kann man die Parse-Methode des Int32-Typs verwenden. Dieser Methode wird zudem mitgeteilt, dass hexadezimale Zahlen für das Parsen erlaubt sind. Die so generierte 16-Bit-Zahl wird durch zwei Divisionen mit der Zahl 256 (Integer-Division und Modulo-Division) in zwei 8-Bit-Zahlen getrennt. Damit sind die erste und zweite Ziffer der v4-Adressierung bestimmt. Für die dritte und vierte Ziffer wird exakt genauso vorgegangen. Die so errechneten Teile werden zusammengefügt und dem aufrufenden Programm übergeben.

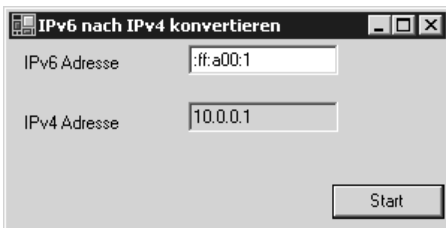


Abbildung 190: IPv6-Adresse nach IPv4 umwandeln

Eine Beispielumrechnung findet sich in Abbildung 190. Die Adresse :ff:a00:1 ist mit führenden Nullen geschrieben :ff:0a00:0001. Aufgeteilt in 4 16-Bit-Zahlen ergibt sich in hexadezimaler Schreibweise 0a.00.00.01, was nichts anderes als die dargestellt Adresse ist.

## 194 IP-Adresse eines Rechners

Die IP-Adresse eines Rechners zu kennen kann in einigen Fällen recht nützlich sein. Hat ein Rechner im internen Netz eine feste IP-Adresse, so ist das Problem schnell gelöst. Wird aber DHCP eingesetzt oder man ist mit einer Wählverbindung im Internet, ist die Adresse bei jeder Anmeldung typischerweise eine andere.

Mit .NET ist es aber recht einfach, sich diese Information zu beschaffen. Bindet man mit

`Imports System.Net`

den `System.Net`-Namensraum in ein Programm ein, kann man die Funktion `GetIP` aus Listing 357 zu deren Ermittlung einsetzen.

```
Public Function GetIP(Optional ByVal Computer As String = "") _
    As ArrayList

    Dim ComputerName As String
    Dim Host As IPEndPoint
    Dim HostAddresses() As IPAddress
    Dim Num As Integer
    Dim IPs As ArrayList = New ArrayList

    If Computer = String.Empty Then
        ComputerName = Dns.GetHostName
    End If

    Host = Dns.GetHostByName(ComputerName)
    HostAddresses = Host.AddressList

    For Num = 0 To HostAddresses.GetLength(0) - 1
        IPs.Add(HostAddresses(Num).ToString)
    Next

    Return IPs
End Function
```

*Listing 357: IP-Adresse eines Rechners feststellen*

Der Funktion wird der Computername übergeben. Hierbei kann es sich auch um den Namen eines anderen Computers handeln. Wird kein Name angegeben, sucht die Funktion den Namen des Rechners, auf dem das Programm läuft, mit der Methode `GetHostName` der Klasse `Dns`. Damit die Methode Erfolg hat, muss ein entsprechender Dienst im Netz laufen. Dies ist in einem Netzwerk aber typischerweise der Fall. Befindet man sich im Internet, ist dies sogar garantiert.

Mit der Methode `GetHostByName` wird der Host als `IPHostEntry` zurückgegeben. Mit diesem Eintrag kann nun die Liste der IP-Adressen eines Rechners über die Methode `AddressList` gefunden werden. Ein Rechner kann durchaus mehrere IP-Adressen haben, beispielsweise eine Adresse des lokalen Netzes und eine Adresse aus dem Internet. Dies kann in Abbildung 191 gesehen werden. Die erste Adresse ist die lokale Adresse des privaten Hausnetzes, während die zweite Adresse durch die Einwahl ins Internet über einen Internet-Provider zustande gekommen ist.

Über eine Schleife werden diese Adressen ermittelt und in ein Zeichenketten-Array gespeichert. Dieses wird anschließend dem aufrufenden Programm zurückgegeben.

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Propert-  
yGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes



Abbildung 191: IP-Adressen eines Rechners im Internet

## 195 Netzwerkadapter auflisten

Interessiert man sich für die Netzwerk-Anschlüsse eines Rechners, kann man nicht immer das Handbuch des Motherboards zu Hilfe nehmen. Und im Zeitalter der OnBoard-Anschlüsse wundert man sich manches Mal über die Vielfalt der Netzwerkanschlüsse, die der eigene PC so hat ☺.

Die Funktion `GetAllNetworkAdapter` aus Listing 358 schafft hier Abhilfe. Wie der Name schon vermuten lässt, werden alle durch das Betriebssystem erkennbaren Netzwerkanschlüsse ermittelt und einige Informationen für jeden Anschluss dem aufrufenden Programm gemeldet.

---

```
Public Function GetAllNetworkAdapter() As ArrayList
    Dim MgmtSearch As ManagementObjectSearcher
    Dim MgmtObject As ManagementObject
    Dim List As ArrayList = New ArrayList
    Dim str As String
    Dim Adapter As String
    Dim SearchSQL As String = _
        "Select * from Win32_NetworkAdapter"

    MgmtSearch = New ManagementObjectSearcher("root\cimv2", SearchSQL)

    For Each MgmtObject In MgmtSearch.Get()

        str = MgmtObject("Description")
        If str <> Nothing Then
            List.Add(str)
        Else
            List.Add("N/A")
        End If

        str = MgmtObject("DeviceID")
        If str <> Nothing Then
            List.Add(str)
        Else
            List.Add("N/A")
        End If
    
```

---

Listing 358: Netzwerkadapter eines Rechners auflisten

```
str = MgmtObject("ServiceName")
If str <> Nothing Then
    List.Add(str)
Else
    List.Add("N/A")
End If

Adapter = MgmtObject("AdapterType")
If Adapter <> Nothing Then
    List.Add(Adapter)
Else
    List.Add("N/A")
End If

str = MgmtObject("MACAddress")
If str <> Nothing Then
    List.Add(str)
Else
    List.Add("N/A")
End If

str = MgmtObject("Status")
If str <> Nothing Then
    List.Add(MgmtObject("Status"))
Else
    List.Add("N/A")
End If

List.Add("-----")
Next

Return List
End Function
```

Listing 358: Netzwerkadapter eines Rechners auflisten (Forts.)

Um an die Informationen zu gelangen, wird WMI eingesetzt. Näheres hierzu finden Sie im System-Kapitel und im Anhang. Die Informationen werden mit einer WQL-Abfrage über den Namensraum Win32\_NetworkAdapter zusammengestellt.

Die Eigenschaft Device-ID ist eine eindeutige Nummer innerhalb des Rechners, in dem das Device eingebaut ist. Eine Auswahl an erkennbaren Adaptern ist in Tabelle 27 gelistet.

Wie man sowohl dem Listing 358 als auch der Abbildung 192 entnehmen kann, ist man mit dieser Abfrage auch in der Lage, die MAC-Adresse zu ermitteln. Mit einer kleinen Datenbank im Hintergrund ließe sich damit auch der Hersteller der jeweiligen Karte feststellen, da jedem Hersteller ein gesonderter MAC-Adressraum zugeordnet ist.

| Adaptertypen im Win32_NetworkAdapter-Namensraum (Auswahl) |  |
|-----------------------------------------------------------|--|
| Ethernet 802.3                                            |  |
| Token Ring 802.5                                          |  |

Tabelle 27: Auswahl an ermittelbaren Adaptertypen

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

| Adaptertypen im Win32_NetworkAdapter-Namensraum (Auswahl) |
|-----------------------------------------------------------|
| Fiber Distributed Data Interface (FDDI)                   |
| Wide Area Network (WAN)                                   |
| LocalTalk                                                 |
| ARCNET (878.2)                                            |
| ATM                                                       |
| Wireless                                                  |
| Infrared Wireless                                         |

Tabelle 27: Auswahl an ermittelbaren Adaptertypen (Forts.)

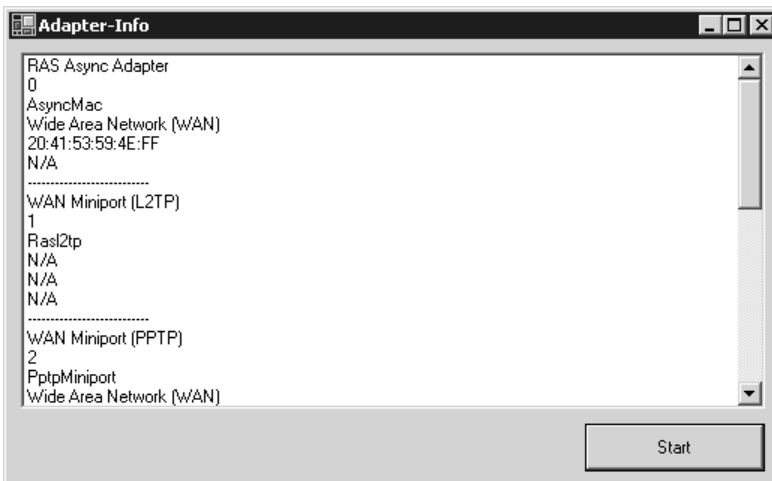


Abbildung 192: Netzwerkadapter eines Rechners anzeigen

Da MAC-Adressen eindeutig sind, lässt sich somit auch feststellen, ob es neue Netzwerkadapter im Hausnetz gibt, die dort eigentlich nichts verloren haben.

## 196 Freigegebene Laufwerke anzeigen

Benötigt man die für das Netz freigegebenen Laufwerke auf einem Rechner, so kann man sich diese Informationen mittels WMI beschaffen. Da die Funktion aus Listing 359 auf WMI beruht, kann man sie auch bei Bedarf auf die Ermittlung der freigegebenen Laufwerke eines beliebigen Netzwerk-PCs erweitern.

```
Public Function GetSharedInfo() As ArrayList
    Dim mQuery As WqlObjectQuery
    Dim mSearch As ManagementObjectSearcher
    Dim mCol As ManagementObject
    Dim mStrSQL As String
    Dim mListe As ArrayList = New ArrayList
```

Listing 359: Freigegebene Laufwerke anzeigen



```
mStrSQL = "Select * from win32_share"

' WMI-Abfrage erstellen
mQuery = New WqlObjectQuery(mStrSQL)
mSearch = New ManagementObjectSearcher(mQuery)

' Schleife über jedes freigegebene Laufwerk
For Each mCol In mSearch.Get()
    mListe.Add("Name = " + mCol("Name").ToString)
    mListe.Add("Caption = " + _
        mCol("Caption").ToString)
    mListe.Add("Beschreibung = " + mCol("description"))
    mListe.Add("Path = " + mCol("path").ToString)
    mListe.Add("Status = " + mCol("Status").ToString)
    mListe.Add("Typ = " + mCol("Type").ToString)
    mListe.Add("-----")
Next
Return mListe

End Function
```

Listing 359: Freigegebene Laufwerke anzeigen (Forts.)

Näheres zu WMI finden Sie im System-Kapitel und im Anhang. Um die freigegebenen Laufwerke feststellen zu können, wird eine WQL-Abfrage im Win32\_Share-Namensraum von WMI durchgeführt. Sollte mindestens ein Laufwerk oder ein Verzeichnis eine Freigabe besitzen, werden in der Schleife Informationen zu dieser Freigabe gesammelt und in einer ArrayList gespeichert. Das Ergebnis dieser Funktion für einen PC mit mehreren freigegebenen Laufwerken ist in Abbildung 193 zu sehen.

| Wert       | Bedeutung                          |
|------------|------------------------------------|
| 0          | Laufwerk                           |
| 1          | Drucker Warteschlange              |
| 2          | Device                             |
| 3          | IPC                                |
| 2147483648 | Laufwerk (Verwaltung)              |
| 2147483649 | Drucker Warteschlange (Verwaltung) |
| 2147483650 | Device (Verwaltung)                |
| 2147483651 | IPC (Verwaltung)                   |

Tabelle 28: Freigabetypen des Betriebssystems in Win32\_Share

Die einzelnen abgefragten Eigenschaften dürften bis auf eine Ausnahme selbsterklärend sein. Die Ausnahme bezieht sich auf den Typ der Freigabe. In Tabelle 28 sind die verschiedenen Freigabetypen aufgeführt. Wie man sieht, können nicht nur Laufwerke freigegeben werden ;-).

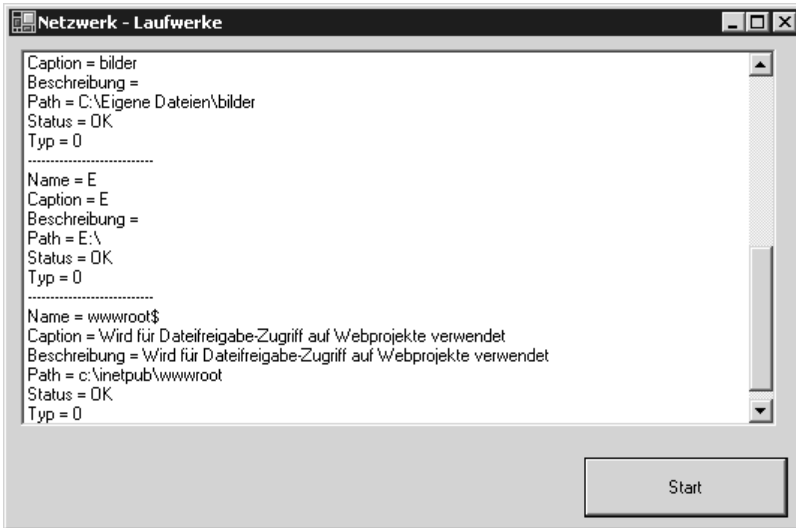


Abbildung 193: Ermittelte Freigaben eines PCs

Für Anwendungen sind die Freigabetypen 0 – 3 interessant, da die anderen Freigabetypen vom Betriebssystem für die Verwaltung des Rechners benutzt werden. Will man allerdings einen Rechner vor Angriffen schützen, sollten diese Freigaben nicht übersehen werden.

## 197 Web-Service

Web-Services sind nicht die alleinige Domäne von Web-Programmiersprachen. Sie dienen allgemein dem Austausch von Informationen und können daher auch in lokalen Netzen und mit »klassischen« Programmiersprachen eingesetzt werden.

Der Austausch von Daten geschieht bei den Web-Services mit dem WSDL-Protokoll. Diese Abkürzung steht für *Web Services Description Language*. Dieses Protokoll gilt plattformübergreifend. Sie können also mit einem Windows-basierten System ohne weiteres auf die Dienste eines Unix-Rechners zugreifen.

Da die Programmierung eines Web-Servers mit WSDL-Protokoll kein Thema für ein Codebook sein kann, sollten Sie sich bei Interesse ein spezielles Werk zu diesem Thema besorgen. Die Programmierung eines Clients ist allerdings angenehm einfach. Wie ein solcher Client erstellt wird, soll am Web-Service von Google demonstriert werden.

Bevor Sie eine Abfrage über diesen Client verschicken können, müssen Sie sich bei Google für diesen Service kostenlos registrieren lassen. Die Web-Seite hierzu ist <http://www.google.com/apis/>. Über diese Seite können Sie auch das SDK herunterladen.

Bestandteil des SDK ist die WSDL-Datei, die Sie auf jeden Fall benötigen. Wenn Sie sich registriert haben, wird Ihnen ein Registrierschlüssel per eMail zugeschickt, den Sie für die Abfragen brauchen. Zurzeit dürfen Sie »nur« 1000 Abfragen pro Tag durchführen.

Da VB.NET mit WSDL direkt nichts anfangen kann, muss diese Datei erst in ein für VB lesbares Format umgewandelt werden. Dazu geben Sie auf der Befehlszeile

```
wsdl /l:vb /p:httppost googlesearch.wsdl
```

ein. Hierdurch wird eine neue Datei erstellt, *GoogleSearchService.vb*. Diese muss in das Projekt eingebunden werden, da in der Datei die Klassen für den Zugriff auf den Web-Service enthalten sind.

Damit das Programm die entsprechenden Klassen des .NET kennt, müssen Sie Verweise auf *System.Web.Service* und *System.XML* einfügen. Die anderen Verweise sind default-mäßig dem Projekt schon zugeordnet. Mit der Funktion *GetGoogle* aus Listing 360 wird die Abfrage realisiert.

Der Funktion wird der suchende Ausdruck als Zeichenkette übergeben. Ein zweiter Übergabeparameter wird per Referenz übergeben, da er die Anzahl der gefundenen Treffer an das aufrufende Programm zurückgibt. Als Erstes wird ein Objekt der Klasse *GoogleSearchService* erstellt. Diese Klasse befindet sich in der gerade erstellten Datei *GoogleSearchService.vb*. Die Variable, die das Suchergebnis aufnehmen soll, ist vom Typ *GoogleSearchResult*, einer Klasse, die sich ebenfalls in der erwähnten Datei befindet. Um die Abfrage zu starten benötigen Sie noch Ihren Identifikationsschlüssel, da dieser bei jeder Abfrage mit übermittelt werden muss.

Die Abfrage erfolgt dann mit der Methode *doGoogleSearch*, der neben den erwähnten Parametern noch einige weitere Einstellungen übergeben werden. So wird in Listing 360 die Anzahl der darzustellenden Antworten auf 10 Stück begrenzt. In vielen Fällen reicht ein hier dargestellter Standardaufruf. Insgesamt sind die Möglichkeiten sehr mannigfaltig und in der Dokumentation recht gut dargestellt.

---

```
' Liefert RTF-String zurück
Public Function GetGoogle(ByVal SearchString As String, _
    ByRef ResultCount As Integer) As String

    ' Objekt aus der mit WSDL erstellten Klasse (.vb)
    Dim GoogleSearch As GoogleSearchService = New GoogleSearchService

    ' Ergebnisobjekt aus der mit WSDL erstellten Klasse
    Dim SearchResult As GoogleSearchResult

    ' Laufvariable für die Ergebnismenge
    Dim i As Integer

    ' Nimmt die aufbereiteten Ergebnisse auf
    Dim ResString As String = String.Empty

    ' Identifikationsschlüssel von Google (personalisiert)
    Dim GoogleKey As String = "diesisteingeheimerschlüssel"

    Try
        ' Die Methode zum Suchen aufrufen
        SearchResult = GoogleSearch.doGoogleSearch(GoogleKey, _
            SearchString, 0, 10, False, "", True, "", "", "")
    Catch ex As Web.Services.Protocols.SoapException
        Throw New ApplicationException _
            ("GetGoogle bei SOAP fehlgeschlagen.", ex)
    End Try

    ' Ergebnisse insgesamt, die zur Verfügung stehen
```

---

Listing 360: Abfrage des Google-Web-Service

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

```

ResultCount = SearchResult.estimatedTotalResultsCount

' Array-Variable mit richtigen Grenzen für das Ergebnis
' bereitstellen, Klasse mit WSDL generiert
Dim Res(SearchResult endIndex) As ResultElement

' Ergebnisse abspeichern
Res = SearchResult.resultElements

' RTF-Header
ResString = "{\rtf1\Ansi\ "

For i = SearchResult.startIndex - 1 To SearchResult.endIndex - 1
    ResString += Res(i).title
    ' \par steht fuer Absatz (RTF)
    ResString += " \par "
    ResString += Res(i).snippet
    ResString += " \par "
    ResString += Res(i).URL
    ResString += " \par ----- \par \par "
Next

' RTF-Abschluss
ResString += "}"

' Ersetze HTML-Tags
ResString = ResString.Replace("<b>", "{\b")
ResString = ResString.Replace("</b>", "}")

Return ResString
End Function

```

---

*Listing 360: Abfrage des Google-Web-Service (Forts.)*

Aus dem Objekt `SearchResult` kann die geschätzte Zahl an Treffern geliefert. Für die einzelnen Ergebnisse der Abfrage wird eine passende Variable deklariert. Die Abfrage liefert zwar nur zehn Antworten zurück, da die Abfrage so gestellt wurde, doch soll es noch vorkommen, dass die Ergebnismenge kleiner ist. Über die Eigenschaft `endIndex` kann die richtige Dimension festgelegt werden.

Die Ergebnisse der Abfrage sollen als RTF-Zeichenkette zurückgegeben werden. Der Grund hierfür liegt einfach darin begründet, dass RTF-Zeichenketten in Formularen dargestellt werden und Hyperlinks benutzt werden können. Hierzu wird der Zeichenkette eine Präambel vorgesetzt, die diese Zeichenkette als RTF-Zeichenkette definiert. In der folgenden Schleife werden einige Informationen aus der Ergebnismenge in der Zeichenkette abgespeichert.

Zum Schluss werden die HTML-Tags für die Fett-Darstellung durch die entsprechenden RTF-Tags ersetzt. Ein Beispiel ist in Abbildung 194 zu sehen. Die Abfrage nach den drei Begriffen »visual basic .net« liefert ca. 1.45 Mio Treffer. Durch die Darstellung als RTF-Text ist ein Anklicken auf den Hyperlink möglich. Wie man dieses Ereignis in der `RichTextBox` abfängt, ist in Kapitel 361 gezeigt. Wie man in Abbildung 194 sieht, sind nicht alle HTML-Tags umgesetzt. So lassen sich noch `<br>`-Tags im Text finden.

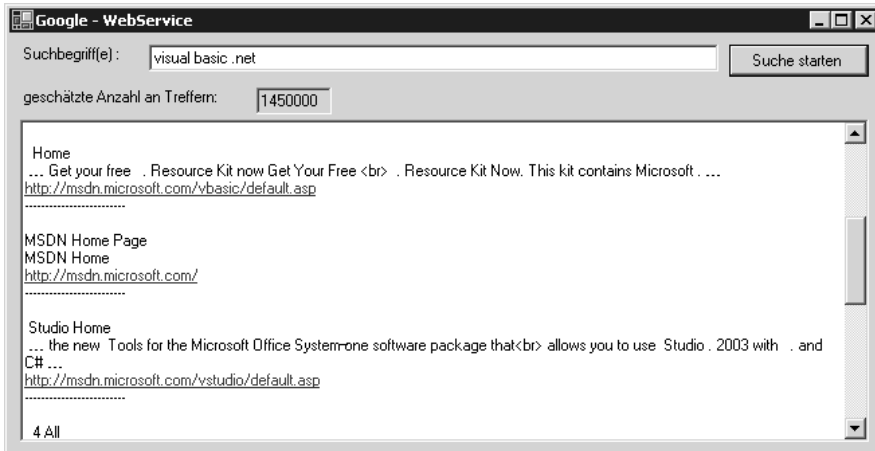


Abbildung 194: Ergebnis einer Google-Abfrage

## 198 Internet Explorer starten

Dieses Rezept könnte auch im System-Kapitel stehen. Da aber ein direkter Zusammenhang mit dem vorhergehenden Rezept besteht, haben wir dieses Rezept an dieser Stelle belassen. Zumal man es auf Grund des Steuerelementes `RichTextBox` auch noch an einer anderen Stelle hätte unterbringen können.

Hat man in einer `RichTextBox` einen Text, der URLs enthält, so kann man diese mit

```
rtxtResult.DetectUrls = True
```

scharf schalten, d.h. sie werden als Hyperlinks angezeigt. Was noch fehlt, ist eine Funktion, die beim Klicken auf eine URL den Internet Explorer mit der entsprechenden Seite startet. Diese Funktionalität wird durch ein Ereignis in der `RichTextBox` zur Verfügung gestellt: `LinkClicked`.

```
' HTML-Link im RTF-Steuerelement wurde angeklickt
Private Sub rtxtResult_LinkClicked _
    (ByVal sender As Object, ByVal e As _
    System.Windows.Forms.LinkClickedEventArgs) _
    Handles rtxtResult.LinkClicked

    ' Prozess-Variable zum Starten des Internet Explorers
    Dim proc As New System.Diagnostics.Process

    ' Starte Internet Explorer mit angeklickter Seite
    proc = System.Diagnostics.Process.Start("IExplore.exe", e.LinkText)

    ' Warten, bis IE beendet
    proc.WaitForExit()

    ' Entsorgung
    proc.Dispose()
End Sub
```

Listing 361: Internet-Explorer per Programm starten und definierte Seite anzeigen

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Propor-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

Der Ereignisroutine wird neben dem immer auftauchenden `Sender` ein Parameter vom Typ `System.Windows.Forms.LinkClickedEventArgs` übergeben. Diese Variable hat eine Eigenschaft `LinkText`, die die aktivierte URL enthält. Damit kann dann über die Methode `Start` der Klasse `Process` aus dem Namensraum `System.Diagnostics` der Internet Explorer mit der entsprechenden Seite aufgerufen werden.

Die Funktion wartet so lange, bis der Internet Explorer beendet wird, und verlässt erst dann die Ereignisroutine. Wenn dies nicht gewünscht wird, muss die Zeile `proc.WaitForExit` auskommentiert oder gelöscht werden.

# System

In diesem Abschnitt des Buches geht es um das Herzstück eines Rechners, das System. Da viele Informationen über WMI ermittelt werden, werden zu Beginn einige Rezepte vorgestellt, die sicherstellen, dass WMI auch auf dem Rechner vorhanden ist und welche Informationen das installierte WMI bietet. Eine kleine Einführung in die Thematik des WMI finden Sie im Anhang.

Es werden in den einzelnen Rezepten teilweise unterschiedliche Vorgehensweisen bei der Informationsbeschaffung mit WMI gezeigt. Dies resultiert aus den unterschiedlichsten Möglichkeiten, WMI anzusteuern. Selbst wenn man in den direkten Schranken des .NET bleibt, kann man sehr unterschiedlich an das Problem herangehen.

In den Rezepten wird gezeigt, wie man die Hardware bestimmen kann, unter der ein System laufen soll. Dies beginnt beim BIOS und endet bei den physikalischen und logischen Festplatten des Systems. Dies ist die eine Seite der Medaille. Die andere Seite besteht aus dem Betriebssystem und den laufenden Prozessen und Einstellungen. Hier finden Sie Rezepte, wie Sie Prozesse und Dienste aus Ihrem Programm heraus verwalten können. Aber auch eine Handvoll Rezepte, die nicht direkt mit dem System zu tun haben, aber doch stark systemlastig sind, sind hier zu finden. So zum Beispiel Rezepte zum Versenden von Fax und eMail.

## 199 WMI-Namensräume

Um zu ermitteln, welche WMI-Namensräume auf dem Rechner zur Verfügung stehen, muss im Programm als Erstes der entsprechende .NET-Namensraum importiert werden.

```
Imports System.Management
```

Da in diesem Bereich mit Namen gearbeitet wird, empfiehlt sich eine Funktion, die alle Namensräume als Liste zurückliefert.

```
Public Function GetNameSpaces() As ArrayList
    Dim mScope As ManagementScope = New ManagementScope("root")
    Dim mPath As ManagementPath = New ManagementPath("__Namespace")
    Dim mClass As ManagementClass = _
        New ManagementClass(mScope, mPath, Nothing)
    Dim mObject As ManagementObject
    Dim mList As ArrayList = New ArrayList

    ' Schleife über alle Namensräume
    For Each mObject In mClass.GetInstances
        mList.Add(mObject("Name").ToString)
    Next
    Return mList
End Function
```

*Listing 362: Die Funktion GetNameSpaces()*

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Propor-  
tityGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

Zuerst werden drei Variable definiert, die einem in Kontext von WMI immer wieder begegnen können:

Standardmäßig wird der Namensraum »root\cimv2« vorgegeben. Da wir aber alle Namensräume betrachten wollen, muss WMI mitgeteilt werden, dass wir diesen Standard-Namensraum nicht nutzen wollen, sondern den Namensraum »root«. Dies geschieht mit der Klasse `ManagementScope`. Bei der Definition wird der zu betrachtende Namensraum angegeben. Mit `ManagementPath` wird anschließend innerhalb dieses Namensraumes der Weg zur Basisklasse angegeben. Der Pfad »\_\_namespace« besagt, dass wir uns mit Namensräumen beschäftigen wollen. Hat man beides definiert, kann man die eigentlich gewünschte Klasse mit `ManagementClass` definieren. Wer sich diese drei Klassen in der Online-Hilfe anschaut, wird feststellen, dass es zahlreiche Überladungen dieser Klassen gibt. Es existieren auch Varianten, in denen nur ein Objekt der Klasse `ManagementClass` benutzt wird, um mit dem WMI-Provider in Verbindung zu treten. Im Interesse der späteren Erweiterungen (z.B. auf mehrere Rechner) sollte man sich aber diesen Dreisprung zur Gewohnheit machen.

Um nun auf eine konkrete Instanz einer Klasse zugreifen zu können, wird eine Variable vom Typ `ManagementObject` benötigt.

Viele Eigenschaften von WMI-Objekten kommen als Liste daher, auch wenn es nur einen sinnvollen Wert geben sollte. Diese Listen können wie gewohnt durchlaufen werden. In der Schleife von Listing 363 wird dabei auf die Eigenschaft »Name« des jeweiligen Objektes zugegriffen. Hierüber erhält man den Namen des jeweiligen Namensraumes, der einer `ArrayList` angehängt wird.

## 200 WMI-Klassen

Hat man die Namensräume ermittelt, interessieren die darin enthaltenen Klassen. Ein anderer Fall kann sein, dass man von einem bestimmten Rechner die Klassen eines Namensraumes ermitteln möchte. Diese können von Rechner zu Rechner unterschiedlich vorhanden sein. Die Funktion `GetClasses` liefert diese Information:

---

```
Public Function GetClasses(ByVal NameSpaceString As String) As ArrayList
    Dim mScope As ManagementScope = _
        New ManagementScope("ROOT\" + NameSpaceString)
    Dim mQuery As WqlObjectQuery
    Dim mSearchString As String
    Dim mSearch As ManagementObjectSearcher
    Dim mClass As ManagementClass
    Dim mList As ArrayList = New ArrayList

    mSearchString = "Select * from meta_Class"
    mQuery = New WqlObjectQuery(mSearchString)

    mSearch = New ManagementObjectSearcher(mScope, mQuery, Nothing)

    For Each mClass In mSearch.Get
        mList.Add(mClass("__CLASS").ToString)
    Next
    Return mList
End Function
```

---

Listing 363: Die Funktion `GetClasses()`



Der Funktion wird der Namensraum als Parameter übergeben und ein entsprechendes `ManagementScope`-Objekt erzeugt. In diesem Rezept wird eine WQL-Abfrage zur Ermittlung der entsprechenden Werte eingesetzt. Der Einfachheit halber wird eine Abfrage benutzt, die alle Spalten einer Tabelle zurückliefert. Wie in SQL können die gewünschten Eigenschaften als Spalten in der Abfrage angegeben werden. Der dritte Parameter im Konstruktor von `ManagementObjectSearcher` ist eine Instanz der Klasse `EnumerationOptions`. Mittels dieses Parameters können zum Beispiel Angaben zum Timeout der Abfrage gemacht oder abgefragt werden. Ein weiterer Ansatzpunkt ist die Festlegung, ob die Ergebnisse für eine Auflistung in einem Rutsch oder für jede Position einzeln ermittelt werden sollen. Letzteres bedingt einen erhöhten Netzwerkverkehr, sollte man die Eigenschaften eines anderen Rechners abfragen. Bei der Eigenschaft `__CLASS` handelt es sich um eine so genannte Meta-Eigenschaft von WMI. Sie liefert nicht nur die Eigenschaften der aktuellen Klasse, hier der Namespace-Klasse, sondern auch die Eigenschaften der davon abgeleiteten Klassen.

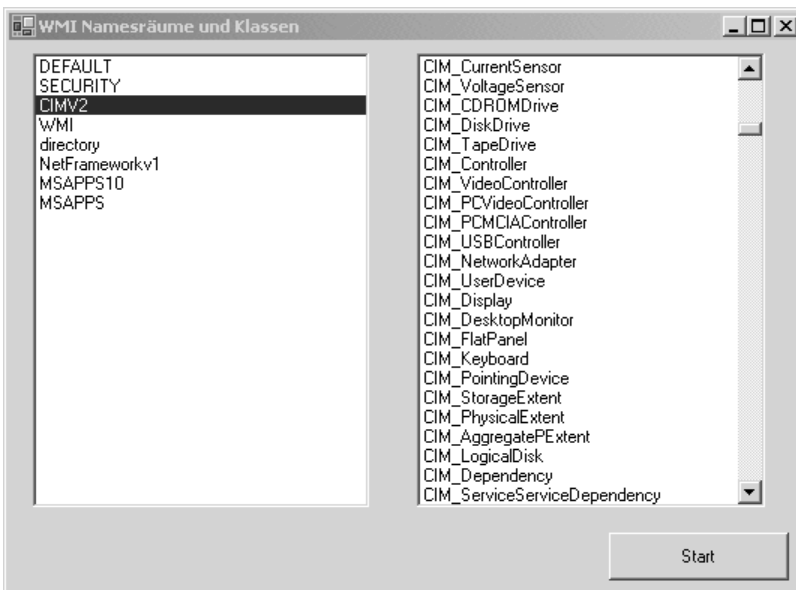


Abbildung 195: Anwendung von `GetClasses()` / `GetNameSpaces()`

In Abbildung 195 ist das Abfrageergebnis für einen Rechner mit installiertem Windows 2000 zu sehen. Auf der linken Seite sind die Namensräume, auf der rechten die dazu gehörenden Klassen aufgelistet. Am Rollbalken der rechten Auflistung kann man den Umfang der Klassen für den Standardnamensraum »root\cimv2« erahnen.

## 201 Ist WMI installiert?

WMI wird erst ab Windows 2000 standardmäßig mit dem Betriebssystem installiert. Für ältere Betriebssysteme kann WMI nachinstalliert werden – sofern es eine Version für dieses Betriebssystem gibt. Naturgemäß ist nicht damit zu rechnen, dass es eine Version für Windows 3.11 gibt oder geben wird. Dies liegt an den grundlegenden Eigenschaften der alten Betriebssysteme. Das Betriebssystem sollte schon COM unterstützen. Es wird ab Windows 9x unterstützt, wobei die Netzwerkfunktionalität erst ab NT möglich ist. Für diese Systeme kann man sich das entsprechende Setup-Programm von den Microsoft-Web-Seiten downloaden.

Will man feststellen, ob nun WMI auf dem zu betrachtenden Rechner installiert ist, kann man in der Registry nach einem entsprechenden Eintrag suchen. Dies erledigt die Funktion `HasWMI()` in Listing 363.

---

```
Public Function HasWMI() As Boolean
    Dim WMIKey As RegistryKey

    WMIKey = Registry.LocalMachine.OpenSubKey("Software\Microsoft\Wbem")
    If WMIKey.Equals(Nothing) Then
        Return False
    Else
        Return True
    End If

End Function
```

---

*Listing 364: Die Funktion HasWMI*

Ist WMI auf dem abzufragenden Rechner installiert, existiert der Schlüssel `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\WBEM`. Sollte es ihn nicht geben, ist entweder WMI nicht installiert oder die Installation sollte nochmals durchgeführt werden.

## 202 BIOS-Informationen

Es gibt Situationen oder Programmvorgaben, die Informationen aus dem BIOS des Rechners erfordern. Auch hier kann man auf WMI zurückgreifen und sich die entsprechenden Angaben auslesen. Die Funktion `GetBIOS()` führt genau dies an einigen BIOS-Werten durch. Hierfür wird die WMI-Klasse `Win32_Bios` ausgenutzt.

---

```
Public Function GetBIOS() As ArrayList
    Dim mQuery As WqlObjectQuery
    Dim mSearch As ManagementObjectSearcher
    Dim mCol As ManagementObject
    Dim mStrSQL As String
    Dim Liste As ArrayList = New ArrayList
    Dim BiosChar As UInt16()

    mStrSQL = "Select * from win32_bios"
    mQuery = New WqlObjectQuery(mStrSQL)
    mSearch = New ManagementObjectSearcher(mQuery)

    For Each mCol In mSearch.Get()
        If mCol("Name") = Nothing Then
            Liste.Add("Name = N/A")
        Else
            Liste.Add("Name = " + mCol("Name"))
        End If

        If mCol("Version") = Nothing Then
            Liste.Add("Version = N/A")
        End If
    Next
```

---

*Listing 365: GetBIOS()-Funktion*

```

Else
    Liste.Add("Version = " + mCol("Version"))
End If

If mCol("BuildNumber") = Nothing Then
    Liste.Add("BuildNumber = N/A")
Else
    Liste.Add("BuildNumber = " + mCol("BuildNumber"))
End If

If mCol("Description") = Nothing Then
    Liste.Add("Beschreibung = N/A")
Else
    Liste.Add("Beschreibung = " + mCol("Description"))
End If

If mCol("SerialNumber") = Nothing Then
    Liste.Add("S/N = N/A")
Else
    Liste.Add("S/N = " + mCol("SerialNumber"))
End If

If mCol("TargetOperatingSystem").ToString = Nothing Then
    Liste.Add("Für Betriebssystem = N/A")
Else
    Liste.Add("Für Betriebssystem = " + _
        Convert.ToString(mCol("TargetOperatingSystem")))
End If

If mCol("ReleaseDate") = Nothing Then
    Liste.Add("Release Datum = N/A")
Else
    Liste.Add("Release Datum = " + mCol("ReleaseDate").ToString)
End If

If mCol("IdentificationCode") = Nothing Then
    Liste.Add("Identifikation = N/A")
Else
    Liste.Add("Identifikation = " + mCol("IdentificationCode"))
End If

If mCol("BiosCharacteristics").Equals(Nothing) Then
    Liste.Add("BIOS Eigenschaften = N/A")
Else
    Dim i As Integer
    Dim Temp As String
    BiosChar = mCol("BiosCharacteristics")
    For i = 0 To BiosChar.GetUpperBound(0) - 1
        Temp += (BiosChar(i).ToString + "; ")
    Next
    Temp += BiosChar(i).ToString

```

Listing 365: GetBIOS()-Funktion (Forts.)

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Propert-  
yGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

---

```

        Liste.Add("BIOS Eigenschaften = " + Temp)
    End If
Next

Return Liste
End Function

```

---

*Listing 365: GetBIOS()-Funktion (Forts.)*

In diesem Rezept wird nicht der »Dreisprung« benutzt, um an die Klasseninstanz zu gelangen. Da es mehrere Wege innerhalb WMI gibt, um an die benötigten Informationen zu gelangen, wird hier der Weg über das Abfrage-Objekt gewählt. Mit einer Instanz der Klasse `WqlObjectQuery` wird die WQL-Abfrage generiert, die dann dem Konstruktor der Klasse `ManagementObjectSearcher` übergeben wird. Damit stehen die Informationen zur Verfügung und können mit einer Schleife über alle BIOS-Versionen eines Rechners ausgelesen werden. Zurzeit gibt es im Intel-PC-Bereich allerdings nur Rechner, die mit einem BIOS (soweit uns bekannt) starten. Die Hardware-Hersteller müssen also noch aufrüsten, um dieses Feature zu unterstützen J.

Nicht jedes BIOS kann Informationen zu den einzelnen Feldern dieser Abfrage liefern. Daher wird mit dem Test auf `Nothing` geprüft, ob die entsprechenden Werte überhaupt vorhanden sind. Die hier vorgestellten BIOS-Werte stehen repräsentativ für eine umfangreiche Liste weiterer Werte. Diese können in der Online – Hilfe unter dem Stichwort »Win32\_Bios« nachgeschlagen werden.

Ein Wert erfordert aber noch ein wenig Beachtung: `BiosChar`. Er ist als Array von `UInt16`-Werten definiert. Dieses Array enthält eine Aufzählung der Features, die das BIOS unterstützt. Hätte das Array beispielsweise nur zwei Einträge `BiosChar(0) = 4` und `BiosChar(1) = 7`, so würde das BIOS nur die Schnittstellen ISA und PCI unterstützen. Würde das BIOS nur PCI unterstützen, so wäre `BiosChar(0) = 7`. Eine Auflistung der gültigen Schlüssel sind in Listing 367 als Enum codiert.

---

```

Enum BIOS_CHAR
    NotSupported = 3
    ISA
    MCA
    EISA
    PCI
    PCMCIA
    PlugPlay
    APM
    BIOSUpgradable
    BIOSshadowing
    VLVEsa
    ESCD
    BootCD
    SelectableBoot
    BIOSsocketed
    BootPCMCIA
    EDD
    NEC_9800_1_2mb
    Toshiba_1_2mb

```

---

*Listing 366: Enumeration für GetBIOS*

```

KB360Floppy
MB1_2Floppy
KB720Floppy
MB2_88Floppy
PrintScreenService
Keyboard8042
SerialServices
PrinterServices
CGA_Mono_Video
NEC_PC98
ACPI
USB
AGP
I20_boot
LS_120_boot
ZIP_Drive_boot
boot_1394
Smart_Battery
End Enum

```

Listing 366: Enumeration für GetBIOS (Forts.)

Da die Inhalte von `BiosCharacteristics` etwas unhandlich sind – entweder muss man die oben gezeigte Komma-separierte Zeichenkette aufsplitten und die einzelnen Werte abfragen oder ein `UInt16`-Array durchtesten –, empfiehlt sich die Bereitstellung einer entsprechenden Funktion `GetBIOSCharacteristics()` mit einem Parameter vom Typ `BIOS_CHAR`. Diese Funktion ist in Listing 368 zu sehen.

```

Public Function GetBIOSCharacteristics(ByVal ToCheck As BIOS_CHAR) _
    As Boolean

    Dim mQuery As WqlObjectQuery
    Dim mSearch As ManagementObjectSearcher
    Dim mCol As ManagementObject
    Dim mStrSQL As String
    Dim BiosChar As UInt16()
    Dim Result As Boolean = False
    Dim QueryResult As UInt16

    ' Abfrage für nur einen Parameter
    mStrSQL = "Select BiosCharacteristics from win32_bios"
    mQuery = New WqlObjectQuery(mStrSQL)
    mSearch = New ManagementObjectSearcher(mQuery)

    ' Get() liefert immer eine Collection
    For Each mCol In mSearch.Get
        If mCol("BiosCharacteristics").Equals(Nothing) Then
            Result = False
        Else
            Dim i As Integer

```

Listing 367: `GetBIOSCharacteristics()`-Funktion

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Proper-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

---

```

    Dim Temp As String
    BiosChar = mCol("BiosCharacteristics")
    For i = 0 To BiosChar.GetUpperBound(0) - 1
        ' Andere If-Konstruktionen lieferten kein
        ' richtiges Ergebnis
        QueryResult = BiosChar(i)
        If Convert.ToInt32(QueryResult) = ToCheck Then
            Result = True
        End If
    Next
End If
Next

Return Result
End Function

```

---

*Listing 367: GetBIOSCharacteristics()-Funktion (Forts.)*

In dieser Funktion wird analog zu der schon gezeigten Funktion aus Listing 366 die Information über eine WQL-Abfrage ermittelt. In diesem Fall werden aber nicht alle Spalten der Pseudotabelle Win32\_Bios abgefragt, sondern nur die uns interessierende: »select BiosCharacteristics from win32\_bios«.

Da die Methode Get() immer nur eine Liste von Werten zurückliefert, muss auch hier eine Schleife über diese Liste gebildet werden. Die Eigenschaft BiosCharacteristics sollte zwar immer vorhanden sein, aber Vorsicht ist in diesem Bereich immer geboten. Daher wird getestet, ob es überhaupt ein zu testendes Ergebnis gibt. Kann die Eigenschaft nicht gefunden werden, wird False zurückgeliefert. Das Ergebnis entspricht also der Aussage: »Die abgefragte Eigenschaft ist nicht vorhanden«.

Testet man diese Funktionen auf einem Laptop, so erhält man die Ergebnisse aus Abbildung 196.

Wie man erkennt, ist das Datum in einem recht eigenwilligen Format angegeben worden. Diese Form der Zeitdarstellung nennt sich DMTF-Format und ist ein spezielles Datumsformat, auf das sich die Desktop Management Task Force geeinigt hat (s.o.).

Um dieses Zeitformat in ein für Anwendungsprogrammierer nutzbares Format zu konvertieren kann man lange suchen oder sich der Hilfe des Tools MgmtClassGen bedienen. Öffnet man ein Kommandofenster und führt den Befehl

```
mgmtclassgen win32_bios /L VB /P C:\temp\win32_bios.txt
```

aus, so findet man in der Datei win32\_bios.txt die Funktion

```
Shared Function ToDateTime(ByVal dmtfDate As String) As Date
```

die diese Umrechnung durchführt. Jede auf diese Art und Weise generierte Datei aus dem Win32\_-Klassenraum enthält eine solche Funktion, sobald in der entsprechenden Klasse mit Datum und Zeit operiert wird. Die umgekehrte Konvertierung kann man mittels der Funktion

```
Shared Function ToDmtfDateTime(ByVal [date] As Date) As String
```

durchführen.

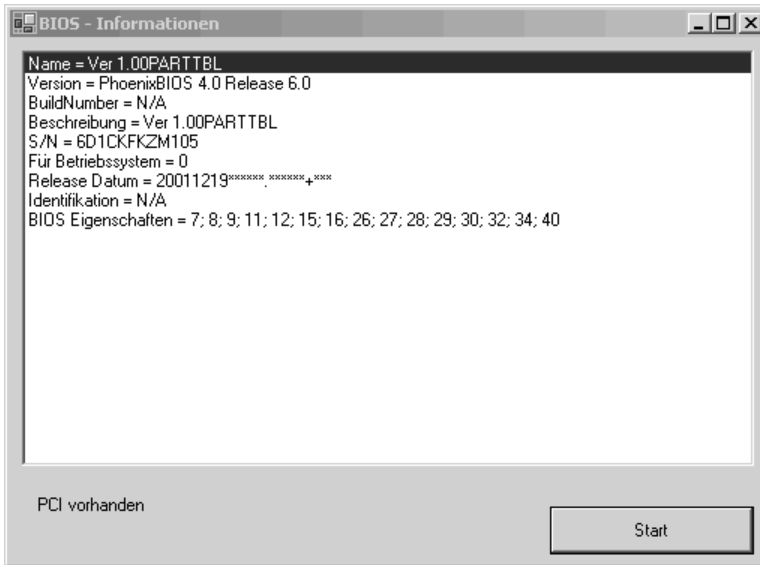


Abbildung 196: BIOS-Information aus Win32\_Bios

## 203 Computer-Modell

Will man zum Beispiel für eine Bestandsliste der eingesetzten Hardware, respektive zur Kontrolle, welche Hardware sich denn hinter welchem Rechner versteckt, die Bezeichnung des Computer-Modells haben, kann man ebenfalls WMI einsetzen.

```
Public Function GetSystemModel() As String
    Dim mQuery As WqlObjectQuery
    Dim mSearch As ManagementObjectSearcher
    Dim mCol As ManagementObject
    Dim mStrSQL As String
    Dim mStr As String

    mStrSQL = "Select * from Win32_ComputerSystem"
    mQuery = New WqlObjectQuery(mStrSQL)
    mSearch = New ManagementObjectSearcher(mQuery)
    For Each mCol In mSearch.Get()
        mStr = mCol("Manufacturer") + " "
        mStr += mCol("Model").ToString
    Next
    Return mStr
End Function
```

Listing 368: GetSystemModel()-Funktion

Auch in dieser Funktion wird über eine WQL-Abfrage die benötigte Information abgerufen. Will man die Menge an Information begrenzen, die von dieser Abfrage geliefert wird (zum Beispiel auf Grund der höheren Netzlast), so kann man die WQL-Abfrage auf *select Manufacturer from Win32\_ComputerSystem* beschränken.

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Proper-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

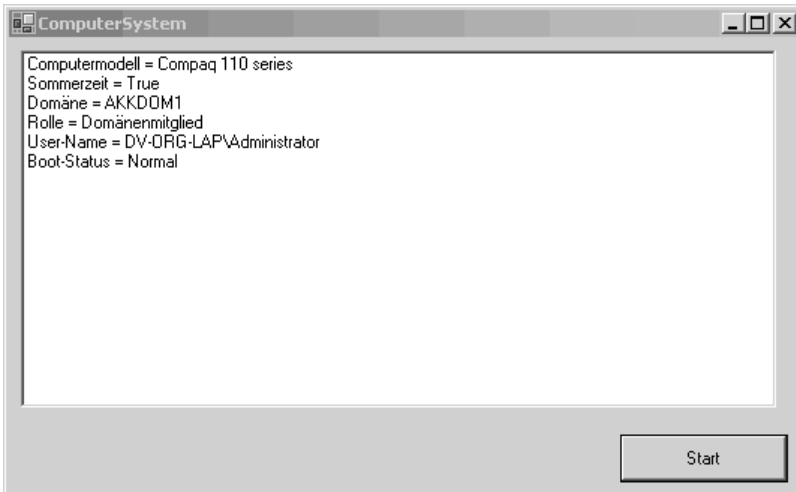


Abbildung 197: Ergebnisse für Kasse Win32\_Computersystem

Das Ergebnis einer Abfrage mit dieser Funktion ist in der ersten Zeile von Abbildung 197 zu sehen.

## 204 Letzter Boot-Status

Interessiert man sich für das Ergebnis des letzten Boot-Vorganges, so kann man diese Information ebenfalls über WMI abfragen. Die hierfür notwendige Funktion ist in Listing 369 zu sehen.

---

```
Public Function GetBootUpState() As String
    Dim mQuery As WqlObjectQuery
    Dim mSearch As ManagementObjectSearcher
    Dim mCol As ManagementObject
    Dim mStrSQL As String
    Dim mStr As String

    mStrSQL = "Select * from Win32_ComputerSystem"
    mQuery = New WqlObjectQuery(mStrSQL)
    mSearch = New ManagementObjectSearcher(mQuery)
    For Each mCol In mSearch.Get()
        mStr = mCol("BootupState")
    Next
    Return mStr
End Function
```

---

Listing 369: Die Funktion GetBootUpState()

Das Ergebnis dieser Funktion kann zurzeit drei Werte annehmen, die als Zeichenkette zurückgeliefert werden, wie sie in Tabelle 29 aufgeführt sind.



| Ergebnisse von GetBootUpState |
|-------------------------------|
| Normal                        |
| Fail-safe                     |
| Fail-safe with network        |

Tabelle 29: GetBootUpState-Ergebnisse

## 205 Sommer-/Winterzeit

Wenn Sie sich davon überzeugen wollen, ob zurzeit die Sommerzeit auf einem Rechner aktiv geschaltet ist, können Sie dies mit der Funktion `GetDayLight()` erreichen.

```
Public Function GetDayLight() As Boolean
    Dim mQuery As WqlObjectQuery
    Dim mSearch As ManagementObjectSearcher
    Dim mCol As ManagementObject
    Dim mStrSQL As String
    Dim mErg As Boolean

    mStrSQL = "Select * from Win32_ComputerSystem"
    mQuery = New WqlObjectQuery(mStrSQL)
    mSearch = New ManagementObjectSearcher(mQuery)
    For Each mCol In mSearch.Get()
        mErg = mCol("DaylightInEffect")
    Next
    Return mErg
End Function
```

Listing 370: Die Funktion `GetDayLight()`

Liefert die Funktion `True` zurück, so ist die Sommerzeit aktiv geschaltet. Ein Beispiel sehen Sie in Abbildung 197 in der untersten Zeile.

## 206 Computerdomäne

Auch die Domäne, in der der entsprechende Computer integriert ist, kann über `Win32_ComputerSystem` ermittelt werden. Die entsprechende Funktion `GetDomain` finden Sie im Listing 371.

```
Public Function GetDomain() As String
    Dim mQuery As WqlObjectQuery
    Dim mSearch As ManagementObjectSearcher
    Dim mCol As ManagementObject
    Dim mStrSQL As String
    Dim mStr As String

    mStrSQL = "Select * from Win32_ComputerSystem"
    mQuery = New WqlObjectQuery(mStrSQL)
```

Listing 371: Die Funktion `GetDomain()`

---

```

mSearch = New ManagementObjectSearcher(mQuery)
For Each mCol In mSearch.Get()
    mStr = mCol("Domain")
Next
Return mStr
End Function

```

---

*Listing 371: Die Funktion GetDomain() (Forts.)*

Auch hier finden Sie ein Beispielergebnis in Abbildung 197.

## 207 Domänenrolle

Die Funktion des Rechners innerhalb einer Domäne – handelt es sich um ein Domänenmitglied oder ist es eventuell sogar der Primäre Domänenkontroller, kann mit der Funktion `GetDomainRole()` ermittelt werden. Die Eigenschaft `DomainRole` der `Win32_ComputerSystem`-Klasse liefert einen `Uint16`-Wert zurück, der über diese »Rolle im Spiel der Computer« Rückschlüsse zulässt. Nun gibt es diverse Einschränkungen, will man diesen `Uint16`-Wert weiter verarbeiten. Diese Einschränkungen basieren auf dem Typ `Uint16`. Also muss man ihn in einen Typ konvertieren, der eine vernünftige Weiterverarbeitung zulässt. So kann beispielsweise die `Select Case`-Anweisung keinen `Uint16`-Wert auswerten. Die Typkonvertierung nach `Int32` ist allerdings nicht CLS-konform. Da die WQL-Abfrage nur Werte zwischen 0 und 5 zurückliefert, kann man dies an dieser Stelle aber vertreten.

---

```

Public Function GetDomainRole() As String
    Dim mQuery As WqlObjectQuery
    Dim mSearch As ManagementObjectSearcher
    Dim mCol As ManagementObject
    Dim mStrSQL As String
    Dim mErg As Integer
    Dim mStr As String

    mStrSQL = "Select * from Win32_ComputerSystem"
    mQuery = New WqlObjectQuery(mStrSQL)
    mSearch = New ManagementObjectSearcher(mQuery)

    For Each mCol In mSearch.Get()
        ' nicht CLS-kompatibel
        mErg = Convert.ToInt32(mCol("DomainRole"))
    Next

    Select Case mErg
        Case 0
            mStr = "Einzelplatz"
        Case 1
            mStr = "Domänenmitglied"
        Case 2
            mStr = "Server"
        Case 3
            mStr = "Server In Domäne"
    End Select
End Function

```

---

*Listing 372: Die Funktion GetDomainRole()*

```
Case 4
    mStr = "Backup Domänen Controller"
Case 5
    mStr = "Primärer Domänen Controller"
End Select

Return mStr
End Function
```

Listing 372: Die Funktion GetDomainRole() (Forts.)

Von der Funktion wird eine dem WQL-Wert entsprechende Klartext-Zeichenkette zurückgegeben. Eine Anwendung ist in Abbildung 197 zu sehen.

## 208 Benutzername

Wollten Sie nicht schon immer mal wissen, wer mit Ihrem Programm arbeitet? Dieses Feature kann man aber auch dazu nutzen, in Bildschirmmasken den Benutzernamen vorzublenken, so dass die Eingabe für den Benutzer erleichtert wird. Zu beachten ist, dass der Benutzername mit dem Domänennamen/Rechnernamen des entsprechenden Accounts geliefert wird. Auf diese Weise kann man auch erkennen, ob in einer Domäne jemand lokal angemeldet ist.

```
Public Function GetUserName() As String
    Dim mQuery As WqlObjectQuery
    Dim mSearch As ManagementObjectSearcher
    Dim mCol As ManagementObject
    Dim mStrSQL As String
    Dim mStr As String

    mStrSQL = "Select * from Win32_ComputerSystem"
    mQuery = New WqlObjectQuery(mStrSQL)
    mSearch = New ManagementObjectSearcher(mQuery)

    For Each mCol In mSearch.Get()
        mStr = mCol("UserName")
    Next

    Return mStr
End Function
```

Listing 373: Die Funktion GetUserName()

Auch hier kann man in Abbildung 197 das Ergebnis dieser Funktion sehen. In diesem Beispiel ist zu erkennen, dass der lokale Administrator angemeldet ist.

Die Klasse Win32\_Computersystem ist sehr umfangreich. Die einzelnen Eigenschaften der Klasse kann man sich in der Online-Hilfe anschauen. Zum Vergleich und zum Verständnis ist diese Klasse in Listing 374 in VB.NET-Nomenklatur zu sehen.

---

```
Public Class Win32_ComputerSystem
    Inherits CIM_UnitaryComputerSystem

    Dim AdminPasswordStatus As UInt16
    Dim AutomaticResetBootOption As Boolean
    Dim AutomaticResetCapability As Boolean
    Dim BootOptionOnLimit As UInt16
    Dim BootOptionOnWatchDog As UInt16
    Dim BootROMSupported As Boolean
    Dim BootupState As String
    Dim Caption As String
    Dim ChassisBootupState As UInt16
    Dim CreationClassName As String
    Dim CurrentTimeZone As Int16
    Dim DaylightInEffect As Boolean
    Dim Description As String
    Dim DNSHostName As String
    Dim Domain As String
    Dim DomainRole As UInt16
    Dim EnableDaylightSavingsTime As Boolean
    Dim FrontPanelResetStatus As UInt16
    Dim InfraredSupported As Boolean
    Dim InitialLoadInfo As String
    Dim InstallDate As DateTime
    Dim KeyboardPasswordStatus As UInt16
    Dim LastLoadInfo As String
    Dim Manufacturer As String
    Dim Model As String
    Dim Name As String
    Dim NameFormat As String
    Dim NetworkServerModeEnabled As Boolean
    Dim NumberOfProcessors As UInt32
    Dim OEMLogoBitmap() As Byte
    Dim OEMStringArray() As String
    Dim PartOfDomain As Boolean
    Dim PauseAfterReset As Int64
    Dim PowerManagementCapabilities() As UInt16
    Dim PowerManagementSupported As Boolean
    Dim PowerOnPasswordStatus As UInt16
    Dim PowerState As UInt16
    Dim PowerSupplyState As UInt16
    Dim PrimaryOwnerContact As String
    Dim PrimaryOwnerName As String
    Dim ResetCapability As UInt16
    Dim ResetCount As Int16
    Dim ResetLimit As Int16
    Dim Roles() As String
    Dim Status As String
    Dim SupportContactDescription() As String
    Dim SystemStartupDelay As UInt16
    Dim SystemStartupOptions() As String
```

---

```

Dim SystemStartupSetting As Byte
Dim SystemType As String
Dim ThermalState As UInt16
Dim TotalPhysicalMemory As UInt64
Dim UserName As String
Dim WakeUpType As UInt16
Dim Workgroup As String
End Class

```

Listing 374: Die Pseudoklasse Win32\_ComputerSystem (Forts.)

Die in Listing 374 gezeigte Pseudo-Klasse existiert so nicht im WMI. Sie kann aber als Übersicht über diese Klasse sehr gut genutzt werden.

## 209 Monitorauflösung

Für bestimmte Anwendungen ist es von Vorteil, wenn man die aktuelle Auflösung des Monitors kennt. Dies per Programm abzufragen, ist mittels der Klasse Win32\_DesktopMonitor kein großes Problem mehr. Die entsprechende Funktion GetResolution ist in Listing 375 zu sehen. Mit den Eigenschaften ScreenWidth und ScreenHeight wird die klassische Auflösungsangabe Breite x Höhe zusammengebaut.

```

Public Function GetResolution() As String
    Dim mQuery As WqlObjectQuery
    Dim mSearch As ManagementObjectSearcher
    Dim mCol As ManagementObject
    Dim mStrSQL As String
    Dim mErg As Integer
    Dim mStr As String

    mStrSQL = "Select * from Win32_DesktopMonitor"
    mQuery = New WqlObjectQuery(mStrSQL)
    mSearch = New ManagementObjectSearcher(mQuery)
    For Each mCol In mSearch.Get()
        mStr = Convert.ToString(mCol("ScreenWidth"))
        mStr += " x "
        mStr += Convert.ToString(mCol("ScreenHeight"))
    Next
    Return mStr
End Function

```

Listing 375: Die Funktion GetResolution()

Ein Ergebnis dieser Abfrage können Sie ebenfalls in Abbildung 198 sehen.

## 210 Der Monitor-Typ

Bei neueren Bildschirmen kann man zusätzliche Informationen über den angeschlossenen Monitor ermitteln. So kann man unter anderem auch den Typ des Monitors herausfinden. Wie man dies mittels WMI durchführt, ist in Listing 375 zu sehen. Da der Monitortyp über eine Schleife ermittelt wird, können auf diese Weise alle an diesem PC angeschlossenen Monitore ermittelt werden.

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tung  
Win-  
dows  
Forms

Controls

Proper-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

---

```

Public Function GetMonitorType() As String
    Dim mQuery As WqlObjectQuery
    Dim mSearch As ManagementObjectSearcher
    Dim mCol As ManagementObject
    Dim mStrSQL As String
    Dim mStr As String

    mStrSQL = "Select * from Win32_DesktopMonitor"
    mQuery = New WqlObjectQuery(mStrSQL)
    mSearch = New ManagementObjectSearcher(mQuery)

    For Each mCol In mSearch.Get()
        mStr = mCol("MonitorType")
    Next

    Return mStr
End Function

```

---

Listing 376: Die Funktion *GetMonitorType()*

Nach der Erstellung der bekannten Objekte wird die Eigenschaft *MonitorType* für den Monitor abgefragt und dem aufrufenden Programm zurückgeliefert. Ist nichts Besonderes bekannt, wird *Default Monitor* zurückgeliefert. Auch hier ist als Beispiel die Abbildung 198 zu betrachten.

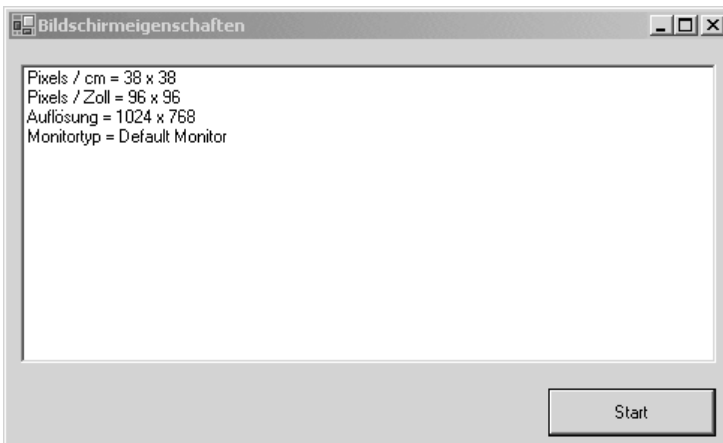


Abbildung 198: Monitoreigenschaften mit WMI

## 211 Auflösung in Zoll

In einigen Fällen ist es nicht nur wichtig, die Auflösung in Pixel, sondern auch die Skalierung in »Pixel / Zoll« oder »Pixel / cm« in Erfahrung zu bringen. Die erste Skalierung lässt sich mit WMI ermitteln. Die Umrechnung in »Pixel / cm« kann dann durch die Division mit 2,54 sehr leicht berechnet werden. Die nachfolgende Funktion *GetLogicalPixels()* erledigt diese Arbeit.

```

Public Function GetLogicalPixels(Optional ByVal cm As Boolean _
    = False) As String

    Dim mQuery As WqlObjectQuery
    Dim mSearch As ManagementObjectSearcher
    Dim mCol As ManagementObject
    Dim mStrSQL As String
    Dim mStr As String

    mStrSQL = "Select * from Win32_DesktopMonitor"
    mQuery = New WqlObjectQuery(mStrSQL)
    mSearch = New ManagementObjectSearcher(mQuery)

    For Each mCol In mSearch.Get()
        Dim Band As UInt32
        Band = mCol("PixelsPerXLogicalInch")

        If cm = True Then
            Band = Convert.ToUInt32(Convert.ToDouble(Band) / 2.54)
        End If

        mStr = Band.ToString
        Band = mCol("PixelsPerYLogicalInch")

        If cm = True Then
            Band = Convert.ToUInt32(Convert.ToDouble(Band) / 2.54)
        End If
        mStr += (" x " + Band.ToString)
    Next

    Return mStr
End Function

```

Listing 377: Die Funktion *GetLogicalPixels()*

Der Funktion wird ein logischer Parameter übergeben, der im Zustand `True` dafür sorgt, dass die Angaben in »Pixel/cm« zurückgegeben werden. Da in vielen Fällen die Angabe in Zoll relevant ist, wird der Parameter per Default-Wert auf diese Einstellung gesetzt.

Da eine automatische Konvertierung von `UInt32` nach `Double` und auch von `Double` nach `UInt32` nicht möglich ist (auch nicht bei `strict off`), wird diese durch den Aufruf der Shared Klasse `Convert` aus dem `System`-Namensraum erzwungen. Dies ist hier ohne Probleme auf Grund der zu erwartenden Zahlenwerte möglich. In solchen Fällen sollte man sich genau überlegen, was man da konvertiert und wie die möglichen Zahlenwerte sein können. So mancher sehr schwer zu findende Fehler basiert auf falschen Konvertierungen. In Abbildung 198 kann man das Ergebnis für beide Varianten sehen.

## 212 Logische Laufwerke mit WMI

Um sich Informationen über ein Laufwerk zu holen, kann man ebenfalls WMI einsetzen. Um die Programmierung etwas abwechslungsreicher zu machen, wird hier eine weitere Methode zur Informationsgewinnung benutzt.

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Proper-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

Öffnet man nach der Installation der WMI-Erweiterung im Server-Explorer (siehe Anhang) den Menübaum von *Management Classes*, so erscheint eine Auflistung aller möglichen WMI-Klassen. Mit der rechten Maustaste über einem Eintrag kann ein Kontextmenü geöffnet werden, welches den Eintrag *Generate Managed Class* enthält. Dieser Eintrag ist auch in der deutschen Version auf Englisch, da es diese Erweiterung nur in englischer Sprache gibt. Führt man diesen Befehl für den Eintrag *Disk Volumes* aus, erscheint im Projektmappen-Explorer eine neue Datei: *Win32\_LogicalDisk.vb* (siehe Abbildung 199). Diese Datei ist 78 Kilobyte groß, entsprechend umfangreich ist der darin enthaltene Quellcode. Einige Teile werden hier besprochen, da sie allgemein gültig sind und auch für andere Klassen ein nützliches Hintergrundwissen darstellen.

Einen ersten Überblick über die Klassen kann man sich mit der Klassenansicht verschaffen. Auch hier überzeugt die Menge der Methoden und Eigenschaften J.

Zu diesen Methoden und Eigenschaften gibt es keine Erklärungen in Handbüchern oder Hilfetexten. Da aber alles im Quelltext vorliegt, kann man sich die fragliche Methode/Eigenschaft in diesem Quelltext anschauen (und dabei vielleicht etwas lernen).

Ein Problem des klassischen WMI-Zugriffs ist die Frage, ob die WMI-Eigenschaft überhaupt etwas zurück liefert. Einige Eigenschaften liefern immer ein Ergebnis zurück, während andere durchaus auch ein *Nothing* zurückliefern können, was unweigerlich zu einer Ausnahme im Programm führen kann. Hier bieten die auf diese Weise erstellten Klassen ein komfortables Hilfsmittel zur Verfügung. Für jede Eigenschaft oder Methode, die *Nothing* zurückliefern könnte, wird eine Funktion nach dem Schema *Is<Eigenschaft>Null* zur Verfügung gestellt.

---

```
<Browsable(false), _
DesignerSerializationVisibility _
  (DesignerSerializationVisibility.Hidden)> _
Public ReadOnly Property IsFreeSpaceNull As Boolean
Get
    If (curObj("FreeSpace") Is Nothing) Then
        Return True
    Else
        Return false
    End If
End Get
End Property
```

---

*Listing 378: IsFreeSpaceNull der Win32\_LogicalDisk-Klasse*

Als Beispiel sei hier die Funktion *IsFreeSpaceNull()* genannt, mit der getestet werden kann, ob ein entsprechender Wert vorhanden ist oder nicht.

Da man sich in diesem Kontext für ein bestimmtes Laufwerk interessiert, muss man mit dieser Klasse auch keine WQL-Abfrage mit *where*-Klausel generieren. Die Klasse enthält eine entsprechende *New()*-Methode:

---

```
Public Sub New(ByVal keyDeviceID As String)
    Me.New(CType(Nothing, System.Management.ManagementScope), _
    CType(New System.Management.ManagementPath _
    (LogicalDisk.ConstructPath(keyDeviceID)), _
```

---

*Listing 379: Die Methode New(keyDeviceID) der Win32\_LogicalDisk-Klasse*



```

        System.Management.ManagementPath), _
        CType(Nothing, System.Management.ObjectGetOptions))
    End Sub

```

Listing 379: Die Methode `New(keyDeviceID)` der `Win32_LogicalDisk`-Klasse (Forts.)

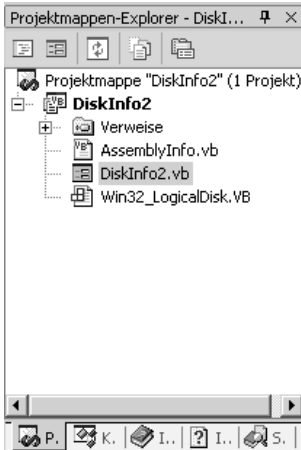


Abbildung 199: Projektmappen-Explorer nach Generierung der WMI-Klasse

Dies ist eine von insgesamt neun New-Methoden, die angeboten werden.

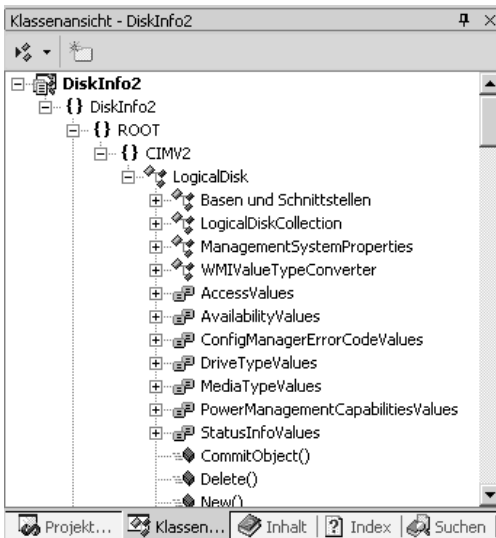


Abbildung 200: Klassenansicht von `Win32_LogicalDisk`

Mit diesem Vorwissen kann die Abfrage nach den Laufwerksinformationen in Angriff genommen werden. Eine Lösung des Problems ist in Listing 381 zu sehen. Die in der Datei `Win32_LogicalDisk.vb` enthaltene Klasse muss importiert werden, da zwar die Datei vom Server-Explorer erstellt und in das Projekt kopiert, aber eine entsprechende `Imports`-Zeile nicht generiert wurde.

---

```
Imports System
Imports System.Management

' strict=On geht mit dem folgenden Import nicht
Imports DiskInfo2.ROOT.CIMV2
```

---

*Listing 380: Imports Anweisungen zur Nutzung WMI – Klasse*

Leider erkaufte man sich durch den Import dieser Klasse einen Nachteil: die Einstellung `strict = on` ist nicht mehr nutzbar, da in der generierten Klasse viele implizite Typkonvertierungen durchgeführt werden.

---

```
Private Sub btnStart_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnStart.Click

    Dim mDiskFreeSpace As String
    Dim mDiskID As String
    Dim mDiskName As String
    Dim mDiskCaption As String
    Dim mPath As String
    Dim mDiskVolumeName As String
    Dim mDriveType As String
    Dim mMediaType As String
    Dim mSize As String
    Dim mSerialNo As String

    mDiskID = txtDisk.Text.ToString

    ' Klassische Variante auch hier möglich
    'mPath = "win32_LogicalDisk.DeviceId=" + "" + mDiskID + ""
    'Dim disk As New LogicalDisk(New ManagementPath(mPath))

    Dim disk As New LogicalDisk(mDiskID)

    If disk.IsFreeSpaceNull = False Then
        mDiskFreeSpace = "Freier Speicherplatz = " + _
            disk.FreeSpace.ToString
    Else
        mDiskFreeSpace = "Freier Speicherplatz = N/A"
    End If

    mDiskName = "Name = " + disk.Name.ToString

    mDiskCaption = "Caption = " + disk.Caption.ToString()

    mDiskVolumeName = "Laufwerksbezeichnung = " + _
        disk.VolumeName.ToString()

    If disk.IsDriveTypeNull = False Then
        mDriveType = "Laufwerkstyp = " + disk.DriveType.ToString()
```

---

*Listing 381: Laufwerksinformationen mittels WMI-Klassen*

```

Else
    mDriveType = "Laufwerkstyp = N/A"
End If

If disk.IsMediaTypeNull = False Then
    mMediaType = "Medientypus = " + disk.MediaType.ToString()
Else
    mMediaType = "Medientypus = N/A"
End If

If disk.IsSizeNull = False Then
    mSize = "Größe = " + disk.Size.ToString()
Else
    mSize = "Größe = N/A"
End If

mSerialNo = "S/N = " + disk.VolumeSerialNumber.ToString

lbList.Items.Add(mDiskName)
lbList.Items.Add(mDiskFreeSpace)
lbList.Items.Add(mDiskCaption)
lbList.Items.Add(mDiskVolumeName)
lbList.Items.Add(mDriveType)
lbList.Items.Add(mMediaType)
lbList.Items.Add(mSize)
lbList.Items.Add(mSerialNo)

End Sub

```

*Listing 381: Laufwerksinformationen mittels WMI-Klassen (Forts.)*

Da die Klasse recht einfach eingesetzt werden kann, wurde auf die Erstellung einer separaten Funktion für Cut and Paste verzichtet und ein Konsolenprogramm erstellt.

Nach der Deklaration einiger Zeichenketten-Variablen wird die Laufwerksbezeichnung aus der Bildschirmmaske entnommen und der `New`-Methode der Klasse `LogicalDisk` übergeben. In den dazwischen liegenden Kommentarseiten ist eine andere Verfahrensweise zur Erstellung der Klasse aufgeführt. Diese entspricht mehr dem klassischen Vorgehen. Im Anschluss daran werden einzelne Eigenschaften der WMI-Klasse abgefragt. Hierbei werden die generierten Funktionen zur `Nothing`-Ermittlung für die Eigenschaften benutzt, die für dieses Problem auftauchen können. Diese Werte werden anschließend einer `TextBox`-übergaben.

Wie man Abbildung 201 entnimmt, wurde der `START`-Button zweimal betätigt. Das erste Mal wurde die lokale Platte C: abgefragt. Zu erkennen ist dies am Laufwerkstyp. Es handelt sich um eine lokale Platte. Bei Laufwerk F: ist an dieser Stelle ein Netzwerklaufwerk vermerkt.

Während `FreeSpace` den noch freien Platz des Laufwerkes in Bytes angibt, zeigt `Size` die Gesamtkapazität des Laufwerkes an.

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Proper-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

## 213 Physikalische Platten

Bei logischen Laufwerken handelt es sich im Prinzip um Partitionen auf einer eventuell größeren physikalischen Platte, die noch weitere Partitionen (Laufwerke) beinhalten kann. Will man also neben den logischen Laufwerken auch wissen, wie viele und welche Festplatten eingebaut sind, kommt man mit der in 375 gezeigten Methode nicht weiter. Leider bietet auch die WMI-Erweiterung des Server-Explorers keine zu generierende Klasse, so dass man auf eine der klassische Methoden zurückgreifen muss.

---

```
Public Structure PhysicalHardDrive
    Public Name As String
    Public Systemname As String
    Public DeviceID As String
    Public Manufacturer As String
    Public Typ As String
    Public IndexNumber As String
    Public Partitions As String
    Public Model As String
    Public Cylinder As String
    Public Sectors As String
    Public Heads As String
    Public Tracks As String
    Public TracksPerCylinder As String
    Public SectorsPerTrack As String
    Public Size As String
End Structure
```

---

*Listing 382: Struktur für physikalische Festplatten*

Da durchaus mehrere Platten in einem Rechner Platz finden, bietet sich eine Struktur an, die eine Festplatte symbolisiert. Eine Möglichkeit der Definition einer solchen Struktur ist in Listing 382 dargestellt. Es sind alle relevanten Daten einer Festplatte aufgeführt.

Die Angabe von `Size` ist redundant, da sie aus der Multiplikation der Angaben zur Anzahl der Zylinder, der Anzahl der Spuren pro Zylinder, der Anzahl der Sektoren pro Spur und der Anzahl der Bytes pro Sektor ermittelt werden kann und intern durch WMI auch so errechnet wird.

Während die `DeviceID` dieser Struktur eine eindeutige Nummer bezogen auf alle Geräte des Computers darstellt, bezieht sich der `Index` auf eine eindeutige Nummerierung der Festplatten innerhalb des Systems.

Die Strukturgröße `Partitions` gibt die Anzahl der Partitionen der Festplatte an. Dies können logische Festplatten (C:, D:, etc.) sein. Unter den Partitionen können auch solche Partitionen aufgeführt werden, die von anderen Betriebssystemen genutzt werden. So werden zum Beispiel von der in Abbildung 202 gezeigten Platte mit der ID `\\.\PHYSICALDRIVE0` drei der vier Partitionen von Linux genutzt.

USB-Sticks werden von WMI als Festplatte erkannt und mit entsprechenden Werten gemeldet. Während die Kapazitätsberechnung bei den Festplatten einwandfrei ist, errechnet das System für einen 128 MB USB-Stick eine Gesamtkapazität von 123 MB. Woran dies liegt, konnte aber noch nicht festgestellt werden.

Ermittelt werden die Größenangaben durch die Funktion `GetHDInfo()`, wie sie in Listing 383 zu sehen ist.

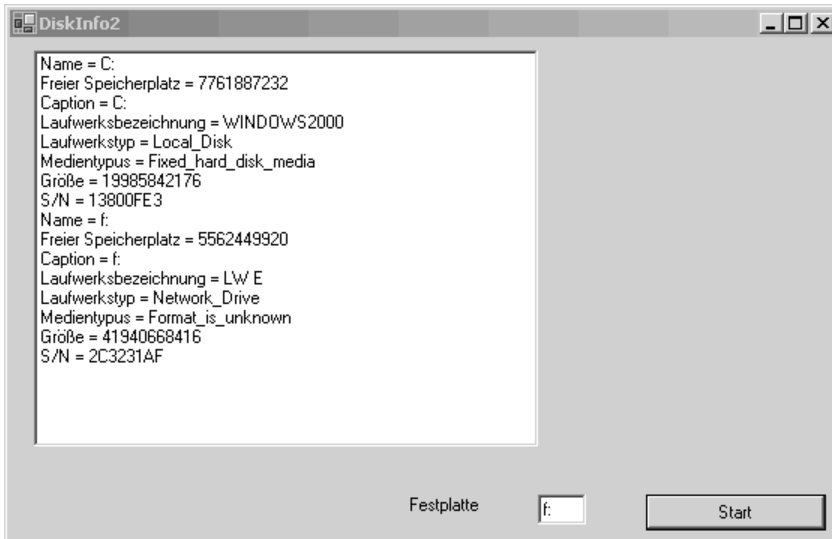


Abbildung 201: Beispielausgabe zur Ermittlung von Laufwerksinformationen

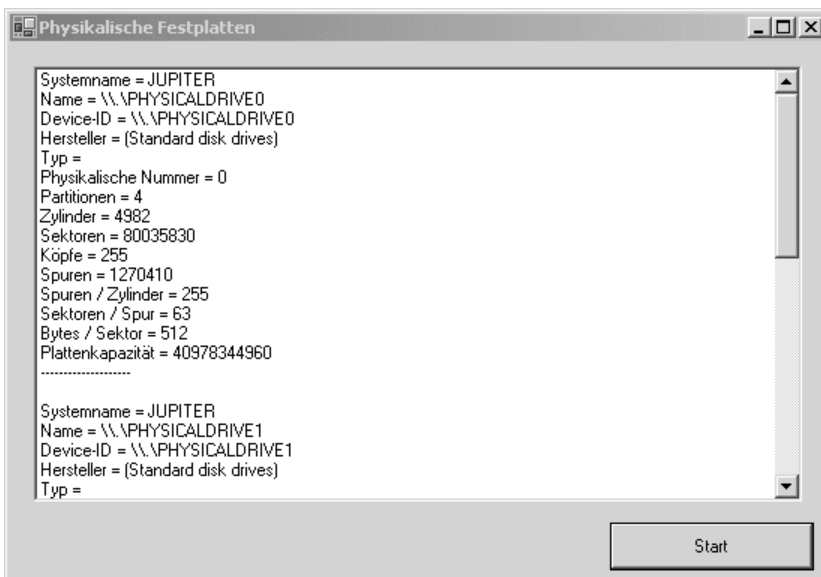


Abbildung 202: Angaben zu physikalischen Platten

```
Public Function GetHDInfo() As ArrayList
    Dim mQuery As WqlObjectQuery
    Dim mSearch As ManagementObjectSearcher
    Dim mCol As ManagementObject
    Dim mStrSQL As String
    Dim HDs As ArrayList = New ArrayList
```

Listing 383: Die Funktion GetHDInfo()

Basics

 Datum/  
Zeit

 Anwen-  
dungen

 Zeich-  
nen

 Bild-  
bearbei-  
tung

 Win-  
dows  
Forms

Controls

 Proper-  
tyGrid

 Datei-  
system

 Netz-  
werk

System

 Daten-  
banken

XML

 Wissen-  
schaft

 Ver-  
schiede-  
nes

---

```

Dim HDInfo As PhysicalHardDrive

mStrSQL = "Select * from win32_DiskDrive"
mQuery = New WqlObjectQuery(mStrSQL)
mSearch = New ManagementObjectSearcher(mQuery)

For Each mCol In mSearch.Get()
    HDInfo.Systemname = mCol("SystemName").ToString
    HDInfo.Name = mCol("Name").ToString
    HDInfo.DeviceID = mCol("DeviceID").ToString
    HDInfo.Manufacturer = mCol("Manufacturer").ToString
    HDInfo.Model = mCol("Model").ToString
    HDInfo.IndexNumber = mCol("Index").ToString
    HDInfo.Partitions = mCol("Partitions").ToString
    HDInfo.Cylinder = mCol("TotalCylinders").ToString
    HDInfo.Sectors = mCol("TotalSectors").ToString
    HDInfo.Heads = mCol("TotalHeads").ToString
    HDInfo.Tracks = mCol("TotalTracks").ToString
    HDInfo.TracksPerCylinder = mCol("TracksPerCylinder").ToString
    HDInfo.SectorsPerTrack = mCol("SectorsPerTrack").ToString
    HDInfo.BytesPerSector = mCol("BytesPerSector").ToString
    HDInfo.Size = mCol("Size").ToString

    HDs.Add(HDInfo)
Next
Return HDs
End Function

```

---

*Listing 383: Die Funktion GetHDInfo() (Forts.)*

Neben den üblichen Deklarationen zur WMI-Verbindung werden zwei zusätzliche Variable definiert: `HDs` für die Rückmeldung an das aufrufende Programm und `HDInfo` zur Speicherung der Informationen für die jeweils betrachtete Festplatte. Nach der erfolgreichen Verbindung zu WMI werden die einzelnen Strukturfelder mit Werten gefüllt und diese Struktur in die `ArrayList` kopiert. Dies wird für jede Festplatte des System durchgeführt. Zum Schluss wird die gefüllte `ArrayList` an das aufrufende Programm zurückgegeben.

Da das Durchlaufen einer `ArrayList` von Strukturen nicht unbedingt jeden Tag vorkommt, ist hier zur Erinnerung auch das aufrufende Programm für das in Abbildung 202 gezeigte Beispiel abgedruckt.

---

```

Private Sub btnStart_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnStart.Click

    ' Struktur deklarieren
    Dim PhysHD As PhysicalHardDrive

    ' ArrayList für diese Struktur
    Dim HDArr As ArrayList

    HDArr = GetHDInfo()

```

---

*Listing 384: Beispielprogramm zum Aufruf von GetHDInfo*

```

For Each PhysHD In HDArr
    lblList.Items.Add("Systemname = " + PhysHD.Systemname)
    lblList.Items.Add("Name = " + PhysHD.Name)
    lblList.Items.Add("Device-ID = " + PhysHD.DeviceID)
    lblList.Items.Add("Hersteller = " + PhysHD.Manufacturer)
    lblList.Items.Add("Typ = " + PhysHD.Type)
    lblList.Items.Add("Physikalische Nummer = " + _
        PhysHD.IndexNumber)
    lblList.Items.Add("Partitionen = " + PhysHD.Partitions)
    lblList.Items.Add("Zylinder = " + PhysHD.Cylinder)
    lblList.Items.Add("Sektoren = " + PhysHD.Sectors)
    lblList.Items.Add("Köpfe = " + PhysHD.Heads)
    lblList.Items.Add("Spuren = " + PhysHD.Tracks)
    lblList.Items.Add("Spuren / Zylinder = " + _
        PhysHD.TracksPerCylinder)
    lblList.Items.Add("Sektoren / Spur = " + _
        PhysHD.SectorsPerTrack)
    lblList.Items.Add("Bytes / Sektor = " + _
        PhysHD.BytesPerSector)
    lblList.Items.Add("Plattenkapazität = " + PhysHD.Size)
    lblList.Items.Add("-----")
    lblList.Items.Add("")
    lblList.Refresh()
Next

End Sub

```

Listing 384: Beispielprogramm zum Aufruf von GetHDInfo (Forts.)

Die lokal definierte `ArrayList HDArr` wird über den Aufruf der Funktion gefüllt. Im Schleifendurchlauf wird die Strukturvariable `PhysHD` jedes Mal neu mit den Werten der aktuellen Platte überschrieben und kann somit für die Ausgabe benutzt werden.

## 214 Installierte Programme

Eine Liste der installierten Programme auf einem Rechner kann auf zwei Arten erzeugt werden: über Einträge in der Registry des betreffenden Rechners und über die WMI-Klasse `Win32_Product`. Kann man über die Registry auch Programme finden, die nicht mit dem Microsoft System Installer (MSI) installiert wurden, findet WMI leider nur solche Programme. Betrachtet man aber die Installation von .NET-Programmen, so tauchen neue Probleme auf, da solche Programme in vielen Fällen nur noch mit der »xcopy-Methode« installiert werden müssen. Programme, die ihre Einstellungen über `.ini`-Dateien absichern, fehlen in den genannten Listen ohnehin. Und es gibt einige größere 32-Bit-Programmsysteme, die mit `.ini`-Dateien arbeiten, da man Textdateien sowohl per Programm als auch per Hand besser bearbeiten kann. Diese Einsicht hat Microsoft wohl auch dazu bewegt, Textdateien zur Konfiguration von Programmen mit .NET wieder einzuführen.

Mit der Kenntnis, nicht alle Programme zu erwischen, kann die Implementierung einer solchen Funktion wie in Listing 387 aussehen. Um die ermittelten Informationen übersichtlicher zu gestalten, werden noch eine Struktur und eine Enumeration eingeführt.

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

```
Public Structure ProductStruc
    Public Name As String
    Public Version As String
    Public InstallDate As String
    Public InstallState As String
    Public SN As String
End Structure
```

---

*Listing 385: Struktur zum Austausch der Produktdaten*

Die Struktur aus Listing 385 ermöglicht den Austausch der Programminformationen über eine einfache `ArrayList`, in der jeder Eintrag eine Produktstruktur darstellt. Das Programm wird dadurch übersichtlicher.

---

```
Public Enum InstState
    Bad = -6
    Invalid = -2
    Unknown = -1
    Advertised = 1
    Absent = 2
    Installed = 5
End Enum
```

---

*Listing 386: Enumeration des Installationsstatus*

Der Status der Installation wird über die Eigenschaft `InstallState` ermittelt, die den Status als vorzeichenbehaftete Integerzahl zurückliefert. Um sowohl das Programm als auch die Ausgabe lesbarer zu gestalten, wird die Enumeration aus Listing 386 eingeführt.

---

```
Public Function GetProd() As ArrayList
    ' Typ Product aus Win32_Product
    Dim pr As New Product

    ' Collection aus Win32_Product
    Dim prEnum As Product.ProductCollection

    Dim mArr As ArrayList = New ArrayList
    Dim str As String

    ' einzelnes Produkt in Struktur speichern
    Dim mProduct As ProductStruc

    ' Sammeln aller Programme in einer Collection
    prEnum = pr.GetInstances

    ' alle mit MSI installierten Programme durchlaufen
    For Each pr In prEnum
        mProduct.Name = pr.Name
        mProduct.Version = pr.Version
        mProduct.InstallDate = pr.InstallDate
```

---

*Listing 387: Die Funktion GetProd()*



```

' Status der Installation über Enum in Klartext umwandeln
Select Case CType(pr.InstallState, InstState)
    Case InstState.Installed
        mProduct.InstallState = "Installed"
    Case InstState.Absent
        mProduct.InstallState = "Absent"
    Case InstState.Advertised
        mProduct.InstallState = "Advertised"
    Case InstState.Bad
        mProduct.InstallState = "Bad"
    Case InstState.Invalid
        mProduct.InstallState = "Invalid"
    Case InstState.Unknown
        mProduct.InstallState = "Unknown"
    Case Else
        mProduct.InstallState = "New Number?"
End Select

mProduct.SN = pr.IdentifyingNumber
mArr.Add(mProduct)
Next
Return mArr
End Function

```

Listing 387: Die Funktion GetProd() (Forts.)

Vor der ersten selbst programmierten Zeile wird aus dem Server-Explorer eine .Net-Klasse aus der WMI-Klasse *Software Products (WMI)* generiert. Im Projektmappen-Explorer erscheint die neue Datei *Win32\_Products.vb*.

Mit diesen Voraussetzungen kann nun eine Instanz der Klasse *Product* erzeugt werden. Die Auflistung der Programmobjekte geschieht in der Variablen *prEnum*.

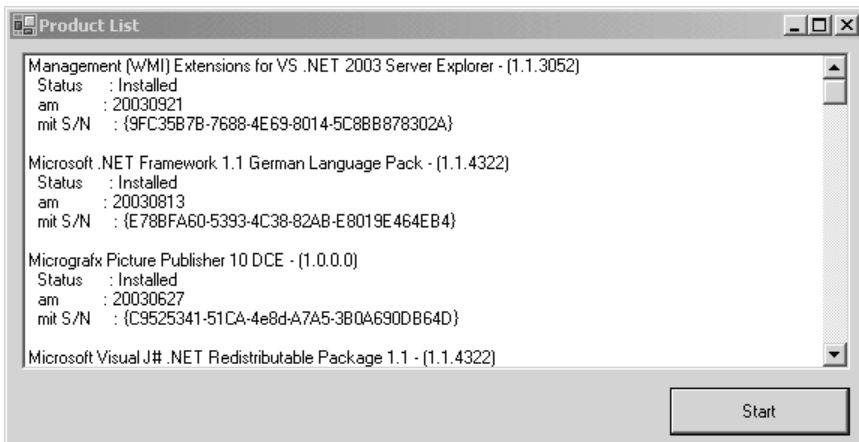


Abbildung 203: Liste der installierten Programme

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

Da die Eigenschaft `InstallState` als Zeichenkette geliefert wird, wird eine Konvertierung zu Beginn des `Select Case`-Zweiges durchgeführt.

Wie eine solche Liste beispielhaft aussieht, kann in Abbildung 203 gesehen werden. Interessant ist hier das Feature der Seriennummer. Mit der Funktion `GetProd()` ist es ein Leichtes, sich innerhalb eines Netzes alle mittels MSI installierten Programme auflisten zu lassen.

## 215 Programm über Namen starten

Möchte man aus seinem Programm heraus ein anderes Programm starten, so hat man die Wahl zwischen zwei Methoden. Einmal kann man ein Programm über den Namen der ausführbaren Datei starten. Diese Methode wird hier gezeigt. Die zweite Art startet ein Programm indirekt über den Dateinamen. Dies wird in Rezept [11.18](#) gezeigt.

---

```
Public Function StartProgram(ByVal PrgName As String, _
    Optional ByVal FileName As String = Nothing) As Boolean

    Dim Program As ProcessStartInfo = New ProcessStartInfo
    Dim ProgramProcess As Process = New Process

    ' Wir starten ein Programm
    Program.UseShellExecute = False

    ' Es soll ein Fenster angezeigt werden
    Program.CreateNoWindow = False

    ' Der Name des auszuführenden Programms
    Program.FileName = PrgName

    ' Normales Fenster
    Program.WindowStyle = ProcessWindowStyle.Normal

    ' Das Arbeitsverzeichnis des Programmes
    Program.WorkingDirectory = "C:\TEMP"

    If FileName <> Nothing Then
        Program.Arguments = FileName
    Else
        Program.Arguments = ""
    End If

    Try
        ProgramProcess.Start(Program)
    Catch ex As SystemException
        Throw New ApplicationException("StartProgram", ex)
    End Try

    Return True
End Sub
```

---

Der Funktion `StartProgram` werden der Name der ausführbaren Datei und ein optionaler Startparameter für dieses Programm übergeben. Die Eigenschaft `UseShellExecute` legt fest, ob zum Starten des Programms die Shell des Betriebssystems benutzt werden soll. Ist diese Eigenschaft auf `False` gesetzt, können dem zu startenden Prozess nur ausführbare Dateien übergeben werden.

Das Programmfenster soll nach dem Start Normal erscheinen. Insgesamt gibt es vier Fenstervariationen, mit denen ein Fenster erstellt werden kann (siehe Tabelle 30).

| Fensterstil | Beschreibung                                                                                                                                                                                                                |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Hidden      | Das Fenster wird versteckt angezeigt. Es ist zwar vorhanden und kann Meldungen empfangen und selber Meldungen schicken. Es ist aber nicht möglich, Anwendereingaben zu verarbeiten.                                         |
| Maximized   | Das Fenster wird in der Größe des Bildschirms dargestellt. Benutzereingaben sind möglich. Sollte das Fenster ein Clientfenster eines Programmes sein, so wird der Bereich des hierarchisch übergeordneten Fensters gefüllt. |
| Minimized   | Das Fenster wird auf der Taskleiste abgelegt.                                                                                                                                                                               |
| Normal      | Das Fenster wird mit seiner Standardgröße auf dem Bildschirm oder im Clientbereich des übergeordneten Programmes angezeigt.                                                                                                 |

Tabelle 30: Mögliche Fensterstile

Es wird ein Arbeitsverzeichnis benötigt, da das Programm zum Beispiel wissen muss, wo es Dateien abzulegen hat, sollte man keine speziellen Angaben während des Programmlaufes des gestarteten Programms zum Speicherort machen.

Da der Dateiname optional mit `Nothing` vorbelegt wurde, muss bei den Übergabe-Argumenten für das Programm nötigenfalls eine leere Zeichenkette angegeben werden. Eine leere Zeichenkette ist eine Zeichenkette mit all dem »Ballast«, den eine Zeichenkette mit sich herum trägt. Der Wert `Nothing` steht für Nichts, noch nicht einmal eine leere Zeichenkette.

Hat der Übergabeparameter `FileName` einen Wert, wird dieser dem Programm als Argument mit gegeben. Dies entspricht der Vorgehensweise bei der Übergabe von Argumenten auf der Befehlszeile.

## 216 Programm über Datei starten

Hat man eine Datei, benötigt man nicht unbedingt die Kenntnis über das Programm, welches diese Datei anzeigen kann. Kennt das Betriebssystem die Dateiendung (zum Beispiel `.txt`), so kann es sich das dazugehörige Programm aus der Registry des Rechners holen und anschließend zusammen mit der Datei starten. Die Funktion `StartShell` aus Listing 389 erledigt diese Aufgabe.

```
Public Function StartShell(ByVal FileName As String) As Boolean

    Dim Program As ProcessStartInfo = New ProcessStartInfo
    Dim ProgramProcess As Process = New Process

    Program.Arguments = FileName
```

Listing 389: Programmstart über einen Dateinamen

---

```

' Dateiname als Startkriterium
Program.UseShellExecute = True

' Fenster soll angezeigt werden
Program.CreateNoWindow = False

' Normales Fenster
Program.WindowStyle = ProcessWindowStyle.Normal

' Arbeitsverzeichnis
Program.WorkingDirectory = "C:\TEMP"

Try
    ProgramProcess.Start(FileName)
Catch ex As SystemException
    Throw New ApplicationException("StartShell", ex)
End Try

Return True
End Sub

```

---

*Listing 389: Programmstart über einen Dateinamen (Forts.)*

Der Funktion wird der Dateiname übergeben, für den ein gesonderter Prozess gestartet werden soll. Die Eigenschaft `UseShellExecute` wird auf `True` gesetzt. Damit wird die Shell des Betriebssystems genutzt, die nach dem registrierten Programm für die übergebene Dateiart sucht. So werden `.txt`-Dateien typischerweise mit dem Notepad gestartet, während `.doc`-Dateien durch Word bearbeitet werden.

## 217 Parameterübergabe per Befehlszeile

Bestimmte Programme gewinnen durch die Tatsache, dass man ihnen Dateien, Einstellungen oder Ähnliches beim Programmstart über die Befehlszeile übergeben kann. Nach Programmstart kann man diese Übergabeparameter mit der Methode `GetCommandLineArgs` aus dem Namensraum `System.Environment` auslesen. Wie dies geschieht, kann man dem Listing 390 entnehmen.

---

```

Sub Main()
    Dim i As Integer

    Console.WriteLine()
    Dim Arguments As String()
    Dim Argument As String
    Dim FileName As String
    Dim User As String

    arguments = Environment.GetCommandLineArgs()

    For i = 0 To arguments.Length - 1
        Console.WriteLine("Übergabeparameter {0}: {1}", i, arguments(i))
    
```

---

*Listing 390: Parameterübergabe per Befehlszeile*

```
Argument = Arguments(i).ToLower
Select Case Argument
    Case "file"
        i += 1
        FileName = Arguments(i)
    Case "user"
        i += 1
        User = Arguments(i)
End Select
Next

Console.WriteLine()
Console.WriteLine("Dateiname = {0}", FileName)
Console.WriteLine("User = {0}", User)

Console.WriteLine("Weiter mit <CR>")
Console.ReadLine()
End Sub
```

Listing 390: Parameterübergabe per Befehlszeile (Forts.)

Die einzelnen Übergabewerte werden als Zeichenketten-Array zur Verfügung gestellt. In Listing 390 wird die Übergabe von zwei Parametern realisiert, einem Dateinamen und einem Benutzer. Nachdem die Übergabeparameter mit `GetCommandLineArgs()` dem Programm bekannt gemacht sind, werden diese in einer Schleife über alle Parameter ausgewertet. Um die Reihenfolge der Angabe von Datei und Benutzer variabel zu halten, wird erst getestet, ob es sich um den Parameter `file` handelt. Ist dies der Fall, so muss der nächste Parameter die übergebene Datei sein. Analog verhält es sich mit dem Benutzer. Daher wird jeweils nach der positiven Feststellung die Schleifenvariable innerhalb der Schleife um 1 erhöht und der entsprechende Wert zugewiesen.

Die Auswertung der Übergabeparameter in diesem Beispiel ist recht einfach gehalten. Man kann sich beliebig viele Fehler vorstellen, die ein Anwender an dieser Stelle begehen kann. Eine fehler-tolerante Implementierung einer solchen Funktion würde den Rahmen sprengen. Zudem ist sie für jede Situation anders. Wenn man aber die Möglichkeit der Parameterübergabe nur für interne Zwecke (also nicht zur freien Verfügung des Programmbenutzers) benutzt, stellen sie eine deutliche Erleichterung in einigen Situationen dar.

Startet man ein Programm wie aus Listing 390 in der Entwicklungsumgebung, ergibt sich zwangsläufig die Frage, wo denn die Übergabeparameter bei Programmstart hergenommen werden. Für diese Zwecke können im Eigenschaften-Dialog des Projektes unter dem Eintrag KONFIGURATIONSEIGENSCHAFTEN → DEBUGGEN im Bereich STARTOPTIONEN die einzelnen Parameter hinterlegt werden (Abbildung 204).

Die Ausgabe des Programmes aus Listing 390 ist in Abbildung 205 zu sehen.

An der Abbildung 205 erkennt man auch sehr deutlich, dass der erste Übergabeparameter auch den Array-Index 1 hat, obwohl die Schleife bei Null beginnt. Der nullte Übergabeparameter ist immer der Programmname inklusive des Pfades.

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Proper-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

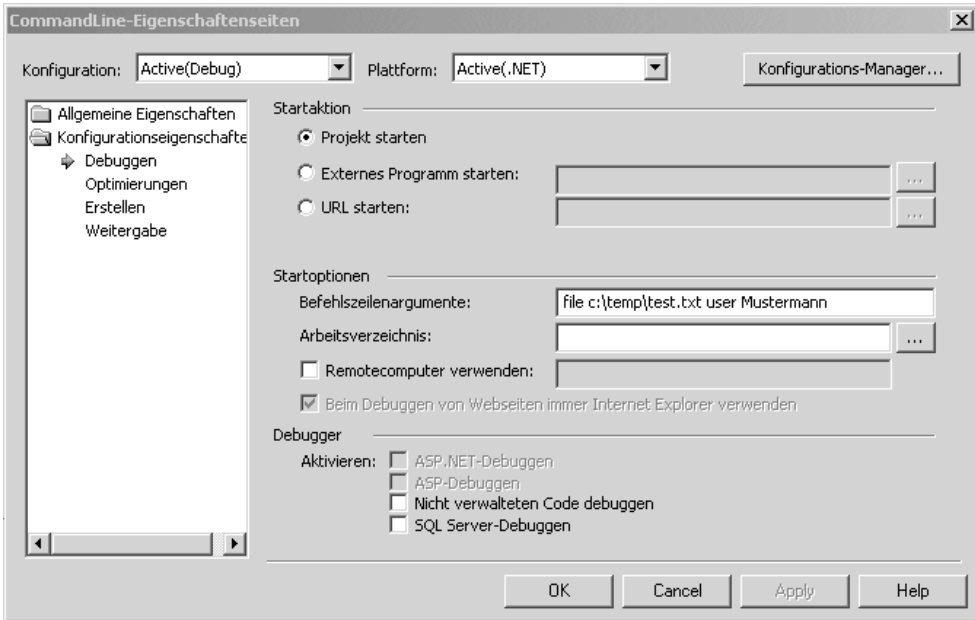


Abbildung 204: Einstellungen zum Testen von Kommandozeilen-Parametern

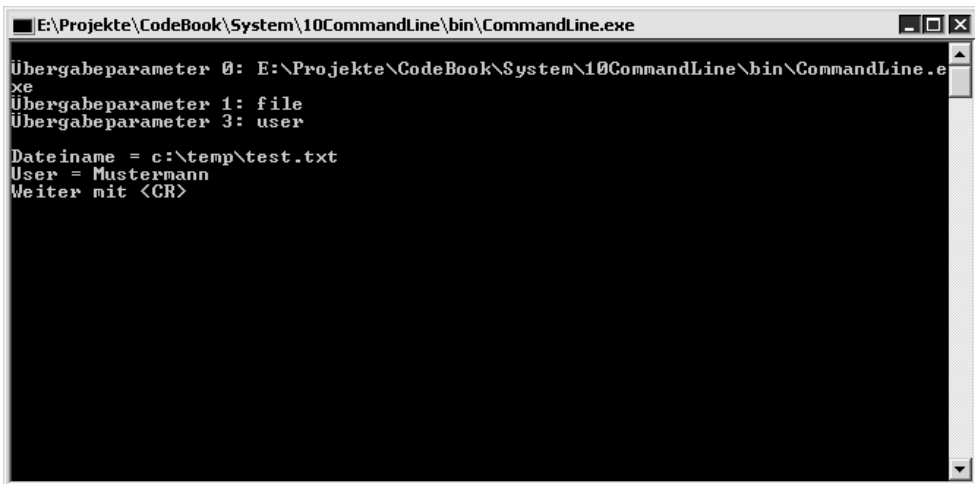


Abbildung 205: Ausgabe des Programmes CommandLine.exe

## 218 Systemprozesse mit WMI

Um eine Liste der gerade laufenden Prozesse innerhalb des eigenen Programmes zu erhalten, gibt es zwei Möglichkeiten. Variante eins benutzt die Möglichkeiten des WMI. Wer sich mit dieser Technologie auskennt und schon in der »Vor-.NET-Ära« angewandt hat (siehe dazu die hier gezeigten Rezepte), kommt recht schnell an erste Ergebnisse. Für die reine Implementierung in .NET siehe Rezept 392.

Um die Daten aus einer Funktion zurückzugeben, empfiehlt sich auch wieder der Datentyp `ArrayList`, da man mit Sicherheit davon ausgehen kann, dass auf dem System mehrere Prozesse aktiv sind. Die Einträge in dieser Liste sind vom Typ `ProcessStruc`, einer Struktur-Variablen, wie sie in Listing 391 zu sehen ist. Zwei Besonderheiten sind an dieser Struktur zu vermerken. Die Strukturvariable `Threads` beinhaltet eine Semikolon-separierte Liste von Prozesskennungen der zum betrachteten Prozess gehörenden, aktiven Threads. Benötigt man die einzelnen Werte dieser Liste, so kann man eine Trennung mit `String.Split` durchführen. Hier kann man sich allerdings auch als Programmiererweiterung eine Liste vorstellen, falls man weitere Informationen zu diesen Threads benötigt.

```
Public Structure ProcessStruc
    Dim Name As String
    Dim Starttime As String
    Dim PageMemoryMax As String
    Dim NonPagedMemory As String
    Dim ProcessID As String
    Dim TotalProcTime As String
    Dim TotalUserTime As String
    Dim WorkingMemory As String
    Dim Threads As String
    Dim StartInfo As System.Diagnostics.ProcessStartInfo
End Structure
```

Listing 391: Struktur zur Übermittlung der Prozessdaten

Die Strukturvariable `StartInfo` ist deklariert als Typ `System.Diagnostics.ProcessStartInfo`. Über sie kann man sich die entsprechenden Informationen zu den Startbedingungen des Prozesses holen. Zu beachten ist hier allerdings, dass nicht alle Member von `StartInfo` mit Werten gefüllt werden.

Die Funktion `GetProcessInfo` aus Listing 392 liefert die geschilderte Liste der Prozessinformationen. Um mit den WMI-Klassen in der dargestellten Form arbeiten zu können, muss aus dem SERVER-EXPLORER eine neue Datei für MANAGEMENT CLASSES → PROCESSES erstellt werden. Die erstellte Datei heißt `Win32_Process.vb` und wird dem Projekt automatisch hinzugefügt. Diese Datei enthält unter anderem die Definition der Klasse `Process`.

```
Public Function GetProcessInfo() As ArrayList
    Dim Struc As ProcessStruc
    Dim mProcess As Process
    Dim mZeile As String
    Dim mThreadCollection As ProcessThreadCollection
    Dim mThread As ProcessThread
    Dim mProcArray As ArrayList = New ArrayList

    For Each mProcess In Process.GetProcesses

        ' Bezeichnung des Prozesses
        Struc.Name = mProcess.ProcessName

        ' Maximale ausgelagerte Speicherseiten
```

Listing 392: Die Funktion `GetProcessInfo`

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

```

Struc.PageMemoryMax = mProcess.PeakPagedMemorySize.ToString

' Nicht ausgelagerte Seiten
Struc.NonPagedMemory = _
    mProcess.NonpagedSystemMemorySize.ToString

' Prozess-Kennung
Struc.ProcessID = mProcess.Id.ToString

' Zeitpunkt des Prozessesstarts
Struc.Starttime = mProcess.StartTime.ToShortDateString + _
    " " + mProcess.StartTime.ToShortTimeString

' Auflistung der zu diesem Prozess gehörigen Threads
mThreadCollection = mProcess.Threads
For Each mThread In mThreadCollection
    mZeile += (mThread.Id.ToString + "; ")
Next
Struc.Threads = mZeile

' Gesamt-Prozessorzeit in Sekunden
Struc.TotalProcTime = mProcess.TotalProcessorTime.TotalSeconds

' Gesamt-Userzeit in Sekunden
Struc.TotalUserTime = mProcess.UserProcessorTime.TotalSeconds

' aktuell benötigter Speicher
Struc.WorkingMemory = mProcess.WorkingSet

' Informationen vom Typ System.Diagnostics.ProcessStartInfo
Struc.StartInfo = mProcess.StartInfo

mProcArray.Add(Struc)
Next

Return mProcArray
End Function

```

---

Listing 392: Die Funktion *GetProcessInfo* (Forts.)

Um die Liste der Prozesse durchlaufen zu können, wird die Variable `mProcess` deklariert, die den jeweils aktuellen Prozess aus der Liste `Process.GetProcesses` enthält.

Die vom Prozess gestarteten Threads werden von der Methode `Threads` des Prozesses als `Collection` zurückgeliefert. Aus dieser `Collection` wird anschließend durch eine `for each`-Schleife eine Semikolon-getrennte Liste erstellt und in einem `String` gespeichert. Die Zusammenstellung des Strings nutzt die neuen Möglichkeiten des *.NET* und entspricht in seiner Knappheit mehr dem C/C++ als dem klassischen Basic.

Die Klassenmitglieder, die einen Datumseintrag und/oder einen Zeitpunkt darstellen, sind vom Typ `DateTime` oder `TimeSpan`, so dass man die entsprechenden Konvertierungsfunktionen dieser Typen anwenden kann. So wird mit `TotalSeconds` zum Beispiel die im User-Modus verbrachte Zeit eines Prozesses umgerechnet.



Ein Ausgabebeispiel ist in Abbildung 206 zu sehen.

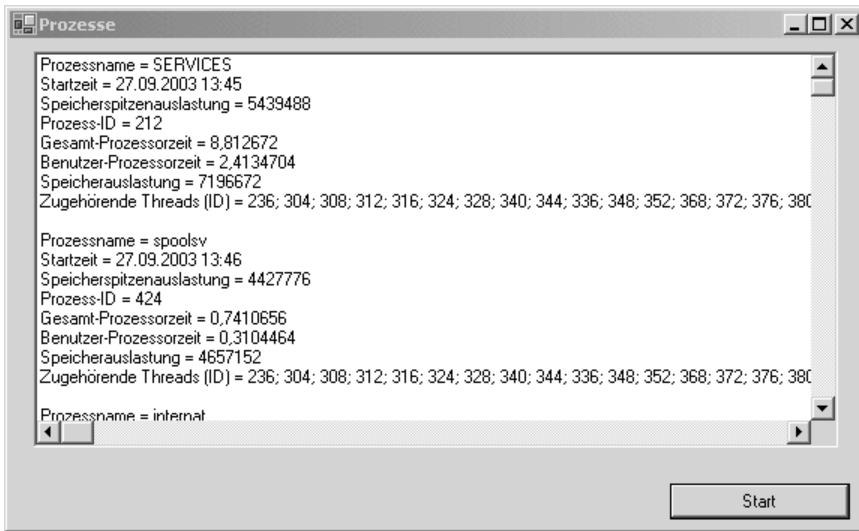


Abbildung 206: Beispielausgabe für die Funktion `GetProcessInfo`

## 219 Systemprozesse mit .System.Diagnostics

Eine etwas anders geartete Informationsstruktur bekommt man über die Methoden von `.NET`. Im recht umfangreichen Namensraum `System.Diagnostics` finden sich Klassen und Methoden, die die Verwaltung von Prozessen behandeln. Es handelt sich um die `Process`-Klasse an sich und die Klassen `ProcessThread` und `ProcessModule`. Hinzu kommt die schon erwähnte Klasse `ProcessStartInfo`, die in den letzten Kapiteln schon aufgetaucht ist.

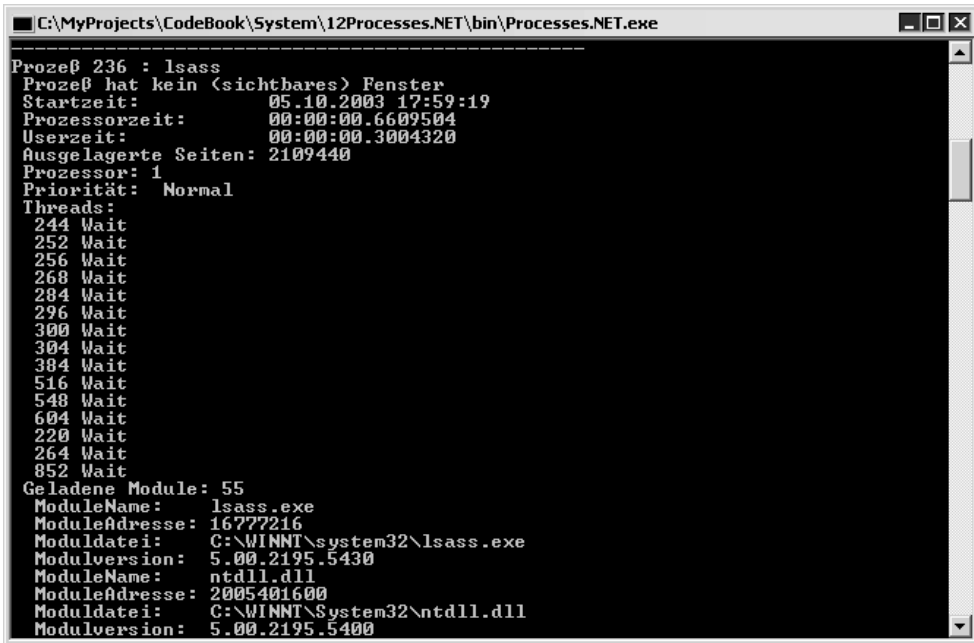
Einen Ausschnitt der gelesenen Möglichkeiten können Sie in Abbildung 207 sehen. Man erkennt die Prozess-ID, den Prozessnamen und als nächste Information, dass dieser Prozess kein sichtbares Fenster hat.

Seit dem Start des Rechners hat der Prozess 0,66 Sekunden Prozessorzeit und 0,3 Sekunden Benutzerzeit beansprucht

Nach den ausgelagerten Seiten ist zu sehen, auf welchem Prozessor der Prozess zurzeit ausgeführt wird. Bei Computern mit nur einem Prozessor ist diese Information dann aber sicherlich nicht überraschend.

Das entsprechende Programmlisting ist in Listing 393 abgedruckt. Die Bedeutung der einzelnen Variablen ist den Kommentaren zu entnehmen. Um an die Prozesse des Computers zu gelangen, muss eine Klasse vom Typ `Process` erstellt werden. Über dieses Objekt kann dann mit der Methode `GetProcesses` eine Prozessliste abgerufen werden. Daher die Bezeichnung `mProcessRoot` für die entsprechende Variable.

Eine Reminiszenz an alte Windows-Zeiten ist die Definition einer Variablen für ein Windows-Handle! Auch mit `.NET` ist man noch nicht vor solchen »Überbleibseln« geschützt. Mit dem Windows-Handle lässt sich feststellen, ob der Prozess ein Fenster hat. Nur wenn das der Fall ist, existiert ein solches Windows-Handle. Um das ungewünschte Erzeugen einer Ausnahme zu verhindern, sollte man vor der Abfrage nach dem Fenstertitel erst mal auf das Vorhandensein eines Fensters testen. Auch unter Windows ist nicht alles ein Window J.



```

C:\MyProjects\CodeBook\System\12Processes.NET\bin\Processes.NET.exe
-----
Prozeß 236 : lsass
Prozeß hat kein <sichtbares> Fenster
Startzeit:      05.10.2003 17:59:19
Prozessorzeit:  00:00:00.6609504
Userzeit:       00:00:00.3004320
Ausgelagerte Seiten: 2109440
Prozessor: 1
Priorität: Normal
Threads:
244 Wait
252 Wait
256 Wait
268 Wait
284 Wait
296 Wait
300 Wait
304 Wait
384 Wait
516 Wait
548 Wait
604 Wait
220 Wait
264 Wait
852 Wait
Geladene Module: 55
ModuleName: lsass.exe
ModuleAdresse: 16777216
Moduledatei: C:\WINNT\system32\lsass.exe
Moduleversion: 5.00.2195.5430
ModuleName: ntdll.dll
ModuleAdresse: 2005401600
Moduledatei: C:\WINNT\System32\ntdll.dll
Moduleversion: 5.00.2195.5400

```

Abbildung 207: Prozessinformationen mit ReadProcess

Ein Windos-Handle ist ein Zeiger auf einen Integer-Wert (C lässt grüßen). Dieser kann für die hier benötigten Zwecke durch einen `IntPtr` dargestellt werden. Bei 32-Bit-Systemen ist die Bitbreite von `IntPtr` 32 Bit, bei 64 Bit entsprechend ebenfalls 64 Bit.

---

```

Public Function ReadProcesses() As String
    ' Die Wurzel unserer Prozessbetrachtung
    Dim mProcessRoot As Process = New Process

    ' Laufvariable für Schleife über alle Prozesse
    Dim mProcess As Process

    ' Array aller Prozesse
    Dim mProcesses As Process()

    ' Thread-Collection zu mProcess
    Dim mProcessThreads As ProcessThreadCollection

    ' Laufvariable für die Thread-Collection
    Dim mThread As ProcessThread

    ' Handle des Prozessfensters
    Dim mHandleZero As IntPtr

    Dim mPriority As ProcessPriorityClass
    Dim StrB As StringBuilder

```

---

Listing 393: Die Funktion ReadProcess

```

Dim vbCrLf As String

StrB = New StringBuilder(1000)

' Abkürzung...
vbCrLf = ControlChars.CrLf

' Ermitteln aller laufenden Prozesse
mProcesses = mProcessRoot.GetProcesses

' Verarbeitung über jeden ermittelten Prozess
For Each mProcess In mProcesses
    ' Initialisierung des Window-Handle mit NULL-Pointer
    mHandleZero = New IntPtr(0)
    StrB.Append("-----")
    StrB.Append(vbCrLf)

    StrB.Append("Prozess " + mProcess.Id.ToString + " : " + _
        mProcess.ProcessName)
    StrB.Append(vbCrLf)

    If mProcess.MainWindowHandle.Equals(mHandleZero) Then
        StrB.Append(" Prozess hat kein (sichtbares) Fenster")
    Else
        StrB.Append(" FensterTitel: " + mProcess.MainWindowTitle)
    End If
    StrB.Append(vbCrLf)

    StrB.Append(" Startzeit:          " + _
        mProcess.StartTime.ToString)
    StrB.Append(vbCrLf)

    StrB.Append(" Prozessorzeit:          " + _
        mProcess.TotalProcessorTime.ToString)
    StrB.Append(vbCrLf)

    StrB.Append(" Userzeit:              " + _
        mProcess.UserProcessorTime.ToString)
    StrB.Append(vbCrLf)

    StrB.Append(" Ausgelagerte Seiten: " + _
        mProcess.PagedMemorySize.ToString)
    StrB.Append(vbCrLf)

    ' Wenn's nicht der Leerlaufprozess ist
    If Not mProcess.Id.Equals(0) Then
        StrB.Append(" Prozessor: " + _
            mProcess.ProcessorAffinity.ToString())
        StrB.Append(vbCrLf)
        StrB.Append(" Priorität: ")
        mPriority = mProcess.PriorityClass

```

Listing 393: Die Funktion ReadProcess (Forts.)

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Propor-  
tityGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schie-  
denes

---

```

Select Case mPriority
    Case ProcessPriorityClass.AboveNormal
        StrB.Append(" über normal")
    Case ProcessPriorityClass.BelowNormal
        StrB.Append(" unter Normal")
    Case ProcessPriorityClass.High
        StrB.Append(" hoch")
    Case ProcessPriorityClass.Idle
        StrB.Append(" Wartend")
    Case ProcessPriorityClass.Normal
        StrB.Append(" Normal")
    Case ProcessPriorityClass.RealTime
        StrB.Append(" Echtzeit")
End Select
StrB.Append(vbCrLf)
End If

mProcessThreads = mProcess.Threads()
StrB.Append(" Threads:")
StrB.Append(vbCrLf)

For Each mThread In mProcessThreads
    StrB.Append(" " + mThread.Id.ToString + " " + _
        mThread.ThreadState.ToString)
    StrB.Append(vbCrLf)
Next

' ID = 0 : Leerlaufprozess
' ID = 8 : Systemprozess
If Not ((mProcess.Id.Equals(0)) Or (mProcess.Id.Equals(8))) Then

    Try
        StrB.Append(" Geladene Module: " + _
            mProcess.Modules.Count.ToString)
        StrB.Append(vbCrLf)
        If mProcess.Modules.Count > 0 Then
            Dim i As Integer

            For i = 0 To mProcess.Modules.Count - 1
                StrB.Append(" ModuleName: " + _
                    mProcess.Modules.Item(i).ModuleName)
                StrB.Append(vbCrLf)

                StrB.Append(" ModuleAdresse: " + _
                    mProcess.Modules.Item(i).BaseAddress.ToString)
                StrB.Append(vbCrLf)

                StrB.Append(" Moduldatei: " + _
                    mProcess.Modules.Item(i).FileName)
                StrB.Append(vbCrLf)
            Next i
        End If
    Catch ex As Exception
    End Try

```

---

```

        StrB.Append("  Modulversion: " + _
            mProcess.Modules.Item(i).FileVersionInfo.FileVersion)
        StrB.Append(vbCrLf)
    Next
End If

Catch ex As Exception
    StrB.Append(ex.ToString)
    StrB.Append(vbCrLf)
End Try
End If
Next

Return StrB.ToString
End Function

```

Listing 393: Die Funktion *ReadProcess* (Forts.)

Der weitere Ablauf der Funktion ist wohl weitestgehend selbsterklärend. Nur an zwei Stellen könnte man stutzig werden. An der ersten Stelle wird auf die Prozess-ID = 0, an der zweiten Stelle auf die Prozess-ID 8 geprüft.

Der Prozess mit der ID 0 ist auf allen Rechnern der Leerlaufprozess. Für diesen Prozess sind einige Angaben einfach unsinnig oder erst gar nicht zu bekommen (Ausnahme-Aufruf). Der Prozess mit der ID 8 ist der System-Prozess. Für ihn gilt Ähnliches. Auf allen getesteten Rechnern hat dieser Prozess die ID 8. Informationen zu den Prozessen 2 bis 7 haben wir nicht gefunden.

Bei der Anzeige von Modulinformationen wird *FileVersionInfo* eingesetzt. Näheres zu dieser Klasse ist im Kapitel 9 ((Dateisystem)) zu finden.

Da alle Informationen in einem *String* abgelegt werden, wird innerhalb mit einem *StringBuilder*-Objekt gearbeitet. Am Ende wird dieses Objekt in einen normalen *String* konvertiert und zurückgeliefert. Diesen *String* kann man dann zum Beispiel mit Mitteln der *Regex*-Klasse analysieren.

## 220 Liste aller Dienste

Benötigt man Informationen zu Diensten auf einem Computer, so kann man sich der Hilfsmittel, die im Namensraum *System.ServiceProcess* angesiedelt sind, bedienen. Dieser Namensraum ist allerdings nicht standardmäßig eingeblendet. Man muss sich erst über PROJEKT/VERWEIS HINZUFÜGEN einen Verweis auf *System.ServiceProcess.DLL* in das Projekt integrieren und über Imports *System.ServiceProcess* dem Programm bekannt machen. Die *ServiceController*-Klasse in diesem Namensraum repräsentiert einen Dienst und über ein entsprechendes Objekt lassen sich die entsprechenden Informationen holen.

```

Public Function GetServiceList() As ArrayList
    Dim Services() As ServiceController
    Dim Service As ServiceController
    Dim Liste As ArrayList = New ArrayList

```

Listing 394: Die Funktion *GetServiceList*

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

```

Services = _
    ServiceController.GetServices()

For Each Service In Services
    Liste.Add(Service.ServiceName + " = " + _
        Service.Status.ToString)
Next

Return Liste
End Function

```

---

Listing 394: Die Funktion *GetServiceList* (Forts.)

Mit der Methode `GetServices()` werden die Namen der Dienste in ein `ServiceController`-Array abgelegt. Hierbei werden aber alle Dienste, die etwas mit Gerätetreibern zu tun haben, außen vor gelassen. Benötigt man Informationen zu solchen Gerätetreiber-Diensten, muss man `GetDevices` aus dem gleichen Namensraum anwenden.

Sowohl `GetServices()` als auch `GetDevices()` können als Parameter den Namen des Computers erhalten, für den die Liste ermittelt werden soll, entsprechende Rechte vorausgesetzt. Eine Funktion, die eine entsprechende Liste liefert, ist in Listing 394 zu sehen.

Diese Liste beinhaltet auch den Status des Dienstes, also Aktiv, Angehalten, Gestoppt. In Abbildung 208 ist die Anwendung dieser Funktion zu sehen. Sie wurde auf einem Computer mit amerikanischem Windows 2000 und deutscher Kultureinstellung ausgeführt.

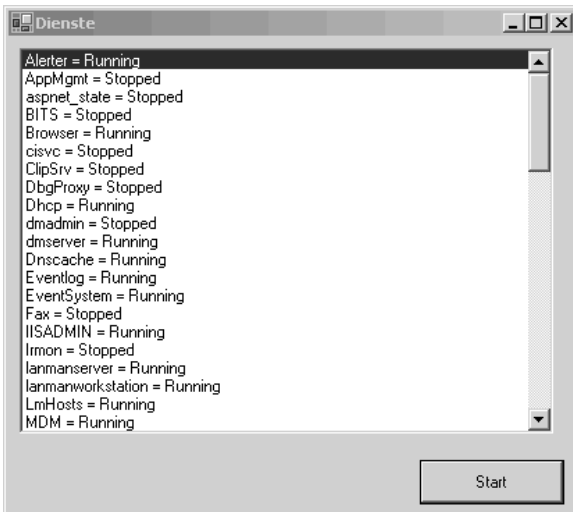


Abbildung 208: Anwendung der Funktion *GetServiceList()*

## 221 Dienst starten

Neben der Auflistung der installierten Dienste ermöglichen die Klassen im Namensraum `ServiceProcess` auch das Starten, Anhalten, Weiterlaufen lassen und Stoppen (`Start`, `Pause`, `Continue`, `Stop`) einzelner Dienste. Hierzu muss ein Verweis auf `System.ServiceProcess.DLL` in das Projekt aufgenommen und mittels

```
Imports System
Imports System.ServiceProcess
```

importiert werden. Um eine Ausnahme zu verhindern, muss man auf den aktuellen Status des jeweiligen Dienstes achten. Dienste arbeiten nach dem Prinzip der »State-Machine«, d.h. von einem Status sind nur einige wenige, manchmal nur ein Ausgang möglich. Alle anderen sind verboten. Wie diese Übergänge miteinander verbunden sind, ist in Abbildung 209 als UML-Status-Diagramm näher erläutert.

Da die Zwischenzustände (...Pending) nur eine relativ kurze Zeit in Anspruch nehmen, geht man normalerweise das Risiko ein und lässt bei der Programmierung diese Zustände außen vor und behandelt sie als Fehler.

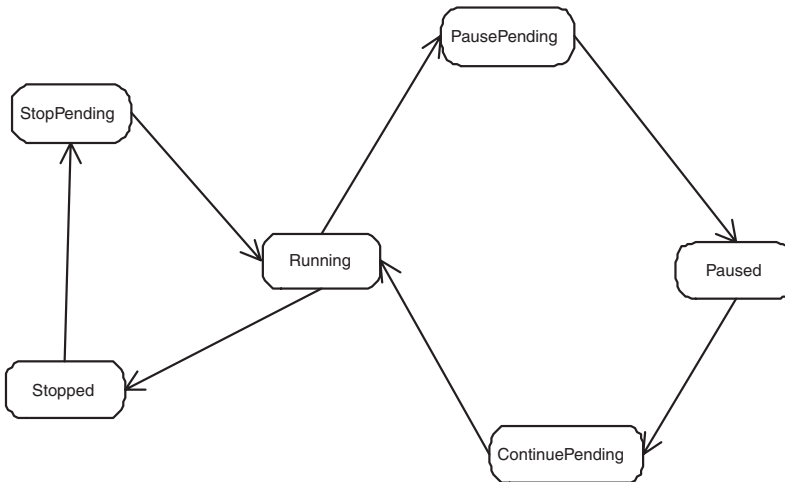


Abbildung 209: Statusübergänge von Diensten



Abbildung 210: Dienste-Beispiel

Eine mögliche Implementierung einer Funktion zum Starten eines Dienstes ist in Listing 396 zu sehen. Dieser Funktion werden der zu startende Dienst, der Computer, auf dem dieser Dienst gestartet werden soll, die Wartezeit in Sekunden und beliebig viele Startparameter des Dienstes übergeben. Die Wartezeit dient der Funktion dazu, nach Ablauf dieser Zeit nach dem Status zu schauen und damit festzustellen, ob die Aktion auch durchgeführt wurde.

Da Dienste eine unbestimmte Anzahl an Startparametern haben können, wurde dies durch ein `ParamArray` realisiert, in dem die Startparameter als Zeichenkette hinterlegt werden.

Als Dienstnamen können die aus Listing 394 ermittelten Namen genommen werden. Sie entsprechen den Namen, die man auch über die `SYSTEMSTEUERUNG` einsehen kann.

---

```
Public Function StartService(ByVal Name As String, _
    ByVal Computer As String, ByVal Wait As Integer, _
    ByVal ParamArray Parameter() As String) _
    As ServiceControllerStatus

    Dim ts As TimeSpan

    If Name = String.Empty Or Name = Nothing Then
        Throw New ApplicationException("Dienste - Name fehlt")
    End If

    If Computer = String.Empty Or Computer = Nothing Then
        Computer = "."
    End If

    ' Verweis auf System.ServiceProcess.DLL
    Dim mService As ServiceController = _
        New ServiceController(Name, Computer)

    If (mService.Status = ServiceControllerStatus.Running Or _
        mService.Status = ServiceControllerStatus.ContinuePending Or _
        mService.Status = ServiceControllerStatus.StartPending) Then

        Throw New ApplicationException _
            ("Dienst kann nicht gestartet werden.")
    Else
        If Not (Parameter Is Nothing) Then
            mService.Start(Parameter)
        Else
            mService.Start()
        End If
    End If

    ' Man könnte auch auf StartPending warten, Event senden und dann
    ' auf Running warten
    ' Wait ist Wartezeit in Sekunden
    ts = New TimeSpan(0, 0, Wait)
    mService.WaitForStatus(ServiceControllerStatus.Running, ts)

    Return mService.Status
End Function
```

---

Listing 395: Dienst starten mit `StartService()`



Nach der Deklaration der Variablen `ts` für die Wartezeit wird neben dem Check auf den Dienstnamen eine Überprüfung des Rechnernamens durchgeführt. Durch den Einsatz des `ParamArrays` kann man den Rechnernamen nicht mit einem optionalen Wert für den lokalen Rechner an die Funktion übergeben. `Optional` und `ParamArray` schließen einander aus. Sollte der Funktion kein Rechnernamen übergeben werden, wird hier der lokale Rechner eingestellt.

Befindet sich der Prozess in einem ungültigen Status, wird eine Ausnahme ausgelöst, ansonsten wird der Dienst gestartet. Da die Methode `WaitForStatus` eine Zeitangabe vom Typ `TimeSpan` benötigt, wird die übergebene Zeit in Sekunden in den entsprechenden `TimeSpan`-Wert überführt. Die Funktion liefert anschließend den aktuellen Status als Typ `ServiceControllerStatus` zurück. Erforderlich ist dieses Vorgehen, da ansonsten die Funktion sofort beendet wird und der Status des Dienstes nicht bekannt ist.

```
Private Sub btnStart_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnStart.Click

    Dim Status As ServiceControllerStatus

    Try
        Status = StartService(txtName.Text, ".", 2)
    Catch ex As ApplicationException
        lblStatus.Text = ex.Message
    End Try

    lblStatus.Text = Status.ToString

End Sub
```

Listing 396: Aufrufbeispiel von `StartService`

Wie diese Funktion eingesetzt werden kann, ist beispielhaft im Listing 396 zu sehen. In Abbildung 210 kann man die Bildschirmausgabe dieses Versuches sehen.

## 222 Dienst anhalten

Einen Dienst anhalten bedeutet, den entsprechenden Prozess aktiv zu halten und nur seine Ausführung zu unterbinden. Damit bleiben die mit diesem Prozess zusammenhängenden Parameter wie beispielsweise die Prozess-ID erhalten. Eine entsprechende Funktion, die dies zur Verfügung stellt, ist in Listing 397 zu sehen.

```
Public Function PauseService(ByVal Name As String, _
    ByVal Computer As String, ByVal Wait As Integer) _
    As ServiceControllerStatus

    Dim ts As TimeSpan

    If Name = String.Empty Or Name = Nothing Then
        Throw New ApplicationException("Dienste - Name fehlt")
    End If
```

Listing 397: Dienst anhalten

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Propert-  
yGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

```

If Computer = String.Empty Or Computer = Nothing Then
    Computer = "."
End If

' Verweis auf System.ServiceProcess.DLL
Dim mService As ServiceController = _
    New ServiceController(Name, Computer)

' Kann der Dienst pausieren?
If mService.CanPauseAndContinue = False Then
    Throw New ApplicationException("Dienst kann nicht pausieren.")
End If

' Wenn der Dienst im richtigen Status zum Pausieren ist
If (mService.Status = ServiceControllerStatus.Paused Or _
    mService.Status = ServiceControllerStatus.PausePending Or _
    mService.Status = ServiceControllerStatus.Stopped Or _
    mService.Status = ServiceControllerStatus.Stopped) Then

    Throw New ApplicationException _
        ("Dienst kann nicht angehalten werden.")
Else
    mService.Pause()
End If

ts = New TimeSpan(0, 0, Wait)

mService.WaitForStatus(ServiceControllerStatus.Paused, ts)

Return mService.Status

End Function

```

---

*Listing 397: Dienst anhalten (Forts.)*

Der Ablauf der Funktion ist weitestgehend identisch mit dem Code zum Starten eines Dienstes. Einen Unterschied gibt es aber zu erwähnen. Nicht jeder Dienst kann in den Status `Paused` versetzt werden. Diese Möglichkeit wird mit der Abfrage auf die Eigenschaft `CanPauseAndContinue` gesichert. Sollte der Dienst nicht über diese Möglichkeit verfügen, wird eine Ausnahme generiert.

Nach dem Aufruf der `Pause`-Methode muss auch hier wieder eine Wartezeit eingefügt werden, da die Methode sofort an den Aufrufer zurückgibt ohne auf das Ergebnis zu warten.

Mit dieser Funktion sollte man vorsichtig umgehen, da es andere Dienste geben kann, die von dem gerade zu pausierenden Dienst abhängig sind! Welche Dienste dies sind, kann im Vorfeld mit der Methode `ServiceController.DependentServices()` festgestellt werden. Diese Funktionalität wurde hier nicht implementiert, da der Einsatz einer solchen Abfrage sehr problemspezifisch ist. Mit den Kenntnissen aus diesem Rezept dürfte eine eigene Implementierung aber keine Schwierigkeiten bereiten.

## 223 Dienst fortsetzen

Das Fortsetzen eines Dienstes setzt den Status `Paused` voraus. Auch hier muss wie in 385 darauf getestet werden, ob der Dienst diese Möglichkeit überhaupt vorsieht. Eine entsprechende Implementierung finden Sie in Listing 398.

```
Public Function ContinueService(ByVal Name As String, _
    ByVal Computer As String, ByVal Wait As Integer) _
    As ServiceControllerStatus

    Dim ts As TimeSpan

    If Name = String.Empty Or Name = Nothing Then
        Throw New ApplicationException("Dienste-Name fehlt")
    End If

    If Computer = String.Empty Or Computer = Nothing Then
        Computer = "."
    End If

    ' Verweis auf System.ServiceProcess.DLL
    Dim mService As ServiceController = _
        New ServiceController(Name, Computer)

    If mService.CanPauseAndContinue = False Then
        Throw New ApplicationException("Dienst kann nicht fortsetzen.")
    End If

    If (mService.Status = ServiceControllerStatus.Paused) Then
        mService.Continue()
    End If

    ts = New TimeSpan(0, 0, Wait)

    ' Auch möglich servicecontrollerstatus.StartPending,
    ' dann Event, anschließend
    mService.WaitForStatus(ServiceControllerStatus.Running, ts)

    Return mService.Status
End Function
```

Listing 398: Dienst fortsetzen

Die Funktion hat eine leicht abgeänderte Logik im Bereich der Status-Kontrolle. Es wird nur abgefragt, ob der Dienst im Status `Paused` ist. Sollte dies der Fall sein, wird die Fortsetzung des Dienstes gestartet und für die übergebene Zeitspanne auf den Status `Running` gewartet. Sollte sich der Dienst nicht im Status `Paused` befinden, wartet er für die übergebene Zeitspanne auf den Status `Running`. Die Methode `WaitForStatus()` ist als Sub implementiert, liefert also keinen Statuswert zurück. Ist die Zeitspanne abgelaufen, beendet sich die Methode `WaitForStatus` automatisch und es wird der aktuelle Status mit der `Return`-Anweisung an das aufrufende Programm zurückgeliefert.

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schie-  
denes

## 224 Dienst stoppen

Das Stoppen eines Dienstes hat zur Folge, dass alle Ressourcen des entsprechenden Prozesses wieder freigegeben wurden. Auch hier muss man auf eventuell abhängige Dienste achten (siehe auch Kapitel [11.24](#)). Ob für den Dienst überhaupt das Stoppen erlaubt ist, muss mit der Eigenschaft `CanStop` abgefragt werden. So hätte es sicherlich die eine oder andere kleine Auswirkung, würde man den Systemdienst stoppen.

---

```
Public Function StopService(ByVal Name As String, _
    ByVal Computer As String, ByVal Wait As Integer) _
    As ServiceControllerStatus

    Dim ts As TimeSpan

    If Name = String.Empty Or Name = Nothing Then
        Throw New Exception("Dienste-Name fehlt")
    End If

    If Computer = String.Empty Or Computer = Nothing Then
        Computer = "."
    End If

    ' Verweis auf System.ServiceProcess.DLL
    Dim mService As ServiceController = _
        New ServiceController(Name, Computer)

    If mService.CanStop = False Then
        Throw New ApplicationException _
            ("Dienst kann nicht gestoppt werden.")
    End If

    If (mService.Status = ServiceControllerStatus.StopPending Or _
        mService.Status = ServiceControllerStatus.Stopped) Then
        Throw New ApplicationException _
            ("Dienst kann nicht gestoppt werden.")
    Else
        mService.Stop()
    End If

    ts = New TimeSpan(0, 0, Wait)
    mService.WaitForStatus(ServiceControllerStatus.Stopped, ts)

    Return mService.Status

End Function
```

---

*Listing 399: Dienst stoppen*

Das Listing 399 bringt hier nichts Überraschendes. Man darf die Abfrage nach `CanStop` allerdings nicht mit einer möglichen Abfrage nach `CanShutdown` verwechseln. Die Eigenschaft `CanShutdown` signalisiert, dass dieser Dienst beim Shutdown des Rechners benachrichtigt werden soll.

## 225 Prozess abbrechen (»killen«)

Das unbedingte Abbrechen eines Prozesses kann schwerwiegende Folgen für die Stabilität des laufenden Systems haben. Man sollte diese Möglichkeit nur als letzte Wahl betrachten. Alle Daten des Prozesses gehen verloren, es wird nichts gesichert. Eine sanftere Möglichkeit, einen Prozess zu beenden, ist die Methode `CloseMainWindow`. Sie beendet alle Prozesse ordnungsgemäß und versucht auch, offene Dateien zu speichern und zu schließen. Unter Umständen fragt `CloseMainWindow` den Benutzer über ein Fenster nach anstehenden Aktionen.

Die Funktion `KillProcess` aus Listing 400 zeigt das typische Vorgehen für das »Killen« eines Prozesses. Für die Methode `CloseMainWindow` ist das Vorgehen analog.

Jeder Prozess ist zwar eindeutig durch seine Prozess-ID bestimmt, doch kann ein Programm schlecht im `TASKMANAGER` nachschauen, welche ID das zu schließende Programm hat. Der Prozessname ist nicht eindeutig, da ein Prozess/Programm mehrmals gestartet werden kann. Alle Instanzen haben denselben Namen. Nun haben viele Prozesse durchaus ein Fenster, auch wenn dieses nicht immer zu sehen ist. Daher ist die Standardvorgehensweise in einem solchen Fall, sich den Titel des Fensters als Kriterium zu nehmen. So führen viele Programme den Dateinamen der geöffneten Datei im Titel, oder der Name des Benutzers wird im Fenstertitel eingeblendet.

Daher wird der Funktion `KillProcess` der Fenstertitel des abzubrechenden Prozesses übergeben.

---

```
Public Function KillProcess(ByVal Title As String) As ArrayList

    ' Der zu killende Hauptprozess
    Dim MainProcess As Process

    ' Ausgangsprozess
    Dim RootProcess As Process = New Process

    ' Die Liste aller Prozesse
    Dim Processes() As Process

    ' Laufvariable
    Dim Proc As Process

    ' Der Fenstertitel des gerade betrachteten Prozesses
    Dim RunTitle As String

    ' Die Threadliste eines Prozesses
    Dim ProcThreads As ProcessThreadCollection

    ' Einzelner Thread eines Prozesses
    Dim ProcThread As ProcessThread

    ' Für die Rückgabe an das aufrufende Programm
    Dim ThreadList As ArrayList = New ArrayList

    ' Prozessliste aller Prozesse holen
    Processes = RootProcess.GetProcesses

    ' Schleife über alle Prozesse
```

---

Listing 400: Prozess abbrechen: `KillProcess()`

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Propert-  
yGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

```

For Each Proc In Processes
    ' Windowstitel des Prozesses
    RunTitle = Proc.MainWindowTitle()
    If RunTitle = Title Then
        ' gesuchten Prozess gefunden
        MainProcess = Proc
        Exit For
    End If
Next

Try
    ' Liste der Prozess-Threads holen
    ProcThreads = MainProcess.Threads

    ' Wenn es Threads gibt, in die ArrayList aufnehmen
    If ProcThreads.Count > 0 Then
        For Each ProcThread In ProcThreads
            ThreadList.Add(ProcThread.Id.ToString)
        Next
    Else
        ThreadList.Add("Keine Threads gestartet.")
    End If

    ' Prozess killen
    MainProcess.Kill()
Catch ex As System.Exception
    Throw New ApplicationException _
        ("Prozess kann nicht gekillt werden." + ControlChars.CrLf + _
        ex.ToString)
End Try

Return ThreadList

End Function

```

---

Listing 400: Prozess abbrechen: KillProcess() (Forts.)

Nach der Deklaration und Definition einiger Variabler wird über die Variable `RootProcess` die Liste der Prozesse mittels der Methode `GetProcesses()` ermittelt. Diese Liste wird anschließend Eintrag für Eintrag nach einem Fenstertitel abgefragt. Bei Übereinstimmung hat man den gesuchten Prozess gefunden. Über diesen Prozess werden dann die dazugehörigen Threads ermittelt. Jeder Prozess hat mindestens einen Thread, den so genannten Haupt-Thread. Nach Abbruch des Prozesses wird die Liste der Threads an das aufrufende Programm zurückgeliefert. Mit Abbruch des Prozesses werden automatisch alle zu diesem Prozess gehörenden Threads ebenfalls unsanft beendet. Sollte das Abbrechen nicht möglich sein, wird eine Ausnahme ausgelöst, die vom aufrufenden Programm angefangen werden muss.

Ein Beispiel ist in Abbildung 211 zu sehen. Das dazugehörige Programm finden Sie in Listing 401. Das Beispiel-Programm übernimmt den Fenstertitel aus einem Textfeld des Formulars und übergibt dieses der Funktion `KillProcess()`. Bei einer Ausnahme wird der entsprechende Fehlertext in die `ListBox` eingetragen, anderenfalls wird die Liste der Threads der Eigenschaft `Datasource` zugeordnet.

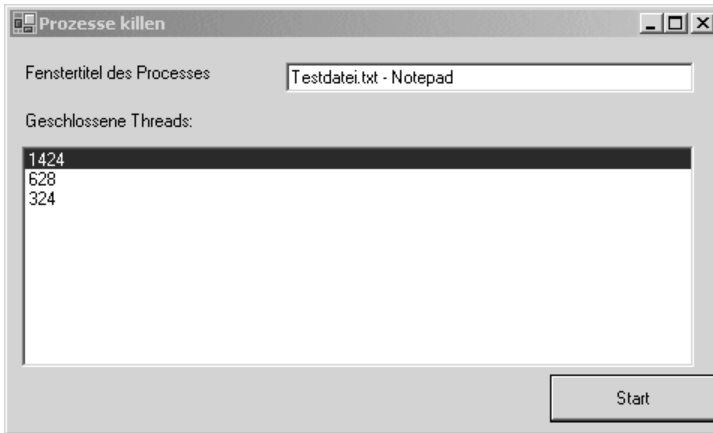


Abbildung 211: Abbruch eines Notepad-Prozesses

```

Private Sub btnStart_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnStart.Click

    Dim ExitThreads As ArrayList

    Try
        ExitThreads = KillProcess(txtName.Text)
    Catch ex As Exception
        ' Statt Fehlerfenster
        lblist.Items.Clear()
        lblist.Items.Add(ex.ToString)
    End Try

    ' Wenn ArrayList nicht nichts ist (also was hat)
    If Not (ExitThreads Is Nothing) Then
        lblist.DataSource = ExitThreads
    End If

End Sub

```

Listing 401: Aufruf der Funktion KillProcess()

## 226 Leistungüberwachung/Performance Counter

Für die Überwachung der Leistungsgrößen (Performance Counter) eines Rechners bietet das Betriebssystem einen gewissen Umfang an Tools an, zum Beispiel den Systemmonitor. Es gibt aber durchaus Situationen, in denen man die Leistungswerte in einem eigenen Programm verarbeiten möchte. Dies ist relativ einfach mit *.NET*-Mitteln zu erreichen.

Zwei Möglichkeiten stehen prinzipiell zur Auswahl, die Integration im VISUAL STUDIO mit dem SERVER-EXPLORER und die Programmierung der *.NET*-Klassen direkt. Da nicht allen ein entsprechendes VISUAL STUDIO zur Verfügung steht (siehe dazu Einleitung und Anhang), sind hier beide Verfahrensweisen aufgezeigt.

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Propor-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

Die Verfahrensweise mit VISUAL STUDIO setzt ein Formular voraus. Dieses Formular muss im lauffähigen Programm nicht unbedingt zu sehen sein. Über den SERVER-EXPLORER sucht man sich unter LEISTUNGSINDIKATOREN den Performancewert, der überwacht werden soll. In Abbildung 212 ist ein Ausschnitt aus dem Bereich Hauptspeicher zu sehen. Mit der linken Maustaste zieht man den gewünschten Überwachungswert in das Formularfenster und benennt ihn um. Letzteres dient der Verständlichkeit des Programmes.

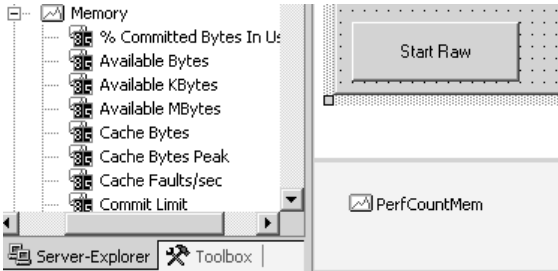


Abbildung 212: Server-Explorer Leistungsindikatoren

Die Bearbeitung geschieht im EIGENSCHAFTEN-Fenster von VISUAL STUDIO. In diesem Beispiel wurde der Name `PerfCountMem` gewählt. In Abbildung 213 ist das entsprechende Fenster zu sehen. Man erkennt, dass der Leistungsmesser den Hauptspeicher überwacht, und zwar die zur Verfügung stehenden Kilobytes. Der vergebene Name wird in Listing 402 benötigt. Die dort dargestellte Funktion `GetPerfCountRAMkB` ermittelt die im Hauptspeicher noch freie Anzahl von Kilobytes in einem mit dem Parameter `Wait` angegebenen Zeittakt in Sekunden. Dies führt die Funktion `Count`-mal durch und liefert dann die Liste der Werte an das aufrufende Programm zurück.

---

```
Public Function GetPerfCountRAMkB(ByVal Wait As Integer, _
    ByVal Count As Integer) As ArrayList

    Dim Arr As ArrayList = New ArrayList
    Dim i As Integer

    For i = 0 To Count
        Arr.Add(PerfCountMem.NextValue)
        Thread.Sleep(Wait * 1000)
    Next
    Return Arr
End Function
```

---

Listing 402: Die Funktion `GetPerfCountRAMkB`

Erreicht wird dies durch den Aufruf der Methode `NextValue` des soeben erstellten `PerfCountMem`-Objektes. `NextValue` liefert den nächsten berechneten Leistungswert ab. Neben dieser Möglichkeit existieren noch `RawValue`, welche einen unbehandelten Rohwert zurückliefert, und `NextSample`, welches einen statistisch ermittelten Wert vom Typ `CounterSample` zurückliefert. Mit diesem Wert kann dann weitere Statistik betrieben werden. Um das Beispiel nicht ausufern zu lassen, beschränkt es sich auf das erwähnte `NextValue`.

Nach der Ermittlung des aktuellen Wertes legt sich der Prozess für die in `Wait` angegebene Anzahl von Sekunden schlafen.





Abbildung 213: Eigenschaften-Fenster

Für die Lauffähigkeit sind zwei Import-Anweisungen von Nöten:

```
Imports System.Threading
Imports System.Diagnostics
```

Hat man die Möglichkeiten des VISUAL STUDIO nicht zur Verfügung, kann man sich die benötigten Leistungsmesser auf eine leicht abgewandelte Form besorgen. Wie, kann man in Listing 403 sehen. Die Bezeichnung Raw im Namen der Funktion deutet nicht auf einen entsprechenden Leistungsermittler hin, wie er oben geschildert wurde. Vielmehr ist hier mit Raw das »reine« .NET gemeint.

```
Public Function GetRawPerfCountRAMKB(ByVal Wait As Integer, _
    ByVal Count As Integer) As ArrayList

    Dim PerfCountMem As System.Diagnostics.PerformanceCounter
    Dim Arr As ArrayList = New ArrayList
    Dim i As Integer

    PerfCountMem = New System.Diagnostics.PerformanceCounter

    PerfCountMem.CategoryName = "Memory"
    PerfCountMem.CounterName = "Available KBytes"
    PerfCountMem.MachineName = "jupiter"

    For i = 0 To Count
        Arr.Add(PerfCountMem.NextValue)
        Thread.Sleep(Wait * 1000)
    Next
    Return Arr
End Function
```

Listing 403: Leistungswerte mit puren .NET: GetRawPerfCountRAMKB

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Propor-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

Die Funktionalität entspricht der aus Listing 402. Einziger Unterschied ist hier die reine .NET-Programmierung. Benötigt wird eine Variable vom Typ `PerformanceCounter` aus dem Namensraum `System.Diagnostics`. In diesem Objekt werden die gewünschte Kategorie, der gewünschte Leistungszähler und der zu überwachende Rechner hinterlegt. In einer Schleife werden dann die Leistungswerte gesammelt und anschließend dem aufrufenden Programm übergeben.

Beide Funktionen arbeiten synchron, was zwangsläufig zur Folge hat, dass das aufrufende Programm die entsprechende Zeit bis zur Beendigung der Funktion wartet. Längere Überwachungen sollte man also asynchron starten.

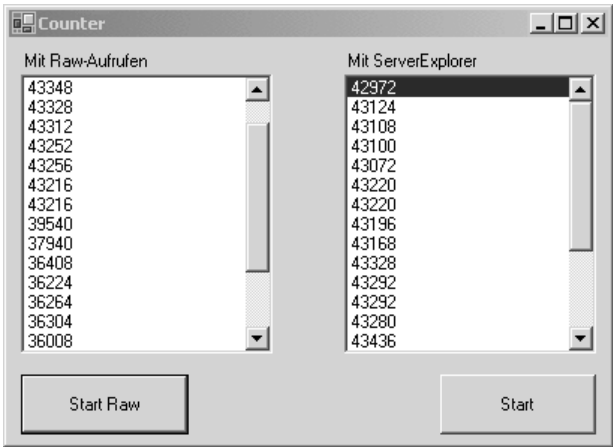


Abbildung 214: Leistungsermittler bei der Arbeit

In Abbildung 214 kann man beide Varianten bei der Arbeit beobachten. Im linken Fenster sieht man die Auswirkungen eines startenden Programmes.

## 227 Registry-Einträge abfragen

Die Registry kann bis zu sieben HKEY\_-Einträge haben, von denen eine Unzahl von Untereinträgen abzweigen. Diese sind in .NET als direkter Zugriff in der `Registry`-Klasse implementiert. So kann man auf den Eintrag `HKEY_LOCAL_MACHINE` mittels `Registry.LocalMachine` zugreifen. Welche Schlüssel möglich sind, ist in Tabelle 31 aufgeführt.

| Schlüssel       | Bedeutung                                             |
|-----------------|-------------------------------------------------------|
| CurrentUser     | Informationen zu Benutzereinstellungen                |
| LocalMachine    | Einstellungen des lokalen Computers                   |
| ClassesRoot     | Informationen zu Typen und Klassen                    |
| Users           | Informationen zur Standardkonfiguration des Benutzers |
| PerformanceData | Leistungsinformationen zu Softwarekomponenten         |
| CurrentConfig   | Benutzerunabhängige Hardwareeinstellungen             |
| DynData         | Dynamische Daten                                      |

Tabelle 31: Die Registry-Schlüssel

Um an die Informationen der Registry zu gelangen, muss der Namensraum `System.Win32.Registry` in das Programm eingebunden werden. Um die Informationen des Teilbaumes `LocalMachine` zu erhalten, wurde die Funktion `GetLMRegistry` implementiert, siehe Listing 404. Diese Funktion übernimmt den kompletten Pfad zum Registry-Schlüssel und liefert eine Liste der Einträge als Hash-Tabelle zurück.

```
' LocalMachine
Public Function GetLMRegistry(ByVal RegKeyString As String) _
    As Hashtable

    Dim RegList As Hashtable
    Dim RegArr() As String
    Dim RegStr As String

    ' Registry hat noch mehr Unterschlüssel, siehe Text
    Dim RegKey As RegistryKey = Registry.LocalMachine

    RegList = New Hashtable

    ' Öffnen des zu untersuchenden Schlüssels, nur Lesen
    RegKey = RegKey.OpenSubKey(RegKeyString, False)

    ' Abruf der Schlüsseleinträge
    RegArr = RegKey.GetValueNames()

    ' Eintragen der Schlüsselwerte in Hashtable
    For Each RegStr In RegArr
        RegList.Add(RegStr.ToString, RegKey.GetValue(RegStr))
    Next

    Return RegList
End Function
```

Listing 404: Die Funktion `GetLMRegistry()`

In der Funktion wird eine Variable vom Typ `RegistryKey` definiert, die mit dem Startwert `Registry.LocalMachine` initialisiert wird. Der zu durchsuchende `SubKey` wird mit der Methode `OpenSubKey` geöffnet, wobei der Zugriff nur lesend erfolgt. Diese Einstellung wird durch den zweiten logischen Parameter erreicht, der angibt, ob auf dem Schlüssel schreibend zugegriffen werden darf. Anschließend werden mit der Methode `GetValueNames()` die Namen der Einträge ermittelt und in einer Schleife der `Hashtable` zugeführt.

Das Ergebnis für einen unbedarften Rechner ist in Abbildung 215 zu sehen. Im angezeigten Registry-Schlüssel finden sich alle die Programme, die beim Start des Rechners ausgeführt werden. Ähnliche Einträge sehen Sie in `RunOnce` und `RunOnceEx`, welche Sie über die `ComboBox` des Beispielprogrammes auswählen können.

Übrigens: In diesen Schlüsseln sammeln sich gerne Viren und Trojaner. Ein Blick könnte nicht schaden.

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Propor-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

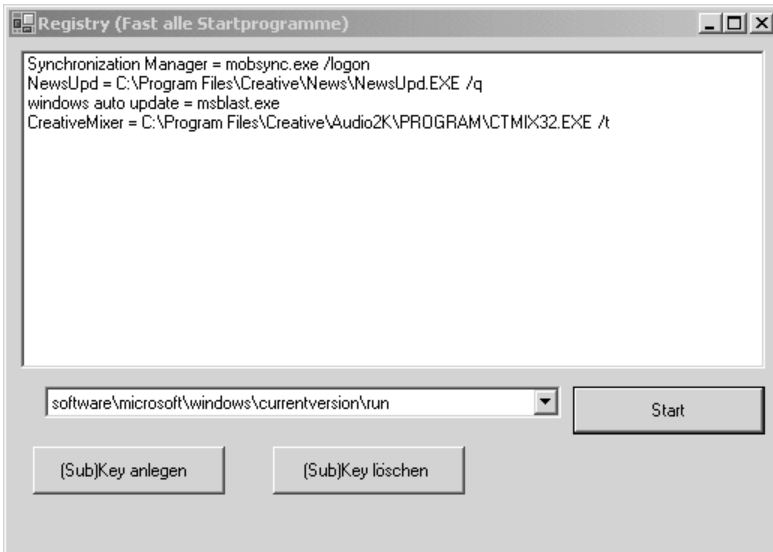


Abbildung 215: Registry-Einträge eines unbedarften Rechners

## 228 Registry-ey anlegen

Eine Funktion zum Anlegen eines Registry-ey's im Bereich CurrentUser ist in Listing 406 dargestellt. Dieser Funktion wird der Schlüsselpfad unterhalb von CurrentUser als Zeichenkette übergeben. Ebenso der Schlüssel selbst und der dazugehörige Wert. Ruft man diese Funktion beispielsweise wie in Listing 405 auf, so wird unterhalb von CurrentUser\Software der Eintrag VB CodeBook erzeugt (falls nicht schon vorhanden) und darunter der Schlüssel Test mit dem Wert 0.

---

```
erg = CreateCURegistry("Software\VB CodeBook", "Test", "0")
```

---

Listing 405: Beispielaufruf der Funktion CreateCURegistry

In der Funktion wird wie in 390 eine entsprechende Variable vom Typ RegistryKey angelegt. Anschließend wird der Pfad mit Schreibrechten geöffnet, welches durch den zweiten Parameter der Methode OpenSubKey garantiert wird. Damit die Funktion fehlerfrei abläuft, muss der Namensraum System.Win32.Registry eingebunden werden.

---

```
Public Function CreateCURegistry(ByVal RegSubKey As String, _
    ByVal RegkeyString As String, ByVal Regvalue As String) _
    As Boolean

    Dim RegKey As RegistryKey = Registry.CurrentUser

    RegKey = RegKey.OpenSubKey(RegSubKey, True)

    If RegKey Is Nothing Then
        RegKey = Registry.CurrentUser.CreateSubKey(RegSubKey)
```

---

Listing 406: De Funktion CreateCURegistry()

```

End If

If Not (RegKey Is Nothing) Then
    RegKey.SetValue(RegkeyString, Regvalue)
Else
    Return False
End If

RegKey.Close()

Return True
End Function

```

Listing 406: De Funktion CreateCURegistry() (Forts.)

Ist der SubKey nicht vorhanden, wird er mittels `CreateSubKey` erstellt, worauf dann Schlüsselname und Schlüsselwert unterhalb des neuen SubKeys angelegt werden. Ob die Funktion beim Erstellen des Registry-Eintrages Erfolg hatte, wird über einen logischen Rückgabewert dem aufrufenden Programm mitgeteilt.

Diese Funktion erstellt nur einen fehlenden SubKey. Sollten mehrere SubKeys im Schlüsselbaum bis zum eigentlichen Eintrag fehlen (fehlende Äste), schlägt die Funktion fehl. Hier bietet sich noch Erweiterungspotential für diese Funktion.

## 229 Registry-Key löschen

Das Löschen eines Registry-Keys erfolgt im Prinzip analog zur Erstellung desselben. Ein Beispiel für den Zweig `HKEY_CURRENT_USER` ist in der Funktion `DeleteCURegistry()` aus Listing 407 realisiert. Auch hier muss der Namensraum `System.Win32.Registry` in das Programm eingebunden werden.

```

Public Function DeleteCURegistry(ByVal RegSubKey As String, _
    ByVal RegkeyString As String) As Boolean

    Dim RegKey As RegistryKey = Registry.CurrentUser

    RegKey = RegKey.OpenSubKey(RegSubKey, True)

    If RegKey Is Nothing Then
        Return False
    End If

    If Not (RegKey Is Nothing) Then
        Try
            RegKey.DeleteValue(RegkeyString)
        Catch ex As Exception
            Throw New ApplicationException("DeleteCURegistry:", ex)
        End Try
    End If

```

Listing 407: Die Funktion DeleteCURegistry()

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Proper-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

---

```

    RegKey.Close()

    Return True
End Function

```

---

*Listing 407: Die Funktion DeleteCURegistry() (Forts.)*

Als Abwandlung wird eine Ausnahme ausgelöst, wenn der zu löschende Eintrag nicht gefunden werden konnte. Die Unterscheidung resultiert aus der Bedeutung der Fehler. Kann man einen Eintrag nicht hinzufügen, ist dies zwar unschön, aber auch noch anderweitig machbar. Geht das Löschen eines Eintrages schief, hat man es wahrscheinlich mit einem größeren Problem zu tun.

## 230 Informationen zum installierten Betriebssystem

Informationen zum installierten Betriebssystem kann man sich über verschiedene Wege besorgen. Die einfachste und schnellste Methode geht sicherlich über das Environment. Um aber an detailliertere Informationen zu kommen, muss man wieder das WMI zu Rate ziehen.

Die einzelnen Funktionen wurden in eine separate Klasse `OpSys` ausgelagert, die als DLL in jedes Programm eingebunden werden kann. Um an die Informationen zu kommen muss eine Instanz dieser Klasse erzeugt werden. Ein Beispiel für die Anwendung dieser Klasse ist in Abbildung 216 zu sehen. Da man Seriennummern nicht frei zur Verfügung stellen sollte, wurde der entsprechende Bereich geschwärzt.

Die Klasse arbeitet mit einigen privaten Variablen, wie sie in Listing 408 aufgeführt sind.

---

```

Public Class OpSys
    Private mOSVersion As String
    Private mOS As OperatingSystem
    Private mVersion As Version
    Private mPlatform As PlatformID
    Private mPlain As New ArrayList
    Dim Searcher As ManagementObjectSearcher
    Dim Query As String
    Dim Computer2Look As String
    Dim ResultCollection As ManagementObjectCollection

```

---

*Listing 408: Private Member der Klasse OpSys*

Wie diese Variablen mit Werten versehen werden, wird im weiteren Verlauf geschildert. Im Vorfeld müssen die Namensräume gemäß Listing 409 bekannt gemacht werden.

---

```

Imports System
Imports System.Management
Imports System.Environment
Imports System.Version

```

---

*Listing 409: Imports-Anweisungen von OpSys*

Die Klasse kennt zwei `New()`-Methoden zur Erzeugung einer Instanz dieser Klasse. Der Default-New-Operator legt als Computer den lokalen Rechner fest, während die zweite New-Methode mit dem Namen eines Rechners aufgerufen wird (siehe Listing 410).

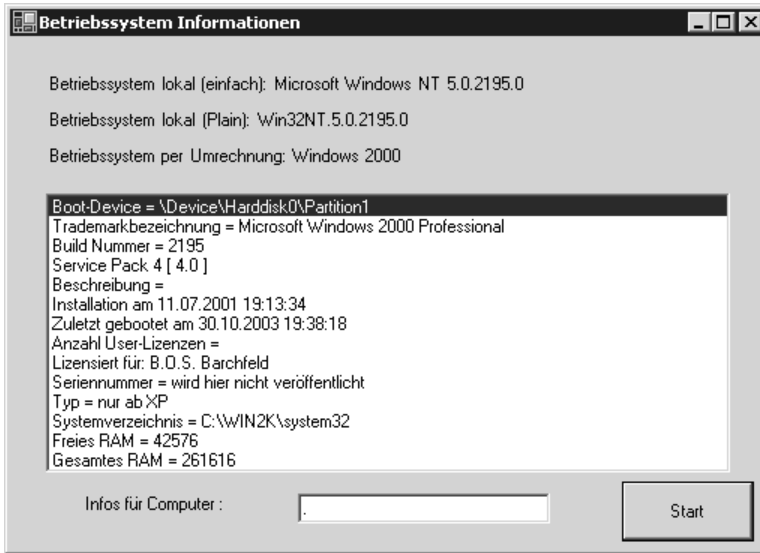


Abbildung 216: Informationen zum installierten Betriebssystem, Beispiel

```
Public Sub New()
    mOS = Environment.OSVersion
    mOSVersion = mOS.ToString()
    mVersion = mOS.Version()
    mPlatform = mOS.Platform()
    Computer2Look = "."
End Sub

Public Sub New(ByVal ComputerName As String)
    mOS = Environment.OSVersion
    mOSVersion = mOS.ToString()
    mVersion = mOS.Version()
    mPlatform = mOS.Platform()
    Computer2Look = ComputerName
End Sub
```

Listing 410: New()-Methoden von OpSys

Aus dem Environment werden die entsprechenden Informationen geholt. In der Variablen `mOS` wird ein Objekt der Klasse `OperatingSystem` abgespeichert. Der Einfachheit halber wird die Zeichenkettendarstellung dieser Variablen in der Variablen `mOSVersion` abgelegt. Mit diesen Werten kann die erste Variante einer Betriebssystem-Ermittlung durchgeführt werden. Sie ist in Listing 411 zu sehen. Es wird nur die ermittelte Zeichenkette des Environments als Eigenschaft zur Verfügung gestellt. Das Ergebnis ist in Abbildung 216 als erstes Label hinter »Betriebssystem lokal (einfach):« dargestellt.

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Proper-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schie-  
denes

---

```

ReadOnly Property OS() As String
    Get
        Return mOSVersion
    End Get
End Property

```

---

*Listing 411: Die Eigenschaft OS von OpSys*

In der Variablen `mOS` sind die einzelnen Teile der Betriebssystemkennung versteckt. Diese können zusammen mit den Informationen über die Plattform mittels der Eigenschaft `OPPlain` abgerufen werden. Die Realisierung dieser Eigenschaft ist Listing 412 zu entnehmen.

---

```

ReadOnly Property OSPlain() As ArrayList
    Get
        mPlain.Add(mPlatform.ToString())
        mPlain.Add(mVersion.Major.ToString())
        mPlain.Add(mVersion.Minor.ToString())
        mPlain.Add(mVersion.Build.ToString())
        mPlain.Add(mVersion.Revision.ToString())
        Return mPlain
    End Get
End Property

```

---

*Listing 412: Die Eigenschaft OSPlain von OpSys*

Das vom Beispiel-Programm zusammengesetzte Ergebnis findet sich im zweiten Label aus Abbildung 216.

Alle diese Informationen enthalten zwar auch den Namen des Betriebssystems (also beispielsweise Windows 2000 oder XP), aber für den ungeübten Benutzer eher nicht erkennbar. Um aus diesem Zahlenkonglomerat den Namen des Betriebssystems zu generieren wurde die Eigenschaft `GetOSEnv` programmiert. Sie basiert auf einer Tabelle, die im MSDN veröffentlicht wurde, und ist recht aktuell. Der letzte Eintrag sollte unter .NET allerdings nicht auftreten.

---

```

ReadOnly Property GetOSEnv() As String
    Get
        Select Case mPlatform
            Case PlatformID.Win32NT
                If mVersion.Major = 4 Then
                    Return "NT 4.0"
                End If

                Select Case mVersion.Minor
                    Case 0
                        Return "2000"
                    Case 1
                        Return "XP"
                    Case 2
                        Return "Server 2003"
                End Select
        End Select
    End Get
End Property

```

---

*Listing 413: Die Eigenschaft GetOSEnv von OpSys*



```

Case PlatformID.Win32Windows
    If mVersion.Minor = 10 Then
        Return "98"
    Else
        Return "Me"
    End If

Case PlatformID.WinCE
    Return "CE"
Case PlatformID.Win32S
    Return "Windows 16 Bit ????"
End Select
End Get
End Property

```

Listing 413: Die Eigenschaft *GetOSEnv* von *OpSys* (Forts.)

Die Funktion liefert die Windows-Version ohne den Vorsatz »Windows«. Das Ergebnis für das Beispielprogramm ist in der dritten Label-Zeile von Abbildung 216 zu erkennen.

Alle bis hierher vorgestellten Methoden liefern nur das Ergebnis für den lokalen Rechner. Benötigt man Angaben zu anderen Rechnern im Netz, führen diese Methoden zu keinem Ergebnis. Hier muss in die Trickkiste des WMI gegriffen werden.

Dies kann mit der Eigenschaft *GetOSInformation* durchgeführt werden. Sollten die Informationen eines entfernten Rechners abgefragt werden, muss das Objekt der Klasse *OpSys* selbstverständlich mit der entsprechenden *New*-Methode initialisiert worden sein.

```

ReadOnly Property GetOSInformation() As ArrayList
    Get
        Return OSInformation(Computer2Look)
    End Get
End Property

```

Listing 414: Die Eigenschaft *GetOSInformation*

Die Eigenschaft *GetOSInformation* reicht den zwischengespeicherten Namen des zu untersuchenden Rechners an die *private* Funktion *OSInformation()* durch, die Sie in Listing 415 abgedruckt finden.

```

Private Function OSInformation(ByVal Computer As String) _
    As ArrayList

    Dim Obj As ManagementObject
    Dim List As ArrayList = New ArrayList
    Dim Scope As String
    Dim OSTyp As Integer
    Dim OSTypString As String

```

Listing 415: Die *private* Funktion *OSInformation()*

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

```

' Für welchen Computer werden Informationen benötigt?
If Computer = "." Then
    Scope = "root\cimv2"
Else
    Scope = "\\" + Computer + "\root\cimv2"
End If

' Abfragestring
Query = "Select * from Win32_OperatingSystem"

' Abfrageobjekt für den Computer generieren
Searcher = New ManagementObjectSearcher(Scope, Query)

' Informationen in Collection holen
ResultCollection = Searcher.Get()

' Gewünschte Informationen in ArrayList abspeichern
For Each Obj In ResultCollection
    List.Add("Boot-Device = " + Obj("BootDevice"))
    List.Add("Trademarkbezeichnung = " + Obj("Caption"))
    List.Add("Build Nummer = " + Obj("BuildNumber"))
    ' Service Pack Version
    List.Add(Obj("CSDVersion") + " [ " + _
        Obj("ServicePackMajorVersion").ToString + "." + _
        Obj("ServicePackMinorVersion").ToString + " ]")

    List.Add("Beschreibung = " + Obj("Description"))
    List.Add("Installation am " + _
        ToDateTime(Obj("InstallDate")).ToString)
    List.Add("Zuletzt gebootet am " + _
        ToDateTime(Obj("LastBootUpTime")).ToString)
    List.Add("Anzahl User-Lizenzen = " + _
        Obj("NumberOfLicensedUsers"))
    List.Add("Lizensiert für: " + Obj("Organization"))
    List.Add("Seriennummer = " + Obj("SerialNumber"))

    ' Ab XP möglich
    Dim ver As String = Obj("Version")
    If Obj("Version") >= "5.1" Then
        OSTyp = Obj("ProductType")
        Select Case OSTyp
            Case 1
                OSTypString = "Work Station"
            Case 2
                OSTypString = "Domänen Controller"
            Case 3
                OSTypString = "Server"
        End Select
        List.Add("Typ (XP / Server 2003) = " + OSTypString)
    Else

```

---

```

        List.Add("Typ = " + "nur ab XP")
    End If

    List.Add("Systemverzeichnis = " + Obj("SystemDirectory"))
    List.Add("Freies RAM = " + Obj("FreePhysicalMemory").ToString)

    ' Für das Betriebssystem sichtbares RAM
    List.Add("Gesamtes RAM = " + _
        Obj("TotalVisibleMemorySize").ToString)
Next
Return List
End Function

```

Listing 415: Die private Funktion `OSInformation()` (Forts.)

In der Funktion wird zuerst festgelegt, welcher Computer analysiert werden soll. Dies kann mit einem `ManagementScope`-Objekt durchgeführt werden. Da eine WQL-Abfrage gestellt werden soll, ergibt sich eine Abkürzung. In einem der Konstruktoren der `ManagementObjectSearcher`-Klasse können Abfrage und Bereich (Scope) als Zeichenketten übergeben werden. Mit der `Get()`-Methode des erstellten Objektes wird die Klassen-Variable `ResultCollection` gefüllt. In der anschließenden Schleife werden die gewünschten Informationen in ein `ArrayList` verpackt, welches zum Schluss der aufrufenden Funktion zurückgegeben wird.

In der Schleife wird die Funktion `ToDateTime` angewandt, die man in der Dokumentation zu *.NET* nicht finden wird. Diese Funktion wird bereitgestellt, wenn man mit dem `SERVER-EXPLORER` eine Klasse aus `MANAGEMENT CLASSES / OPERATING SYSTEM` generiert. Sie wandelt das interne Datumsformat in den Datentyp `DateTime` um. In Abbildung 196 kann man die Darstellung dieses internen Datumsformates sehen. In dem dazugehörigen Kapitel wird auch eine alternative Möglichkeit aufgezeigt, um an diese Funktion zu gelangen.

Die Eigenschaft `TotalVisibleMemorySize` trägt mit Absicht nicht den Namen `TotalMemorySize`, da es Windows-Betriebssysteme gibt, die nicht den gesamten eingebauten Hauptspeicher erkennen können. Diese Größe stellt aber den für das Betriebssystem nutzbaren Hauptspeicher dar, ist also auch in einem solchen Fall eine nutzbare Information.

Ab Betriebssystemversion 5.1 (XP, Server 2003) können weitere Informationen abgerufen werden. Dies wird in der `if`-Abfrage der Funktion berücksichtigt.

Ein Beispiel für diese Abfrage ist in der Listbox von Abbildung 216 zu sehen.

## 231 Prozessorgeschwindigkeit

Neben den bisher bereits ermittelten Informationen zu einem Rechner fehlt noch der zentrale Teil, die CPU. Dies soll nun nachgeholt werden. Die benötigten Angaben werden wieder mittels WMI ermittelt. Dazu wird mit dem `SERVER-EXPLORER` eine Klasse aus `MANAGEMENT CLASSES / PROCESSORS` generiert zusammen mit weiteren Namensräumen in das Projekt importiert, siehe Listing 416. Die so generierte Klasse trägt den Namen `Processor`.

```

Imports System
Imports System.Management
Imports ProcessorSpeed.ROOT.CIMV2

```

Listing 416: Imports-Anweisungen

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Propert-  
yGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

Welche Angaben zum Prozessor beispielhaft möglich sind, kann der Abbildung 217 entnommen werden. In der Abbildung sind auch die Informationen aus den Kapiteln 11.34 – 11.36 zu sehen. Die Taktfrequenz des Prozessors wird mit der Funktion `GetProcessorSpeed` aus Listing 417 ermittelt.

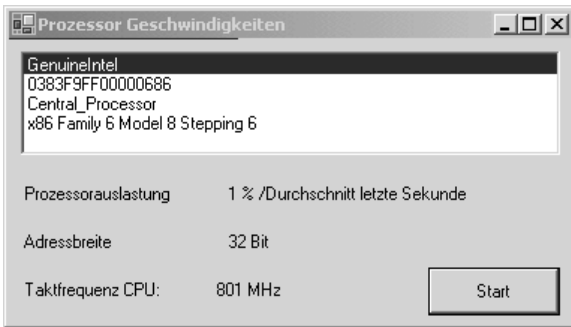


Abbildung 217: Prozessor-Informationen

In der Funktion wird eine Variable vom Typ `Processor` deklariert, die anschließend mit den Werten des ersten Prozessors im Rechner definiert wird. Ist ein zweiter Prozessor auf dem Motherboard, kann man zumindest davon ausgehen, dass dieser die gleiche Taktfrequenz und Bitbreite hat.

Mit der Abfrage auf `IsCurrentClockSpeedNull` wird sichergestellt, dass die benötigte Information ermittelt werden kann. Es handelt sich hier um eine Methode der Klasse `Processor`. Die ermittelte Taktfrequenz wird zur einfacheren Weiterverarbeitung in den Typ `Integer` konvertiert. Da die Taktfrequenz in MHz ermittelt wird, dürfte in nächster Zeit kein Problem mit dieser Konvertierung auftreten.

---

```
Public Function GetProcessorSpeed() As Integer
    Dim proc As Processor
    Dim Speed As UInt32
    Dim SpeedInt As Integer

    ' Instanz für den 1. Prozessor
    proc = New Processor("CPU0")
    If proc.IsCurrentClockSpeedNull = False Then
        ' CurrentClockSpeed ist UInt32
        Speed = proc.CurrentClockSpeed()
        SpeedInt = Convert.ToInt32(Speed)
    Else
        SpeedInt = -1
    End If

    Return SpeedInt
End Function
```

---

Listing 417: Die Funktion `GetProcessorSpeed()`

Sollte die Taktfrequenz nicht abrufbar sein, liefert die Funktion den Wert  $-1$  zurück. Durch Überladung des `New`-Konstruktors können auch die Werte für andere Computer abgefragt werden.

## 232 Prozessorauslastung

Die Ermittlung der Prozessorauslastung geschieht analog zu Kapitel [11.33](#). Die Eigenschaft `LoadPercentage` der Klasse `Processor` liefert die gemittelte Auslastung des Prozessors in der letzten Sekunde zurück. Der Zeitrahmen kann nicht verändert werden.

```
Public Function GetProcessorLoad() As Integer
    Dim proc As Processor
    Dim Load As Integer

    proc = New Processor("CPU0")

    If proc.IsLoadPercentageNull = False Then
        Load = Convert.ToInt32(proc.LoadPercentage)
    Else
        Load = -1
    End If

    Return Load
End Function
```

Listing 418: Die Funktion `GetProcessorLoad`

Auch liefert die Funktion  $-1$  zurück, sollte die Angabe der Prozessorauslastung nicht ermittelt werden können. Eine Beispielanwendung dieser Funktion ist in Abbildung 217 zu sehen.

Durch Überladung des `New`-Konstruktors können auch hier die Angaben von anderen Computern abgefragt werden.

## 233 Bitbreite des Prozessors

Das Vorgehen in dieser Funktion entspricht exakt dem der beiden letzten Kapitel, so dass hier nicht näher darauf eingegangen werden muss.

```
Public Function GetProcessorWidth() As Integer
    Dim proc As Processor
    Dim ProcWidth As Integer

    proc = New Processor("CPU0")

    If proc.IsAddressWidthNull = False Then
        ProcWidth = Convert.ToInt32(proc.AddressWidth)
    Else
        ProcWidth = -1
    End If

    Return ProcWidth
End Function
```

Listing 419: Die Funktion `GetProcessorWidth`

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

## 234 Prozessor-Informationen

Die Klasse `Processor` umfasst über 40 Eigenschaften. Einige wurden schon in den vorherigen Kapiteln als eigenständige Funktionen abgefragt. Einige Angaben von mehr allgemeinem Interesse sind in der Funktion `GetProcessorInfos` zusammengefasst, die eine `ArrayList` von Zeichenkettenwerten zurückliefert.

---

```
Public Function GetProcessorInfos() As ArrayList
    Dim proc As Processor
    Dim ProcList As ArrayList = New ArrayList
    Dim tmp As UInt16

    proc = New Processor("CPU0")

    ProcList.Add(proc.Manufacturer)
    ProcList.Add(proc.ProcessorId)
    ProcList.Add(proc.ProcessorType)
    ProcList.Add(proc.Description)

    Return ProcList
End Function
```

---

*Listing 420: Die Funktion `GetProcessorInfos`*

Für die in Listing 420 ermittelten Werte existieren in der Klasse `Processor` keine Methoden der Form `Is...Null`, so dass davon ausgegangen werden kann, dass diese Eigenschaften immer einen Wert zurückliefern. Ein Beispiel für die so ermittelten Werte ist in Abbildung 217 zu sehen.

## 235 SMTP – eMail

Unter gewissen Umständen erweist es sich als sehr praktisch, wenn man aus dem eigenen Programm eMails verschicken kann. Auch dieses Problem lässt sich mit *.NET* recht übersichtlich lösen. Die hier vorgestellte Methode hat allerdings einen Nachteil. Es reicht nicht, auf dem eigenen Rechner oder irgendwo im Netz einen SMTP-Server zu haben. Es muss der INTERNET INFORMATION SERVER von Microsoft sein. Zusätzlich muss im IIS der virtuelle SMTP-Server aktiviert sein. Hier ist zwischen dem SMTP-Dienst des Rechners und dem genannten virtuellen SMTP-Server des IIS zu unterscheiden. Ist der Dienst gestartet, der virtuelle SMTP Server des IIS aber nicht, können mit dieser Funktion keine eMails verschickt werden. Dienst und Server müssen aktiv sein.

Zusätzlich müssen die Rechte des IIS-SMTP-Servers richtig eingestellt sein. Diese können über den EIGENSCHAFTEN-Dialog des virtuellen SMTP-Servers eingestellt werden.

Sind diese Voraussetzungen erfüllt, steht dem Versenden von eMails mit der Funktion `SendEmail` aus Listing 421 nichts mehr im Wege.

---

```
Private Sub SendEmail(ByVal ToAddress As String, _
    ByVal FromAddress As String, _
    ByVal CCAddress As String, _
    ByVal BCCAddress As String, _
```

---

*Listing 421: Die Funktion `SendEmail`*

```

ByVal SMTPServer As String, _
ByVal AppendFile As String, _
ByVal MessageText As String, _
ByVal SubjectText As String)

Dim Message As MailMessage = New MailMessage
Dim Attach As MailAttachment

Message.To = ToAddress
Message.From = FromAddress
Message.Subject = SubjectText
If CCAccess <> Nothing Then
    Message.Cc = CCAccess
End If

If BCCAddress <> Nothing Then
    Message.Bcc = BCCAddress
End If

Message.Body = MessageText

If AppendFile.Trim <> String.Empty Then
    Attach = New MailAttachment(AppendFile)
    Message.Attachments.Add(Attach)
End If

Try
    SmtpMail.SmtpServer = SMTPServer
    SmtpMail.Send(Message)
Catch ex As System.Web.HttpException
    txtText.Text = ex.Message
End Try
End Sub

```

Listing 421: Die Funktion SendEMail (Forts.)

Die Funktion benötigt den Namensraum `System.Web.Mail`, der mittels Imports eingebunden werden muss. Der Funktion werden die benötigten Informationen zum Versenden einer Mail als Zeichenketten übergeben. Die Funktion unterstützt in der dargestellten Form nur eine Datei als Anhang. Will man mehrere Dateien versenden, bietet sich hier die Möglichkeit, diese Funktion zu erweitern.

Nach dem Erstellen einer Variablen `Message` vom Typ `MailMessage` werden die zum Versand erforderlichen Angaben diesem Objekt übergeben. Die Eigenschaften entsprechen den typischen eMail-Angaben und dürften damit selbsterklärend sein.

Ein möglicher Anhang wird der Funktion als Dateiname übergeben. Mit diesem Dateinamen wird ein Objekt vom Typ `MailAttachment` erzeugt, welches dann dem Objekt `Message` mit der `Add`-Methode der `Attachments`-Eigenschaft angehängt wird.

Die Eigenschaft `SmtpServer` der Klasse `SmtpMail` aus `System.Web.Mail` ist `Shared`, so dass keine eigenständige Instanz erzeugt werden muss. Dieser wird der SMTP-Server übergeben. Die Angabe des Servers kann eine IP-Adresse sein, kann aber auch der Domain-Angabe im virtuellen SMTP-Server entsprechen. Diese Domain hat nichts mit der Domäne eines Windows-Netzwerkes zu tun.

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Proper-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

Die in der Funktion zusammengestellte `Message` wird dann mit der `Send`-Methode von `SmtpMail` verschickt. Die so erzeugte Mail wird als `.em`-Datei im Verzeichnis `Inetpub\mailroot\Drop` des IIS zwischengespeichert.

## 236 Fax senden

Zu den modernen Kommunikationsmitteln gehört neben der eMail sicherlich auch das Fax. Ab Windows 2000 ist im Betriebssystem ein entsprechender Dienst vorhanden, den man auch aus eigenen Programmen heraus ansteuern kann. Diese Funktionalität steckt in der `faxcom.dll` im `System32`-Verzeichnis des Windows-Betriebssystems. Dieses muss per Verweis in das Projekt eingebunden werden.

---

```
Public Structure Recipient
    Public Name As String
    Public Title As String
    Public Department As String
    Public FaxNo As String
    Public Company As String
    Public Zip As String
    Public City As String
    Public Street As String
End Structure
```

---

*Listing 422: Empfängerstruktur zum Fax*

Da der Absender bei Einsatz dieser Funktion nicht häufig wechseln wird, wurden die Absenderangaben der Einfachheit halber in die Funktion gepackt. Der Empfänger wird mittels einer Struktur übergeben. Deren Aufbau ist in Listing 422 zu sehen. Der Funktion `SendFax` aus Listing 423 wird neben dieser Struktur der zu sendende Dateiname inklusive Pfad übergeben. Diese Datei sollte natürlich druckbar sein.

Durch den Verweis auf die DLL werden mehrere neue Klassen generiert, die direkt zu Beginn der Funktion zum Einsatz kommen: `FaxServer` und `FaxDoc`. Nachdem das entsprechende Server-Objekt `Server` erzeugt wurde, wird mit dem Dateinamen des zu sendenden Dokumentes ein Objekt vom Typ `FaxDoc` erstellt. Die Methode `CreateDocument` liefert den Typ `Object` zurück, so dass für `Strict On` eine Typkonvertierung erforderlich wird.

---

```
Public Function SendFax(ByVal FileName As String, _
    ByVal RecStruc As Recipient) As Boolean

    Dim Server As FaxServer
    Dim Doc As FaxDoc

    Try
        Server = New FaxServer
        Doc = CType(Server.CreateDocument(FileName), FaxDoc)
        Server.Connect("\\Jupiter")
        Server.Retries = 5

        ' Absender Angaben
```

---

*Listing 423: Die Funktion SendFax*



```

Doc.FaxNumber = RecStruc.FaxNo
Doc.SenderAddress = "Beispielstrasse 3, 22335 HH"
Doc.SenderCompany = "ACME GmbH"
Doc.SenderDepartment = "IT"
Doc.SenderFax = "+49 815 987654"
Doc.SenderName = "Mustermann"

' Empfänger Angaben
Doc.RecipientName = RecStruc.Name
Doc.RecipientTitle = RecStruc.Title
Doc.RecipientDepartment = RecStruc.Department
Doc.RecipientCompany = RecStruc.Company
Doc.RecipientZip = RecStruc.Zip
Doc.RecipientCity = RecStruc.City
Doc.RecipientAddress = RecStruc.Street
Doc.FaxNumber = RecStruc.FaxNo

' Fax Senden
Doc.Send()

' Aufräumen
Doc = Nothing
Server.Disconnect()
Server = Nothing

Catch ex As SystemException
    Throw New Exception("SendFax-Function: ", ex)
End Try

Return True
End Function

```

Listing 423: Die Funktion SendFax (Forts.)

Nach Verbindung mit dem Fax-Server (hier auch »hart verdrahtet«) wird die Anzahl der Verbindungsversuche auf 5 eingestellt. Die Daten für Sender und Empfänger werden dem Objekt für das Fax-Dokument (Doc) zugeordnet und anschließend mit der Send-Methode verschickt. Die Funktion kehrt sofort zurück und wartet nicht auf die Beendigung des Faxvorganges. Sollte keine aktive Telefonleitung vorhanden sein, wird das Fax automatisch vom Betriebssystem in den Fax-Queue verschoben und dort so lange zwischengespeichert, bis ein Versand durchgeführt oder das Dokument aus der Queue gelöscht wurde. Sind die Einstellungen des Fax-Monitors entsprechend, erscheint ein Fenster wie in Abbildung 218 gezeigt beim Versand des Faxes.

Hat der Rechner ständigen Kontakt zu einer Telefonleitung, erscheint dieses Fenster mitten in Ihrem Programm. Dies kann man unter FAX-EIGENSCHAFTEN / STATUS MONITOR deaktivieren.

## 237 Logon-Sessions mit XP

Unter den neueren Windows-Versionen sind die Möglichkeiten des WMI nochmals erweitert worden. So können Sie sich detaillierte Informationen über eingeloggte Benutzer zusammenstellen. Auf der Client-Seite wird Windows XP, auf der Serverseite Windows Server 2003 vorausgesetzt.

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

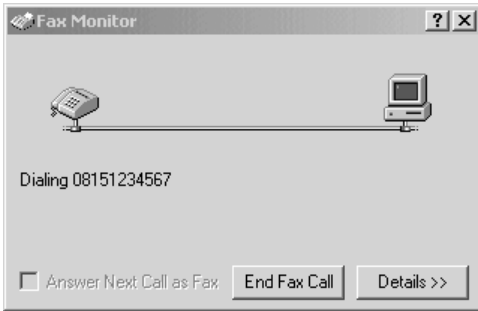


Abbildung 218: Fax-Monitor beim Versenden eines Faxes

Eine Möglichkeit, diese neuen Features abzufragen, wird in Listing 424 mittels einer WQL-Abfrage gelöst. Es handelt sich um eine klassische WQL-Abfrage, wie sie in den vorhergehenden Kapiteln schon eingesetzt wurde. Eine nähere Erklärung kann deshalb an dieser Stelle entfallen.

---

```
Private Sub btnStart_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnStart.Click

    Dim mQuery As WqlObjectQuery
    Dim mSearch As ManagementObjectSearcher
    Dim mCol As ManagementObject
    Dim mStrSQL As String

    mStrSQL = "Select * from win32_LogonSession"
    mQuery = New WqlObjectQuery(mStrSQL)
    mSearch = New ManagementObjectSearcher(mQuery)
    For Each mCol In mSearch.Get()
        lblList.Items.Add(mCol("Caption"))
        lblList.Items.Add(mCol("Description"))
        lblList.Items.Add(mCol("LogonId"))
        lblList.Items.Add(mCol("LogonType"))
        lblList.Items.Add(mCol("Name"))
        lblList.Items.Add(mCol("AuthenticationPackage"))
    Next

End Sub
```

---

Listing 424: LogonSessions unter XP Server 2003 abfragen

Die Eigenschaft `LogonType` kennt zwölf verschiedene Varianten, wie eine `LogonSession` angemeldet sein kann. Näheres kann man in der Online-Hilfe von VISUAL STUDIO unter dem Stichwort `Win32_LogonSession` erfahren.

# Datenbanken

In den folgenden Rezepten geht es um die Verwaltung des Microsoft SQL-Servers bzw. um die kostenfrei verteilbare Version, die Microsoft Database Engine, MSDE. Alle Rezepte sind in einer DLL implementiert, so dass man die hier vorgestellten Rezepte sofort in den eigenen Programmen einsetzen kann. Sie dienen zugleich als eine Einführung in die Thematik der Database Management Objects (DMO), die in beiden Versionen der Datenbank mitgeliefert werden und so in eigenen Programmen verwandt werden können. Dies hat vor allem bei der MSDE einen großen Vorteil, da diese Version ohne jegliche Verwaltungssoftware ausgeliefert wird. Die Möglichkeiten sind aber auch so umfassend, dass sie den Rahmen dieser Sammlung sprengen würden.

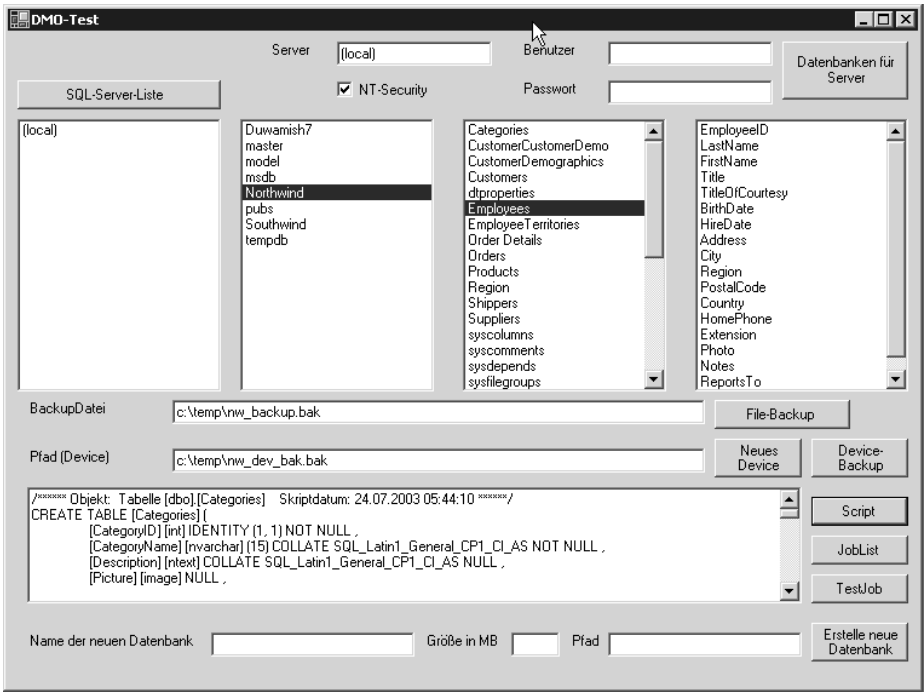


Abbildung 219: Beispielprogramm zu den Möglichkeiten der DMO-DLL

Das Beispielprogramm auf der CD zeigt exemplarisch den Aufruf der einzelnen Methoden der hier vorgestellten Klassen. Die DMO-Funktionen stammen alle aus einer Zeit vor .NET. Die hier entwickelte verwaltete DLL stellt im Wesentlichen Wrapper-Klassen für die SQLDMO.DLL zur Verfügung, die sich im Detail von der klassischen Einbindung unterscheiden.

Die SQLDMO befindet sich typischerweise unter C:\Programme\Microsoft SQL Server\80\Tools\bin\sqlcmdmo.dll. Nach dem Hinzufügen eines entsprechenden COM-Verweises werden diese Funktionen über

eingebunden.

Die Rezepte 12.1 bis 12.18 beziehen sich auf die Benutzung des Database Management Objects. In den darauf folgenden Kapiteln wird auf die Thematik der Datenbanken mit der Zielrichtung ADO.NET eingegangen. Mit Rezepten über ADO.NET könnte man ebenfalls ein Buch füllen, und da es mittlerweile auch gute Literatur für Anfänger und Fortgeschrittene über dieses Thema gibt, haben wir nur ein paar Rezepte in dieses Buch aufgenommen. So war uns das Kapitel Indizierung wichtig, da es dort immer noch viel Le(e|h)rmaterial aus der Theorie gibt. Auch der Zugriff auf Excel-Tabellen spielt im täglichen DV-Betrieb eine große Rolle, doch findet man den Zugriff auf Excel mit OleDb recht selten in der Literatur.

## 238 Erreichbare SQL-Server

Die Frage nach den überhaupt zur Verfügung stehenden Datenbanken kann nicht nur bei der Sicherung von Datenbanken eine wichtige Rolle spielen. Diese Fragestellung kann mit den Hilfsmitteln der DMO recht zügig beantwortet werden:

---

```
' Ermittelt die erreichbaren SQL-Server
Shared Public Function ListSQLServers() As ArrayList
    Dim SrvGrp As SQLDMO.ServerGroup
    Dim mApp As SQLDMO.Application = New SQLDMO.Application()
    Dim bGroupUse As Boolean
    Dim i As Integer
    Dim j As Integer

    Try
        bGroupUse = mApp.UseCurrentUserServerGroups
    Catch e As System.Exception
        Throw New System.ArgumentException( _
            "DMO: ServerGruppe nicht erreichbar.", e)
    End Try

    ' Liste der erreichbaren SQL-Server erstellen
    mServerList = New ArrayList()

    ' Schleife über alle erreichbaren Servergruppen
    For i = 1 To mApp.ServerGroups.Count
        SrvGrp = mApp.ServerGroups.Item(i)

        ' Schleife über alle Server der aktuellen Gruppe
        For j = 1 To SrvGrp.RegisteredServers.Count
            mServerList.Add _
                (SrvGrp.RegisteredServers.Item(j).Name.ToString)
        Next

    Next

    Return mServerList
End Function
```

---

Listing 425: SQL-Serverliste

Um die registrierten Server zu ermitteln, wird kein Benutzer zum Login auf dem SQL-Server benötigt. Da die weiteren Methoden dies voraussetzen (s.u.), wird diese Methode `Shared` definiert. Für die weitere Verwendung sind nur die Namen der Datenbanken von Interesse, so dass die Methode `ListSQLServers()` eine `ArrayList` aus Zeichenketten zurückliefert. `ServerGroup` ist ein `User Registry Key`, der eine Servergruppe darstellt. Diese Gruppen werden bei der Installation erstellt oder können mittels Programm erstellt werden.

Um Informationen über die Server zu bekommen, muss eine Instanz des DMO-Anwendungsobjektes vorhanden sein. Diese wird in der zweiten Zeile nach dem Funktionskopf für die Behandlung des Server-Buttons aus dem Beispielprogramm erstellt. Nach einem Check der User-Einstellung werden die registrierten Servergruppen durchlaufen und für jede Datenbank der Name in der `ArrayList` eingetragen und schlussendlich dem aufrufenden Programm zurückgeliefert.

Im Beispielprogramm befindet sich der Button für den Aufruf oberhalb der linken `ListBox`:

---

```
Private Sub btnServer_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnServer.Click

    Dim mServerList As ArrayList = New ArrayList()
    Dim Search As DMO.NET.DMOServer = New DMO.NET.DMOServer()

    mServerList = Search.ListSQLServers
    lbServer.Items.Clear()
    lbServer.Items.AddRange(mServerList.ToArray)
End Sub
```

---

*Listing 426: Aufruf der Serverliste im Beispielprogramm*

## 239 Default-Anmeldung am SQL-Server

Um die Datenbank verwalten zu können, muss man sich mit den entsprechenden Rechten in der ausgewählten Datenbank anmelden. Dafür gibt es zwei unterschiedliche Methoden: das Anmelden mit der Berechtigungsverwaltung des SQL-Servers oder mit den Rechten des Betriebssystem-Benutzers, der so genannten NT-Security.

Die Anmeldung wird über verschiedene Überladungen der `New`-Prozedur erreicht. Die einfachste, gefährlichste und sehr häufig funktionierende Variante ist die Anmeldung mittels des Standardbenutzers auf der lokalen Datenbank:

---

```
Public Sub New()
    ' Default-Einstellungen des SQL-Servers (Sicherheitsrisiko)
    mUser = "sa"
    mPWD = ""
    mSecurity = False
    Create("(localhost)")
End Sub
```

---

*Listing 427: Anmeldung mittels Standarduser*

Die Variablen `mUser`, `mPWD` und `mSecurity` sind als `private` Variablen der Klasse deklariert. `mSecurity` spiegelt das oben geschilderte Anmeldeverfahren wider, wobei `mSecurity = True` für das NT-Secu-

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

rity-Anmeldeverfahren steht. Der Benutzer »sa« ohne Kennwort wird standardmäßig bei der Installation des SQL-Servers angelegt. Wird dies nicht geändert, kann jeder als Administrator der Datenbank aktiv werden, der diesen Zugang kennt! Daher auch die Anmerkung, dass die Möglichkeit dieses Anmeldeverfahrens mit Gefahren verbunden ist, nicht nur im hier betrachteten Kontext.

Anschließend werden die weiteren Schritte der Anmeldung einer Prozedur `create` übergeben, da sich diese Schritte in den verschiedenen Verfahren nicht unterscheiden.

---

```
Private Sub Create(ByVal srv As String)
    Try
        mServerString = srv
        mApp = New SQLDMO.Application()
        mServer = New SQLDMO.SQLServer()
        mDB = New SQLDMO.Database()
        mDBList = New ArrayList()
        mTableList = New ArrayList()
        mFieldList = New ArrayList()
    Catch e As System.Exception
        Throw New System.ArgumentException("DMO.NET.create: ", e)
    End Try
End Sub
```

---

*Listing 428: create()-Prozedur der Anmeldung*

Die in dieser Prozedur gesetzten Variablen sind private Variablen der Klasse und werden im späteren Verlauf in den verschiedenen Methoden benötigt. Der Servername wird dieser Prozedur übergeben und in einer `String`-Variablen gespeichert. Für den Zugriff auf die Datenbank werden für jede SQLDMO-Hierarchiestufe eine Variable definiert. `mDBList` nimmt die Namen der Datenbanken des betrachteten Servers auf, `mTableList` die Tabellen der Datenbank und `mFieldList` die Feldnamen der entsprechenden Tabelle.

Damit sind alle Angaben bekannt, um sich mit dem Datenbankserver zu verbinden. Dies geschieht mittels der Methode `ConnectSQL()`. Die Funktionalität wurde in zwei Methoden aufgeteilt.

---

```
' Verbindet zum Server mit SQL-Berechtigung
Public Sub ConnectSQL()
    mServer.LoginSecure = False
    Try
        mServer.Connect(mServerString, mUser, mPWD)
    Catch e As System.Exception
        Throw New System.ApplicationException( _
            "DMO: Keine Verbindung möglich.", e)
    End Try
End Sub
```

---

*Listing 429: ConnectSQL-Methode für die Verbindung zum Server*

Nach der Festlegung einer SQL-Server eigenen Authentifizierung (`LoginSecure = False`) wird die Verbindung über die DMO-Prozedur `Connect()` hergestellt. Dieser Prozedur werden der Name des Servers `mServerString`, der SQL-Server-Benutzer `mUser` und das für diesen Benutzer gültige Kennwort `mPWD` übergeben.

Wird diese Methode ohne Fehler ausgeführt, haben Sie eine aktive Verbindung zum Datenbankserver und die »richtige« Arbeit kann durchgeführt werden.

## 240 NT-Security-Anmeldung am SQL-Server

Bei diesem Anmeldeverfahren wird der angemeldete Benutzer des Betriebssystems für die Anmeldung am SQL-Server benutzt.

```
' Server mit NT-Security-Zugriff
Public Sub New(ByVal srv As String)
    mSecurity = True
    Create(srv)
End Sub
```

*Listing 430: NT-Security-Anmeldung*

Näheres zu den Anmeldeverfahren finden Sie im Kapitel »Verwalten der Sicherheit« des SQL-Server-Handbuchs.

Auch an dieser Stelle wird die Verbindung zum Datenbankserver mit einer separaten Connect-Methode durchgeführt. Diese Methode unterscheidet sich in Feinheiten von der entsprechenden Methode in [12.2](#), da es sich hier auch um eine andere Authentifizierung handelt:

```
' Verbindet zum Server mit NT-Berechtigung
Public Sub ConnectNT()
    mServer.LoginSecure = True
    mServer.Name = mServerString
    Try
        mServer.Connect()
    Catch e As System.Exception
        Throw New System.ApplicationException( _
            "DMO: Keine Verbindung möglich.", e)
    End Try
End Sub
```

*Listing 431: ConnectNT -Methode für die Verbindung zum Server*

Dem Serverobjekt `mServer` wird in der ersten Zeile der Methode mitgeteilt, dass eine Anmeldung mit dem Verfahren der NT-Authentifizierung durchgeführt werden soll. Anschließend wird der Name des Servers festgelegt, mit dem das Programm eine Verbindung aufnehmen will. Die Verbindung wird dann mit der DMO-Prozedur `Connect()` hergestellt.

Auch hier gilt: Nach der fehlerfreien Rückkehr aus dieser Methode können Sie mit der eigentlichen Arbeit am SQL-Server beginnen.

## 241 SQL-Server-Anmeldung

Bei diesem Verfahren wird auf die im Server verwalteten Benutzer und Gruppen zurückgegriffen. Diese müssen mit den Benutzern und Gruppen des Betriebssystems nicht identisch sein. Dadurch kann man eine angepasstere Form der Sicherheit für die Datenbankanwendung realisieren. Benötigt werden für dieses Verfahren die gültige Benutzerkennung und das dazugehörige Passwort. Vom Prinzip her ist es mit der Default-Anmeldung (Rezept [12.2](#)) identisch:

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Propert-  
yGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

```
' Server mit SQL-Sicherheits-Zugriff
Public Sub New( _
    ByVal srv As String, ByVal usr As String, _
    ByVal pwd As String)

    mUser = usr
    mPWD = pwd
    mSecurity = False
    create(srv)
End Sub
```

---

*Listing 432: Anmeldung mittels SQL-Server-Authentifizierung*

Dem Konstruktor werden der Name des SQL-Servers, der Benutzer und das entsprechende Kennwort übergeben.

Für die Verbindung zum Datenbank-Server wird wie in [12.2](#) die Methode `ConnectSQL()` benutzt.

Um bei diesem Anmeldeverfahren den Weg bis zur nutzbaren Datenbank zu verkürzen, gibt es noch eine weitere überladene `New`-Prozedur, die direkt eine übergebene Datenbank öffnet:

---

```
' Server mit SQL-Sicherheitszugriff und Datenbank-Auswahl
Public Sub New( _
    ByVal srv As String, ByVal db As String, _
    ByVal usr As String, ByVal pwd As String)

    mUser = usr
    mPWD = pwd
    mSecurity = False
    create(srv)
    ConnectSQL()
    Use(db)
End Sub
```

---

*Listing 433: Anmeldung mit Öffnen einer Datenbank*

Das Öffnen der Datenbank geschieht mittels der Prozedur `use` (siehe Rezept [12.6](#)). Der Name dieser Prozedur wurde aus nostalgischen Gründen gewählt.

## 242 Datenbanken eines Servers

Hat man sich am Server angemeldet, kann man sich mit der folgenden Methode alle von diesem Server verwalteten Datenbanken anzeigen lassen. Zur Ermittlung wird eine Variable vom Typ `SQLDMO.Database` benötigt. Mit dieser Variablen kann man über die Eigenschaft `mServer` iterieren und sich die Namen der Datenbanken holen. Diese werden in einer `ArrayList` abgespeichert und zurückgegeben.

---

```
Public Function Databases() As ArrayList
    Dim db As SQLDMO.Database
    mDBList.Clear()
```

---

*Listing 434: Alle Datenbanken eines Servers ermitteln*



```

For Each db In mServer.Databases
    mDBList.Add(db.Name)
Next
Return mDBList
End Function

```

*Listing 434: Alle Datenbanken eines Servers ermitteln (Forts.)*

## 243 Datenbank festlegen

Zur Auswahl einer Datenbank stellt die Klasse eine Methode `Use()` zur Verfügung, deren Namen sowohl die Absicht dieser Methode her- als auch einen historischen Bezug darstellt. In dieser Methode werden sowohl der Datenbankname als `String` als auch die Variable `mDB` vom Typ `SQLDMO.Database` gesetzt. Dies geschieht durch den Zugriff auf die `Databases`-Auflistung der `SQLDMO.DLL`. In dieser Auflistung kann man über den Namen der Datenbank ein entsprechendes Objekt abrufen, welches den Zugriff auf alle Datenbank-relevanten Informationen erlaubt.

```

Public Sub Use(ByVal db As String)
    If db = String.Empty Then
        Throw New System.ApplicationException( _
            "DMO: Datenbankname wird benötigt")
    End If
    mDBString = db
    mDB = mServer.Databases.Item(db)
End Sub

```

*Listing 435: Auswahl einer Datenbank*

## 244 Tabellen einer Datenbank

Nach Auswahl einer Datenbank (siehe `Use()` Rezept 12.6) ist die aktuelle Datenbank in der Klassenvariablen `mDB` abgespeichert. Wie bei der Ermittlung der Datenbanken (siehe 12.5), kann hier über die Tabellen iteriert werden und die Namen als `ArrayList` an das aufrufende Programm zurückgeliefert werden.

```

Public Function TableList() As ArrayList
    Dim tbl As SQLDMO.Table
    mTableList.Clear()
    For Each tbl In mDB.Tables
        mTableList.Add(tbl.Name)
    Next
    Return mTableList
End Function

```

*Listing 436: Tabellen einer Datenbank*

Die Variable `mTableList` ist innerhalb der Klasse definiert. In Folge dieser Methode wird diese Liste mit Werten gefüllt und anschließend dem aufrufenden Programm übergeben. Dabei bleibt die Liste innerhalb der Klasse naturgemäß mit den ermittelten Daten gefüllt. Auf diese Art muss man die Tabellen nicht jedes Mal neu ermitteln, sondern kann auf die bestehende Liste zurückgreifen.

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

## 245 Felder einer Tabelle

Die Klasse ist so gestaltet, dass Server und Datenbank, aber nicht die aktuelle Tabelle als Variable gespeichert werden. Daher wird der Methode für die Ermittlung der Felder einer Tabelle der benötigte Tabellename übergeben. Damit diese Methode auch sinnvolle Ergebnisse liefert, wird zumindest sichergestellt, dass der Tabellename kein Leerstring oder Empty ist. Für die Iteration über alle Felder der Tabelle wird vorher eine `SQLDMO.Column`-Variable lokal definiert.

---

```
Public Function FieldList(ByVal Table As String) As ArrayList
    Dim fld As SQLDMO.Column

    If Table = String.Empty Then
        Throw New System.ApplicationException( _
            "DMO: Tabelle muss angegeben werden.")
    End If
    mFieldList.Clear()
    mTableString = Table

    For Each fld In mDB.Tables.Item(mTableString).Columns()
        mFieldList.Add(fld.Name)
    Next
    Return mFieldList
End Function
```

---

*Listing 437: Felder einer Tabelle*

Wie bei der Ermittlung der Tabellen einer Datenbank (siehe [12.7](#)) wird eine Variable zur Speicherung der Werte benutzt. Alternativ hätte man hier sicherlich eine methodenlokale `ArrayList` anlegen können, die dann dem aufrufenden Programm übergeben wird. Man hätte so ein paar (k)Bytes an Hauptspeicher sparen können, je nach Umfang der Felderliste.

Ob dies an dieser Stelle also genauso sinnvoll ist wie bei der Tabellenermittlung kann, sicherlich diskutiert werden. Aus Konformitätsgründen wurde dieses Verfahren benutzt (diese Begründung sticht immer ;-)).

## 246 Einfaches Backup einer Datenbank

Um eine einfache Datensicherung der Datenbank durchzuführen, stellen die Database Management Objects eine Funktion zur Verfügung, die, mit einigen Informationen versorgt, ein Backup einer Datenbank in eine Datei durchführt. Für zeitgesteuerte und regelmäßige Datensicherungen stellen die DMOs umfangreichere Funktionen zur Verfügung, die denen des SQL-Explorers durchaus ebenbürtig sind (siehe z. B. [12.12](#)).

---

```
Public Sub BackupDB(ByVal Database As String, _
    ByVal BackupFile As String)

    Dim mBackup As SQLDMO.Backup

    mBackup = New SQLDMO.Backup
    mBackup.Action = SQLDMO.BACKUP_TYPE.SQLDMOBackup_Database
```

---

*Listing 438: Backup einer Datenbank*

```

mBackup.Database = Database
mBackup.DatabaseFiles = Database & "File"
mBackup.Files = BackupFile
mBackup.MediaType = Database & "Media." & DateTime.Now.ToString
mBackup.BackupSetName = Database & "_bs"
mBackup.BackupSetDescription = "Backup of " & Database
mBackup.SQLBackup(mServer)
End Sub

```

*Listing 438: Backup einer Datenbank (Forts.)*

Der Methode werden der Name der zu sichernden Datenbank und der Dateiname der Datensicherung übergeben. Um die Datensicherung gezielt in einem Verzeichnis zu sichern, kann der komplette Pfad übergeben werden, also zum Beispiel *D:\Sicherungen\DBSicherung.bak*.

Nach dem Anlegen der Instanz `mBackup` von `SQLDMO.Backup` wird die Aktion festgelegt, die diese Instanz durchführen soll. Hierfür gibt es vier Möglichkeiten, die in Tabelle 33 Mögliche Backup-Aktionen aufgelistet und im darauf folgenden Text näher erläutert sind.

## 247 Einfaches Zurücksichern einer Datenbank

Leider kommt es vereinzelt dazu, dass die Datensicherung nicht umsonst war, sondern zurückgespeichert werden muss ...

Die entsprechende Methode ist in der Klasse als `RestoreDB()` realisiert. Dieser Methode wird die zu restaurierende Datenbank, die Datensicherungsdatei und der Mediennamen übergeben.

```

Public Sub RestoreDB(ByVal Database As String, _
    ByVal BackupFile As String, ByVal MediaType As String)

    Dim mRestore As SQLDMO.Restore
    mRestore = New SQLDMO.Restore
    mRestore.Action = SQLDMO.RESTORE_TYPE.SQLDMORestore_Database
    mRestore.Database = Database
    mRestore.DatabaseFiles = Database & "File"
    mRestore.Files = BackupFile
    mRestore.MediaType = MediaType
    mRestore.BackupSetName = Database & "_bs"
    mRestore.SQLRestore(mServer)
End Sub

```

*Listing 439: Einfaches Zurücksichern einer Datenbank*

Analog zur Datensicherung wird ein `Restore`-Objekt erstellt und diesem mitgeteilt, was zurückgesichert werden soll. Hier gibt es aber nur drei Möglichkeiten (siehe auch [12.14](#)).

Die Datenbank wird abschließend mit `SQLRestore()` wieder eingespielt. Will man die Datensicherungsintegrität testen, so kann man stattdessen die `SQLVerify()`-Methode aufrufen.

Will man sichergehen, dass die Datenbank zurückgespielt wird, gleichgültig ob es schon eine Datenbank mit diesem Namen gibt oder nicht, kann man dies dadurch erzwingen, indem man vor `SQLRestore()` die Zeile

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

```
mRestore.ReplaceDatabase = True
```

---

einfügt.

## 248 Erstellen eines Backup-Devices

Für mehr Flexibilität und erweiterte Möglichkeiten bietet es sich an, mit so genannten »Backup-Devices« zu arbeiten. Hiermit sind aber keineswegs nur Bandgeräte gemeint. Unter diese Kategorie fallen auch normale Dateien und Netzwerk-Pipes (Named Pipes). Hierbei werden die Sicherungen im MSTF-Format gespeichert, dem Microsoft Tape Format. Bei jeder Sicherung wird ein so genanntes Backup-Set in einer MSFT-Einheit (Unit) erstellt. Diese wird auch als Sicherungsmedium bezeichnet. Je nach Typ des Devices müssen unterschiedliche Angaben an das entsprechende Objekt übergeben werden. Wir beschränken uns hier auf die Sicherung in ein Datei-Device. Die anschließende Sicherung auf einem Magnetband kann dann mit den üblichen Verfahren durchgeführt werden. Diese Vorgehensweise ist in vielen Fällen auch schneller als eine Sicherung auf Magnetband mit diesem Verfahren. Um ein solches Device nutzen zu können, muss dieses erst erstellt werden. Hierfür gibt es die Methode `CreateBackupDevice`:

---

```
Public Sub CreateBackupDevice(ByVal DevName As String, _
    ByVal UNCPATH As String)
    Dim mBackupDev As SQLDMO.BackupDevice

    If DevName = String.Empty Then
        Throw New System.ApplicationException( _
            "DMO: Devicename ist leer")
    End If

    If UNCPATH = String.Empty Then
        Throw New System.ApplicationException( _
            "DMO: UNC-Pfad ist leer")
    End If

    If mServer.Status <> SQLDMO_SVCSTATUS_TYPE.SQLDMOSvc_Running Then
        Throw New System.ApplicationException("DMO: " & _
            "Nicht verbunden zum Server oder Server nicht aktiv.")
    End If

    Try
        mBackupDev = New BackupDevice
        mBackupDev.Name = DevName
        mBackupDev.PhysicalLocation = UNCPATH
        mBackupDev.Type = SQLDMO_DEVICE_TYPE.SQLDMODevice_DiskDump
        mServer.BackupDevices.Add(mBackupDev)
    Catch e As Exception
        Throw New System.ApplicationException( _
            "DMO: CreateBackupDevice", e)
    End Try
End Sub
```

---

Der Methode werden der Device-Name und der Datei-Pfad übergeben. Der Name des Devices ist im Falle der Festplattensicherung frei wählbar. Ebenso die komplette Pfadangabe der zu erstellenden Datei.

Nach der Deklaration der Variablen `mBackupDev` vom Typ `SQLDMO.BackupDevice` und dem groben Test auf die Verwendbarkeit der übergebenen Werte wird mittels der vorher erstellten Variablen `mServer` getestet, ob der Dienst noch läuft (zwischen Definition der Variablen `mServer` und dem Aufruf dieser Methode könnte jemand den Dienst beendet haben...). Ist dies der Fall, wird die Variable `mBackupDev` definiert. Dieser wird anschließend die Bezeichnung des Backup-Devices übergeben. Im Falle der Dateisicherung ist die Eigenschaft `PhysicalLocation` festzulegen, was mit der übergebenen Pfadangabe geschieht.

Die Art des Sicherungsmediums wird mit der Eigenschaft `Type` festgelegt. Diese hat zurzeit die folgenden Ausprägungen:

| Typ                      | Wert | Beschreibung                                  |
|--------------------------|------|-----------------------------------------------|
| SQLDMODevice_DiskDump    | 2    | Festplattensicherung                          |
| SQLDMODevice_FloppyADump | 3    | Sicherung auf Laufwerk A:                     |
| SQLDMODevice_FloppyBDump | 4    | Sicherung auf Laufwerk B:                     |
| SQLDMODevice_TapeDump    | 5    | Sicherung auf Magnetband                      |
| SQLDMODevice_PipeDump    | 6    | Sicherung über Named Pipes                    |
| SQLDMODevice_CDROM       | 7    | Nicht benutzt (für die Zukunft)               |
| SQLDMODevice_Unknown     | 100  | Fehler: Gerät ungültig oder nicht ansprechbar |

Tabelle 32: Typus von Sicherungsmedien

Schlussendlich wird das so erstellte Device der Liste der dem Server bekannten Devices hinzugefügt.

Nach dem Anlegen eines solchen Sicherungsmediums wird die Datei noch nicht physikalisch auf der Festplatte erstellt. Dies geschieht erst bei der ersten Sicherung. Im Enterprise Manager des SQL-Servers kann man allerdings das Device finden:

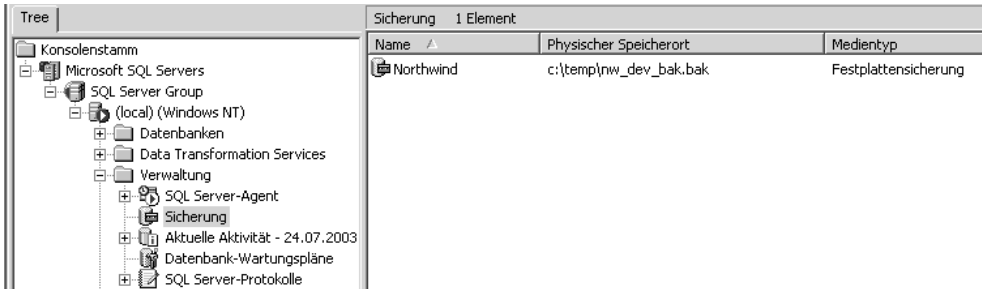


Abbildung 220: Backup-Device im Enterprise Manager

Wie man in Abbildung 220 erkennt, wurde als Name die Bezeichnung der in diesem Medium zu sichernden Datenbank gewählt. In der Spalte »Physischer Speicherort« wird der Name der Datei angezeigt.

## 249 Datensicherung auf ein Backup-Device

Hat man auf die oben geschilderte Weise ein Backup-Device erstellt oder ist dies manuell mittels des Enterprise Managers des SQL-Servers geschehen, kann man eine Datensicherung auf dieses Device durchführen.

Die DMO.NET-Klasse bietet hierfür mit der Methode `DeviceBackup()` eine entsprechende programmtechnische Umsetzung. Diese Methode nutzt nicht alle Möglichkeiten, die dieses Verfahren bietet, eine Erweiterung ist aber ohne Schwierigkeiten möglich, wenn man die Vorgehensweise dieser Methode verstanden hat.

---

```
Public Sub DeviceBackup(Optional ByVal log As Boolean = False)
    Dim mBackup As SQLDMO.Backup = New SQLDMO.Backup
    Dim devs As SQLDMO.BackupDevices
    Dim dev As SQLDMO.BackupDevice
```

---

Die Methode wird mit einem optionalen Parameter aufgerufen, der darüber entscheidet, ob die Transaktionslogs mitgesichert werden sollen oder nicht.

Um die eigentliche Backup-Methode aufzurufen, wird ein Objekt vom Typ `SQLDMO.Backup` erstellt. Diesem Objekt werden die Anforderungen der Datensicherung im weiteren Verlauf der Methode übergeben. Hier bieten sich auch entsprechende Erweiterungen der Methode an.

Zusätzlich werden zwei Variable benötigt: eine die die Auflistung der Devices übernimmt, die andere um ein einzelnes Device ansprechen zu können.

---

```
If mDBString = String.Empty Then
    Throw New System.ApplicationException( _
        "DMO: Datenbank erforderlich.")
End If
```

---

Zuerst wird überprüft, ob die Eigenschaft `mDBString` der Klasse nicht leer ist. Dies sollte hier nicht möglich sein, aber bei dieser Methode sollte man Vorsicht walten lassen. Es ist auch denkbar, den Test dieser Eigenschaft zu erweitern. Ein nichtleerer String stellt erst mal noch keine Datenbank dar. Analog zu Listing 12.16 kann man hier auch noch einen Test auf den Status des Servers einfügen. Welche Sicherheit man hier einbaut, hängt im Wesentlichen auch davon ab, ob man diesen Server auf einem Einzelplatz oder im Netz betreibt.

---

```
Try
    mBackup.Action = SQLDMO_BACKUP_TYPE.SQLDMOBackup_Database
    mBackup.Database = mDBString
```

---

Die eigentliche Aktion wird in einem Try-Catch-Block eingeklammert. Die durchzuführende Aktion wird mit `SQLDMOBackup_Database` festgelegt. Die möglichen Aktionen

| Typ                                    | Wert | Beschreibung                                                                         |
|----------------------------------------|------|--------------------------------------------------------------------------------------|
| <code>SQLDMOBackup_Database</code>     | 0    | Sicherung einer kompletten Datenbank                                                 |
| <code>SQLDMOBackup_Differential</code> | 1    | Inkrementelle Datensicherung; Sicherung der Daten nach dem letzten kompletten Backup |

Tabelle 33: Mögliche Backup-Aktionen

| Typ                | Wert | Beschreibung                                               |
|--------------------|------|------------------------------------------------------------|
| SQLDMOBackup_Files | 2    | Sicherung einzelner physikalischer Dateien einer Datenbank |
| SQLDMOBackup_Log   | 3    | Sicherung der Transaktionslogs der Datenbank               |

Tabelle 33: Mögliche Backup-Aktionen (Forts.)

Bei SQLDMOBackup\_Database wird die gesamte Datenbank gesichert. Existieren für die Datenbank mehrere Dateien, so kann man mittels SQLDMOBackup\_Files einzelne Dateien der Datenbank sichern (Datendateien, Indexdateien). Dies empfiehlt sich bei großen Datenbanken. Dabei sollte allerdings auf die Konsistenz der Datensicherung geachtet werden. Um die Dauer der Datensicherung zu minimieren kann man auch eine Differenzsicherung zur letzten Gesamtsicherung durchführen. Dies geschieht mittels SQLDMOBackup\_Incremental. Hierbei muss man einen Kompromiss zwischen Gesamt- und Differenzsicherung finden, da bei einer Rücksicherung alle Differenzsicherungen auch wieder eingespielt werden müssen. Arbeitet die Datenbank mit Transaktions-Logdateien, so können diese mit SQLDMOBackup\_Log gesichert werden.

```
devs = mServer.BackupDevices()  
dev = devs.Item(1)
```

Anschließend wird die Liste der Devices ermittelt. Um die Methode übersichtlich zu halten, wird hier automatisch als Sicherung das erste gefundene Device für das Backup genommen. Über die Reihenfolge der Device-Erstellung kann man darauf Einfluss nehmen, wohin dieses Backup geht (siehe Tabelle 33). Alternativ kann man sich im Vorfeld eine Liste der Devices holen (siehe 12.13) und diese Methode um den entsprechenden Übergabeparameter erweitern. Wie man hier auch erkennt, ist die Auflistung 1-basiert.

```
mBackup.Devices = "[" & dev.Name & "]"  
mBackup.BackupSetDescription = "Komplett-Backup von " & _  
    mDBString & " vom " & DateTime.Now.ToString  
mBackup.BackupSetName = mDBString & "-Komplett"  
mBackup.SQLBackup(mServer)
```

Dem Backup-Objekt wird der Name des Devices in eckigen Klammern übergeben. Die Beschreibung des Backups ist frei wählbar, hier werden der Datenbankname und der aktuelle Zeitpunkt als Beschreibung gewählt. Nachdem man dem Backup auch noch einen Namen gegeben hat, wird das Backup der Datenbank für das Serverobjekt durchgeführt.

```
If log Then ' bei Backup = Simple geht Log nicht  
    mBackup.Action = SQLDMO_BACKUP_TYPE.SQLDMOBackup_Log  
    mBackup.BackupSetDescription = "Backup TranactionLog von " &  
        & mDBString & " vom " & DateTime.Now.ToString  
    mBackup.BackupSetName = mDBString & "-Transaktionslog"  
    mBackup.SQLBackup(mServer)  
End If  
Catch e As Exception  
    Throw New System.ApplicationException("DMO: Backup", e)  
End Try  
End Sub
```

Wurde der Methode beim Aufruf mitgeteilt, dass die Transaktionslogs ebenfalls gesichert werden sollen (`log = True`), werden die entsprechenden Aktionseinstellungen neu gesetzt und die Transaktionslogs ebenfalls auf diesem Device gesichert.

Nach erfolgreicher Sicherung findet sich im entsprechenden Verzeichnis der Festplatte eine entsprechende Datei. Mit den Einstellungen, wie sie in Abbildung 219 angegeben sind, findet man im Verzeichnis `C:\Temp` die Datei `nw_dev_bak.bak` für die Device-Sicherung.

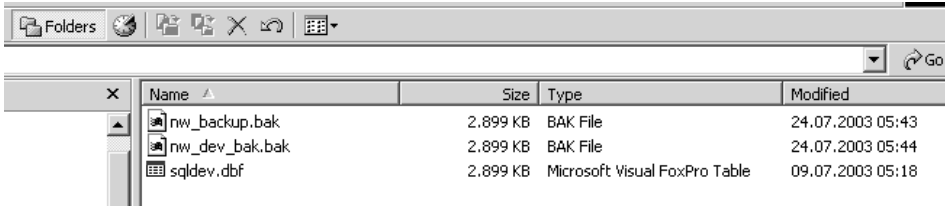


Abbildung 221: Festplattenverzeichnis nach Datensicherung

Man sieht hier auch, dass die Dateigröße bei der Default-Device-Sicherung mit der der einfachen Datensicherung (siehe [12.9](#)) übereinstimmt.

## 250 Liste der Backup-Devices

Wie in [12.12](#) erwähnt, kann man sich eine Liste der Backup-Devices holen, um anschließend entscheiden zu können, auf welches Device die Datensicherung gehen soll. Diese Liste wird mit der folgenden Methode als `ArrayList` ermittelt.

---

```
Public Function GetBackupDevices() As ArrayList
    Dim devs As SQLDMO.BackupDevices
    Dim dev As SQLDMO.BackupDevice
```

---

Wie in [12.12](#) werden zwei Variable für die Liste der Devices und dem Zugriff auf ein einzelnes Device benötigt und zu Beginn der Methode deklariert.

---

```
    If mBackupDevList.Count > 0 Then
        Return mBackupDevList
    End If
```

---

Sollte diese Methode schon einmal aufgerufen worden sein, ist die Liste der DMO.NET-Klasse bereits gefüllt und muss nicht noch einmal gefüllt werden. Daher wird zu Beginn hierauf geprüft und gegebenenfalls direkt zurückgesprungen. Dies ist auch eindeutig, da sich eine Instanz der Klasse nur mit einem SQL-Server verbindet und dieser auch nur eine Device-Liste führt. Sollten die Umstände allerdings so sein, dass sich während der Existenz des DMO.NET-Objektes die Device-Liste des Servers ändern kann, sollte dieser Check deaktiviert werden.

---

```
    devs = mServer.BackupDevices()
    For Each dev In devs
        mBackupDevList.Add(dev.Name)
    Next
    Return mBackupDevList
End Function
```

---



Sind in der Variablen `devs` die einzelnen Device-Objekte hinterlegt, kann die `ArrayList` in einer Schleife gefüllt und anschließend zurückgegeben werden.

## 251 Rücksicherung von einem Backup-Device

Um eine Datensicherung auf ein Device wieder einzuspielen, ist in der `DMO.NET`-Klasse die Methode `DeviceRestore()` realisiert. Die Logik wurde zur Backup-Methode etwas abgewandelt:

```
Public Sub DeviceRestore(ByVal DeviceName As String, _
    ByVal BackupNumber As Integer, ByVal LOG As Boolean, _
    ByVal Last As Boolean)
```

Der Device-Name entspricht dem der Backup-Methode. Das Kennzeichen für eine Rücksicherung der Transaktions-Logs ( `LOG` ) hat keinen Standardwert und ist nicht optional. Zusätzlich wird der Methode die Nummer des Backups und ein Parameter `Last` übergeben. Zu Letzterem finden sich einige Anmerkungen weiter unten im Text. Damit ist diese Methode auch in der Lage, Sicherungen zurückzuspeichern, die mittels des Enterprise-Managers oder mit einer erweiterten Fassung der Backup-Methode durchgeführt wurden.

```
Dim mRestore As SQLDMO.Restore = New SQLDMO.Restore
```

Für die Rücksicherung wird analog zum Backup ein Objekt vom Typ `SQLDMO.Restore` angelegt

```
If DeviceName = String.Empty Then
    Throw New System.ApplicationException( _
        "DMO: Device_name für Restore erforderlich.")
End If
```

Die Überprüfung, ob ein Device-Name übergeben wurde, ist an dieser Stelle sehr einfach gehalten. Es wird zum Beispiel nicht geprüft, ob in diesem String bereits »[« und »]« enthalten sind.

```
If mDBString = String.Empty Then
    Throw New System.ApplicationException( _
        "DMO: Datenbank erforderlich.")
End If
```

Es sollte schon feststehen, welche Datenbank zurückgesichert werden soll ...

```
Try
    If LOG Then
        mRestore.Action = SQLDMO_RESTORE_TYPE.SQLDMORestore_Log
    Else
        mRestore.Action = SQLDMO_RESTORE_TYPE.SQLDMORestore_Database
    End If
```

Hier unterscheidet sich die Logik von der Backup-Methode. Mit dem Übergabeparameter `LOG` wird nicht entschieden, ob nach Rücksicherung der reinen Datenbank auch die Transaktions-Logs wieder hergestellt werden sollen, sondern hier ist eine »entweder – oder«-Logik realisiert. Für die Restore-Aktionen sind die folgenden Werte möglich:

| Typ                    | Wert | Beschreibung                               |
|------------------------|------|--------------------------------------------|
| SQLDMORestore_Database | 0    | Rücksicherung der Datenbank                |
| SQLDMORestore_Files    | 1    | Rücksicherung einzelner Sicherungsdateien  |
| SQLDMORestore_Log      | 2    | Rücksicherung der Transaktions-Log-Dateien |

Tabelle 34: Mögliche Restore-Werte

Damit ist geklärt, was zurückgesichert werden soll, und die weiteren Eigenschaften für das Objekt können gesetzt werden:

```
mRestore.Database = mDBString
mRestore.Devices = "[" & DeviceName & "]"
mRestore.FileNumber = BackupNumber
```

Auf einem Device können mehrere so genannte Backup-Sets gespeichert werden. Diese werden mit einer laufenden Nummer versehen. In der Backup-Methode haben wir diese Eigenschaft implizit gesetzt, da diese Zahl für jede Sicherung auf einem Device automatisch gesetzt wird.

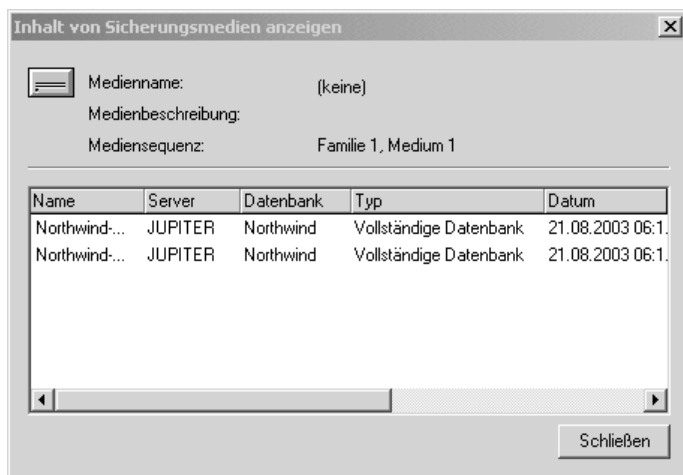


Abbildung 222: Zwei Sicherungen auf einem Device

Führt man mit der oben beschriebenen `DeviceBackup`-Methode zwei Sicherungen durch, so kann man sich im Enterprise-Manager ansehen, dass die Sicherung nicht überschrieben, sondern als eigenständige Sicherung auf dem Device gespeichert wurde (siehe Abbildung 222)

```
mRestore.ReplaceDatabase = True
mRestore.LastRestore = Last
mRestore.SQLRestore(mServer)
Catch e As System.Exception
    Throw New System.ApplicationException("DMO: Restore", e)
End Try
End Sub
```

Die Erläuterungen zum Thema Devices und Medien würden den Rahmen an dieser Stelle sprengen. Einen Einstieg finden Sie im Kapitel »Verwenden von Mediensätzen und Medienfamilien« im Handbuch des SQL-Servers.

## 252 Erstellen einer Datenbank

Will man innerhalb eines Programmes eine neue Datenbank erstellen, so kann man dies sicherlich mit der T-SQL-Anweisung *CREATE DATABASE* durchführen. Es geht aber auch mit der hier gezeigten Methode `CreateDatabase()`.

```
Public Sub CreateDatabase(ByVal Name As String, _
    ByVal Path As String, ByVal InitialSize As Integer)
```

Der Methode werden drei Parameter übergeben. Der Name der Datenbank ist eine Selbstverständlichkeit. Der Parameter `Path` enthält den Dateipfad der physikalischen Datei für diese Datenbank. Dieser ist standardmäßig `C:\Programme\Microsoft SQL Server\MSSQL\Data`. Hat man mehrere Datenbanken im Produktivbetrieb, empfiehlt sich hier, pro Datenbank eine physikalische Platte vorzuhalten. Das kann die Geschwindigkeit der Zugriffe beträchtlich erhöhen, da sich die Datenbanken bei den Schreib-/Lese-Aktionen nicht mehr gegenseitig stören können. In dieser Methode wird der Dateiname der physikalischen Datei aus dem übergebenen Namen der Datenbank und der Endung `.mdf` generiert. Will man den Namen frei vergeben, so muss diese Methode um den entsprechenden Dateinamens-Parameter erweitert werden oder die Logik für den Parameter `Path` wird geändert. Die Datenbank wird mit Transaktions-Logging erstellt. Die Datei hierfür wird analog gebildet, hat aber die Dateierweiterung `.ldf`. In Einzelfällen kann man auch darüber nachdenken, diese physikalische Datei auf eine gesonderte Festplatte zu legen.

Der Parameter `InitialSize` stellt die Größe der Datenbankdatei in Megabyte dar, mit der sie zum Zeitpunkt der Datenbankerstellung kreiert wird.

```
Dim mPath As String
Dim mDB As SQLDMO.Database = New SQLDMO.Database
Dim mDBFile As SQLDMO.DBFile = New SQLDMO.DBFile
Dim mDBLog As SQLDMO.LogFile = New SQLDMO.LogFile
```

Zu Beginn werden die benötigten SQLDMO-Objekte definiert.

```
If Name = String.Empty Then
    Throw New System.ApplicationException( _
        "DMO: Datenbankname wird benötigt.")
End If
```

Wie bei anderen Methoden schon erwähnt, findet eine einfache Prüfung auf den Inhalt des Parameters `Name` statt.

```
If InitialSize <= 0 Then
    Throw New System.ApplicationException( _
        "DMO: Erstellungsgröße der DB sollte > 0 sein.")
End If
```

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tung  
Win-  
dows  
Forms

Controls

Proper-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

Die Größe der Datenbankdatei ist frei wählbar und sollte den Startbedingungen der Datenbank angepasst werden, muss aber auf jeden Fall größer als Null Megabyte sein.

---

```
If mServer.Status <> SQLDMO_SVCSTATUS_TYPE.SQLDMOSvc_Running Then
    Throw New System.ApplicationException( _
        "DMO: Server muss aktiv sein")
End If
```

---

Damit eine Datenbank erstellt werden kann, muss naturgemäß der Server aktiv sein. Dies sollte an dieser Stelle zwar der Fall sein, aber man weiß ja nie, wer oder was gerade den Server heruntergefahren hat ;-).

---

```
Try
    mPath = IO.Path.GetDirectoryName(Path)
Catch e As Exception
    Throw New System.ApplicationException( _
        "DMO: Pfadangaben falsch", e)
End Try
```

---

Die Existenz des Dateipfades wird in dieser Methode vorausgesetzt. Es ist eine Geschmacksfrage, ob man hier die Erstellung eines Pfades erlaubt.

---

```
Try
    mDBFile.PrimaryFile = True
    mDBFile.Name = Name
    mDBFile.PhysicalName = Path & "\" & Name & ".mdf"
    mDBFile.Size = InitialSize
    mDBFile.FileGrowthType = SQLDMO_GROWTH_TYPE.SQLDMOGrowth_Percent
    mDBFile.FileGrowth = 5
    mDB.Name = Name
    mDB.FileGroups.Item("PRIMARY").DBFiles.Add(mDBFile)
```

---

Jede Datenbank kann auf mehrere physikalische Dateien verteilt werden, die über eine Dateigruppe (FileGroup) verwaltet werden. Zusätzlich werden die Dateigruppen über eine Auflistung FileGroups verwaltet. An dieser Stelle wird nur eine Datei erstellt, die über das Objekt mDBFile mit entsprechenden Eigenschaften versorgt wird. Auch hier sei der Hinweis auf große Datenbanken gestattet. Man kann so für die Indizes der Tabellen eine eigene Datei erstellen, die dann eventuell auf einer anderen Festplatte liegt. Dies kann man bei der Erstellung eines Index in der T-SQL-Anweisung angeben: *CREATE ... INDEX ... ON filegroup ...*

Analog kann man auch für eine Tabelle mit *CREATE ... TABLE ... ON filegroup ...* den physikalischen Ort der Tabellenspeicherung festlegen. Auf diese Art und Weise ist eine feine Granulierung der Datenbank möglich. Einstiegspunkt in diese Thematik ist das Kapitel »Physische Datenbankdateien und Dateigruppen« des SQL-Server-Handbuchs, dort finden Sie auch weitere Informationen.

Da eine Datenbank typischerweise immer mehr Daten verwaltet, wird die bei der Erstellung der Datenbank angegebene Größe über kurz oder lang erreicht. Hier kommt die Eigenschaft FileGrowthType des SQLDMO-Objektes zum Tragen.

| Name                 | Wert | Beschreibung                                                                   |
|----------------------|------|--------------------------------------------------------------------------------|
| SQLDMOGrowth_MB      | 0    | Die Datenbank wird um den in FileGrowth angegebenen Wert Megabyte vergrößert   |
| SQLDMOGrowth_Percent | 1    | Die Datenbank wird um den in FileGrowth angegebenen Wert in Prozent vergrößert |

Tabelle 35: FileGrowthType-Werte für das Anwachsen der Datenbank

Die hier vorgestellte Methode vergrößert die Datenbankdatei jeweils um 5 Prozent, da der Vergrößerungsfaktor FileGrowth mit 5 belegt wird. Sollten diese 5 Prozent aber kleiner als 1 Megabyte sein, so wird die Datenbank um 1 Megabyte vergrößert. Dies ist die kleinste Vergrößerungseinheit des SQL-Servers. Wer gerne experimentiert, kann den Fall austesten, wenn die Vergrößerungsart auf SQLDMO\_Groth\_MB gesetzt ist und der Vergrößerungsfaktor auf 0 gesetzt wird.

Anschließend wird der Datenbank der Name gegeben und der primären FileGroup zugeordnet.

```
mDBLog.Name = Name & "Log"
mDBLog.PhysicalName = Path & "\" & Name & ".ldf"
mDBLog.Size = Math.Max(CInt(InitialSize / 10), 1)
mDB.TransactionLog.LogFiles.Add(mDBLog)
mServer.Databases.Add(mDB)
Catch e As Exception
    Throw New System.ApplicationException( _
        "DMO: " & e.Message & e.Source & e.StackTrace, e)
End Try
End Sub
```

Abschließend werden die Transaktions-Log-Eigenschaften festgelegt. Der Name wird aus dem Datenbanknamen und dem Anhang »Log« gebildet. Die Datei selber hat den Namen der Datenbank und die Endung ».ldf« (s.o.). Die anfängliche Größe wird auf 10 Prozent der Datenbankgröße festgelegt, aber mindestens auf 1 Megabyte. Dies geschieht mit dem Math.Max-Operator. Nachdem die Log-Dateien der neuen Datenbank zugeordnet wurden, wird diese Datenbank dem Server zugeordnet.

Wie man den einzelnen Anmerkungen in diesem Kapitel entnehmen kann, wird hier nur eine recht einfache Form der Datenbankerstellung realisiert. Es gibt vielfältige Erweiterungsmöglichkeiten, die aber entscheidend von den individuellen Anforderungen abhängen. Die hier vorgestellte Methode stellt aber einen guten Ausgangspunkt für eigene Erweiterungen dar.

## 253 Erstellen eines T-SQL-Datenbank-Skriptes

Die Möglichkeiten der T-SQL-Skripte sind sehr umfangreich. Sie können an dieser Stelle nicht erschöpfend dargestellt werden. Über dieses Thema gibt es eigene Literatur. Hier sollen nur die Möglichkeiten der SQLDMOs in dieser Richtung dargestellt werden. Eine Möglichkeit ist die Generierung von Skripten für die Struktur einer Datenbank. In Listing 441 ist ein kleiner Auszug aus dem generierten Skript dargestellt, welches mit der Methode CreateScript() für die Datenbank Northwind erstellt wurde.

---

```

/* Objekt: Tabelle [dbo].[Orders] Skriptdatum:
22.08.2003 05:41:33 **/
CREATE TABLE [Orders] (
    [OrderID] [int] IDENTITY (1, 1) NOT NULL ,
    [CustomerID] [nchar] (5) COLLATE
SQL_Latin1_General_CP1_CI_AS NULL ,
    [EmployeeID] [int] NULL ,
    [OrderDate] [datetime] NULL ,
    [RequiredDate] [datetime] NULL ,
    [ShippedDate] [datetime] NULL ,
    [ShipVia] [int] NULL ,
    [Freight] [money] NULL CONSTRAINT [DF_Orders_Freight]
        DEFAULT (0),
    [ShipName] [nvarchar] (40) COLLATE
SQL_Latin1_General_CP1_CI_AS NULL ,
    [ShipAddress] [nvarchar] (60) COLLATE
SQL_Latin1_General_CP1_CI_AS NULL ,
    [ShipCity] [nvarchar] (15) COLLATE
SQL_Latin1_General_CP1_CI_AS NULL ,
    [ShipRegion] [nvarchar] (15)
COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
    [ShipPostalCode] [nvarchar] (10) COLLATE
SQL_Latin1_General_CP1_CI_AS NULL ,
    [ShipCountry] [nvarchar] (15) COLLATE
SQL_Latin1_General_CP1_CI_AS NULL ,
    CONSTRAINT [PK_Orders] PRIMARY KEY CLUSTERED
    (
        [OrderID]
    ) ON [PRIMARY] ,
    CONSTRAINT [FK_Orders_Customers] FOREIGN KEY
    (
        [CustomerID]
    ) REFERENCES [Customers] (
        [CustomerID]
    ),
    CONSTRAINT [FK_Orders_Employees] FOREIGN KEY
    (
        [EmployeeID]
    ) REFERENCES [Employees] (
        [EmployeeID]
    ),
    CONSTRAINT [FK_Orders_Shippers] FOREIGN KEY
    (
        [ShipVia]
    ) REFERENCES [Shippers] (
        [ShipperID]
    )
) ON [PRIMARY]
GO

```

---

In Listing 441 ist die CREATE TABLE-Anweisung für die Tabelle Orders dargestellt. Man erkennt die Erstellung aller Spalten, aller Einschränkungen (Constraints) und die FileGroup, auf der die Tabelle erstellt werden soll.

Das durch diese Methode erstellte Skript wird als String an das aufrufende Programm zurückgegeben. Abgespeichert kann es in den SQL-Server-Query-Analyzer eingelesen werden. Auf diese Art und Weise kann man schnell eine leere Datenbank erstellen, um zum Beispiel die Produktivdatenbank nicht für Testzwecke zu missbrauchen oder um dem Kunden eine saubere Datenbank zu liefern. Was, nebenbei bemerkt, keine Selbstverständlichkeit ist.

```
Public Function CreateScript() As String
    Dim mParam As SQLDMO.SQLDMO_SCRIPT_TYPE = _
        SQLDMO_SCRIPT_TYPE.SQLDMOScript_Default Or _
        SQLDMO_SCRIPT_TYPE.SQLDMOScript_DRI_All Or _
        SQLDMO_SCRIPT_TYPE.SQLDMOScript_IncludeHeaders
    Dim mScript As String
    Dim tbl As SQLDMO.Table
```

Zur Steuerung des Umfanges der Skript-Datei wird ein Parameter vom Typ SQLDMO\_SCRIPT\_TYPE benötigt, dessen Inhalt über eine Oder-verknüpfte Liste von Eigenschaften erzeugt wird. Eine komplette Liste aller möglichen Werte erhalten Sie, wenn Sie im Index des elektronischen Handbuchs SQLDMO\_SCRIPT\_TYPE nachschlagen. Die dort gezeigte Tabelle enthält 39 Einträge, von denen in Tabelle 36 nur ein Auszug der am häufigsten gebrauchten Werte enthält.

| Name                         | Beschreibung                                                                                                                                                                  |
|------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| SQLDMOScript_IncludeHeaders  | Das erzeugte Skript enthält Informationen u.a. über Erstellungsdatum und Zeit                                                                                                 |
| SQLDMOScript_Default         | Muss angegeben werden, sonst wird das Skript nicht erstellt                                                                                                                   |
| SQLDMOScript_DRI_All         | Sammelbezeichnung für alle Werte, die mit SQLDMOScript_DRI_ beginnen                                                                                                          |
| SQLDMOScript_DRI_ForeignKeys | Skript enthält die Constraints für die Fremdschlüssel der betrachteten Tabelle                                                                                                |
| SQLDMOScript_Triggers        | Skript enthält Anweisungen zur Definition der Trigger der jeweiligen Tabelle oder View                                                                                        |
| SQLDMOScript_ToFileOnly      | Ein Großteil der SQLDMO-Skript-Objekte liefert sowohl einen Rückgabewert als auch eine optionale Datei zurück. Hiermit wird eingestellt, dass nur die Datei beschrieben wird. |
| SQLDMOScript_Indexes         | Eine Kombination verschiedener Werte, die zusammen dafür sorgen, dass das Skript alle Informationen bzgl. der Indizes von Tabelle/View enthält                                |
| SQLDMOScript_AppendToFile    | Normalerweise wird bei der Angabe einer Datei diese überschrieben. Mittels dieses Wertes wird an eine vorhandene Datei angehängt                                              |

Tabelle 36: Auswahl von SQLDMO\_SCRIPT\_TYPE-Werten

|                           |
|---------------------------|
| Basics                    |
| Datum/<br>Zeit            |
| Anwen-<br>dungen          |
| Zeich-<br>nen             |
| Bild-<br>bearbei-<br>tung |
| Win-<br>dows<br>Forms     |
| Controls                  |
| Proper-<br>tyGrid         |
| Datei-<br>system          |
| Netz-<br>werk             |
| System                    |
| Daten-<br>banken          |
| XML                       |
| Wissen-<br>schaft         |
| Ver-<br>schiede-<br>nes   |

Nach der Deklaration der Variablen für das Skript und einer Laufvariablen für die Schleife über alle Tabellen wird wieder getestet, ob der Server noch aktiv ist:

---

```
If mServer.Status <> SQLDMO_SVCSTATUS_TYPE.SQLDMOSvc_Running Then
    Throw New System.ApplicationException( _
        "DMO: Server mu aktiv sein")
End If
```

---

In der Schleife wird für jede Tabelle der aktuell gewählten Datenbank das Skript generiert und an das Ende der Zeichenkette angehängt.

---

```
For Each tbl In mDB.Tables
    mScript &= tbl.Script(mParam)
Next
Return mScript
End Function
```

---

Abschließend wird die Zeichenkette zurückgegeben. Will man das Ergebnis gleichzeitig in eine Datei kopieren, so muss die Variable `mParam` um den Wert `SQLDMOScript_AppendToFile` erweitert und die Zeile innerhalb der Schleife durch

---

```
mScript &= tbl.Script(mParam, "c:\temp\skript.sql")
```

---

ersetzt werden.

## 254 Erstellen eines Jobauftrages

Will man die Datenbank in regelmäßigen Abständen warten, so empfiehlt sich, dies automatisch vom Computer durchführen zu lassen. Zu diesem Zweck kann der SQL-Server unterschiedliche Aufgaben automatisch ausführen. Die Planung und durchzuführenden Tätigkeiten kann man sowohl mittels des SQL-Server-Explorers als auch per Programm festlegen. Hat man die MSDE im Einsatz, bleibt nur die eigenprogrammierte Lösung. Die Tätigkeiten werden in einem so genannten Job gesammelt. Dieser Job kann dann zu einem beliebigen Zeitpunkt durchgeführt und wiederholt werden. Die Bezeichnung Job kommt im Übrigen aus der »Frühzeit« der EDV, als noch alle Computerprogramme per Batch-Jobs erledigt wurden.

Als Beispiel wird hier eine Methode vorgestellt, die einen Datensicherungsjob erstellt.

---

```
Public Sub CreateJob(ByVal JobName As String, ByVal StartPoint As _
    DateTime, Optional ByVal BackupFile As String = _
    "c:\temp\backup.bak")
```

---

Jeder Job muss einen Namen haben, der dieser Methode mit dem Parameter `JobName` übergeben wird. Der Zeitpunkt der Datensicherung wird als `StartPoint` übergeben. Wird der Methode kein Dateiname für die Sicherung übergeben, wird per Default-Wert die Datei *backup.bak* im Verzeichnis *C:\TEMP* erzeugt.



```

Dim mJob As SQLDMO.Job = New SQLDMO.Job
Dim mSchedule As SQLDMO.JobSchedule = New SQLDMO.JobSchedule
Dim mStep As SQLDMO.JobStep = New SQLDMO.JobStep
Dim mDate As String
Dim mTime As String

```

Zur Erzeugung eines entsprechenden Jobs werden Variable vom Typ `SQLDMO.Job` und `SQLDMO.JobSchedule` definiert. Da Jobs sehr umfangreich werden können, können Sie die Abarbeitung solcher Jobs in mehrere Schritte aufteilen. Zum Beispiel können im ersten Schritt die Datenbank, im zweiten die Transaktionslogs gesichert werden, um dann in einem dritten Schritt ein externes Programm aufzurufen. Für diese Zwecke muss dem Server mitgeteilt werden, welcher Schritt welche Aktionen durchführt. Zur Steuerung dieser Information wird die Variable `mStep` definiert.

```

If mServer.Status <> SQLDMO_SVCSTATUS_TYPE.SQLDMOSvc_Running Then
    Throw New System.ApplicationException( _
        "DMO: Server muss aktiv sein")
End If

mTime = StartPoint.ToString("HHmmss")
mDate = StartPoint.ToString("yyyyMMdd")
mJob.Name = JobName
mJob.Description = JobName & "-Scheduler"
mJob.Category = "Datenbankwartung"
mSchedule.Name = JobName

```

Nach dem Test auf den Status des Servers wird der Übergabeparameter `StartPoint` in Zeit- und Datumstrings aufgesplittet, da diese Informationen später in diesem Format benötigt werden. Jeder Job läuft im Kontext einer Kategorie ab. Bei Installation des Servers werden einige Kategorien automatisch erzeugt. Zu diesen zählt auch die Kategorie »Datenbankwartung«. Anzeigen und/oder erzeugen kann man solche Kategorien im Enterprise-Manager im Kontextmenü ALLE TASKS unter VERWALTUNG -> AUFTRÄGE. Die Verwaltung von Kategorien ist auch programmtechnisch mit den Mitteln der `SQLDMO` machbar.

```

mServer.JobServer.BeginAlter()
mServer.JobServer.Jobs.Add(mJob)
mServer.JobServer.DoAlter()

```

Werden Änderungen an `SQLDMOs` durchgeführt, führt dies zu einer sofortigen Änderung des entsprechenden Objektes. Will man mehrere Objekte ändern, kann dies zu unerwünschten Nebeneffekten führen. Um solche »Fehler« zu vermeiden, werden diese Änderungen in einen Block `BeginAlter ... DoAlter` eingeschlossen. An dieser Stelle wäre diese Klammerung nicht unbedingt notwendig, aber fügt man zu einem späteren Zeitpunkt etwas ein, vergisst man schnell solche Kleinigkeiten. Will man eine Änderung abbrechen, kann man dies mit `CancelAlter` erreichen.

```

mStep = New SQLDMO.JobStep
mStep.Name = "1st Step"
mStep.StepID = 1
mStep.DatabaseName = mDBString
mStep.SubSystem = "TSQL"

```

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Proper-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

---

```

mStep.Command = "backup database [" & mDBString & _
    "]" To disk = "'" & _
    BackupFile & "' With retaindays = 30"
mStep.OnSuccessAction = _
    SQLDMO_JOBSTEPACTION_TYPE.SQLDMOJobStepAction_QuitWithSuccess
mStep.OnFailAction = _
    SQLDMO_JOBSTEPACTION_TYPE.SQLDMOJobStepAction_QuitWithFailure
mJob.JobSteps.Add(mStep)

```

---

Nachdem der eigentliche Job erstellt ist, kann man an die Ausführungsschritte gehen. In diesem Beispiel ist es nur ein Schritt, aber auch der muss explizit als solcher definiert werden. Dazu dient die Variable `mStep`, die als Typ `SQLDMO.JobStep` angelegt wird. Der Name kann frei vergeben werden und wird hier mit »1st step« belegt. Die `StepID` ist eine 32 Bit Integerzahl, die nicht zwangsweise mit 1 beginnen muss. Hat man mehrere Schritte zu absolvieren, kann man mit den Eigenschaften `OnFailStep` und `OnSuccessStep` die nächsten Schritte angeben.

Die Aktion, die in diesem Schritt ausgeführt werden soll, wird über die Kombination `SubSystem / Command` gesteuert. Neben `TSQL` kann man unter anderem auch `CmdExec` und `ActiveScripting` angeben. Mit `CmdExec` sind beliebige externe Programme gemeint, die einen Exitcode zurückgeben. Mit dem Exitcode kann dann über das weitere Vorgehen entschieden werden. Wird `ActiveScripting` gewählt, so muss in der Eigenschaft `Command` das gesamte Skript abgelegt werden. Ähnliches gilt für die Wahl von `TSQL`: Hierbei muss die `TSQL`-Anweisung in `Command` gespeichert werden. In dieser Methode wird der Befehl `BACKUP DATABASE ...` abgesetzt.

Danach ist die Aktion für diesen Schritt festgelegt und der Job kann verlassen werden, da es sich ja um den einzigen Schritt handelt. Die möglichen Aktionen sind in Tabelle 37 aufgeführt.

| Name                                             | Beschreibung                                                   |
|--------------------------------------------------|----------------------------------------------------------------|
| <code>SQLDMOJobStepAction_Unknown</code>         | Die Schrittlogik hat keine Bedeutung für den aktuellen Schritt |
| <code>SQLDMOJobStepAction_QuitWithSuccess</code> | Erfolgreiches Beenden                                          |
| <code>SQLDMOJobStepAction_QuitWithFailure</code> | Beenden mit Fehler                                             |
| <code>SQLDMOJobStepAction_GotoNextStep</code>    | Zum nächsten Schritt in Reihenfolge gehen                      |
| <code>SQLDMOJobStepAction_GotoStep</code>        | Zu einem bestimmten Schritt gehen                              |

*Tabelle 37: Schritttaktionen für JobStepAction*

---

```

mSchedule.Schedule.FrequencyType = _
    SQLDMO_FREQUENCY_TYPE.SQLDMOFreq_Daily
mSchedule.Schedule.FrequencyInterval = 1
mSchedule.Schedule.ActiveStartDate = CLng(mDate)
mSchedule.Schedule.ActiveStartTimeOfDay = CLng(mTime)

```

---

An dieser Stelle wird dem Scheduler mitgeteilt, wann er diesen Job auszuführen hat. In diesem Beispiel ist dies täglich zum angegebenen Zeitpunkt. Dieser wird als Long übergeben, daher wurde weiter oben der Zeitpunkt in zwei Strings mit speziellem Format zerlegt. Welche Wiederholungsfrequenzen möglich sind, kann man der Tabelle 38 entnehmen.

| Name                       | Beschreibung                                                           |
|----------------------------|------------------------------------------------------------------------|
| SQLDMOFreq_Unknown         | Es gibt keine Wiederholung                                             |
| SQLDMOFreq_OneTime         | Der Job wird nur einmal zum angegebenen Zeitpunkt ausgeführt           |
| SQLDMOFreq_Daily           | Der Job wird täglich zum angegebenen Zeitpunkt ausgeführt              |
| SQLDMOFreq_Weekly          | Wöchentlich                                                            |
| SQLDMOFreq_Monthly         | Monatlich                                                              |
| SQLDMOFreq_MonthlyRelative | Relativer Wert innerhalb eines Monats. Siehe auch SQLDMO_MONTHDAY_TYPE |
| SQLDMOFreq_Autostart       | Job wird automatisch beim Start des SQL-Server-Agents durchgeführt     |
| SQLDMOFreq_OnIdle          | Job wird durchgeführt, wenn der Rechner nicht so viel zu tun hat       |
| SQLDMOFreq_Valid           | Für Abfragen auf die Gültigkeit der obigen Werte nutzbar               |

Tabelle 38: Wiederholungszeiträume für Datenbank-Jobs

Dieser Methode wird kein Zeitpunkt übergeben, wann der Job zum letzten Mal durchgeführt werden soll. Dieser Zeitpunkt wird durch

```
mSchedule.Schedule.ActiveEndDate = 20991231
mSchedule.Schedule.ActiveEndTimeOfDay = 235959
```

auf ein Datum gelegt, welches man für die Zwecke dieser Methode als »immer« übersetzen kann.

```
mJob.ApplyToTargetServer(mServerString)
mJob.BeginAlter()
mJob.JobSchedules.Add(mSchedule)
mJob.DoAlter()
mJob.Start()
End Sub
```

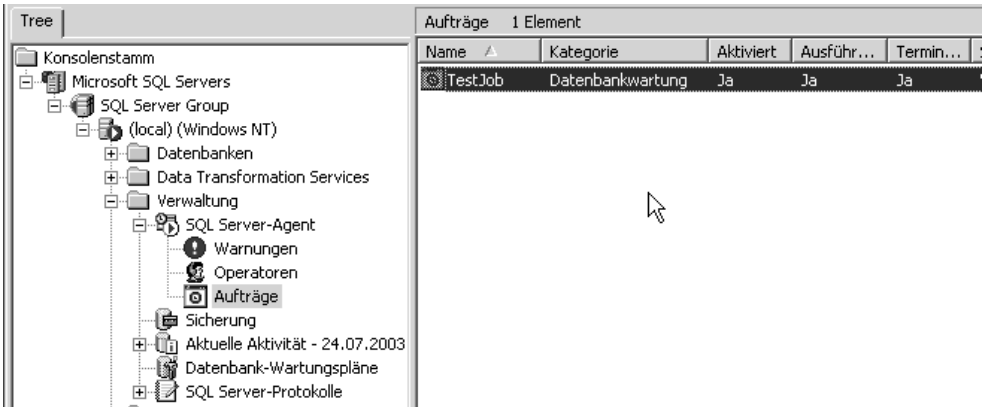


Abbildung 223: Per Programm eingefügter Job im Enterprise-Manager

Abschließend werden die Angaben über die Wiederholung der Schedulingliste übergeben und der Auftrag gestartet. Der Job steht damit in der Warteschlange und wird beim nächsten zutreffenden Termin das erste Mal ausgeführt.

In Abbildung 223 kann man einen mit dieser Methode erstellten Job sehen. Er wurde mit Hilfe des Testprogrammes erstellt, welches Sie auch auf der CD finden. Dieses Programm ruft die oben geschilderte Methode über eine Ereignismethode für den Button TESTJOB auf.

```
Try
    mServer.CreateJob("TestJob", DateTime.Now)
Catch ex As Exception
    MessageBox.Show(ex.ToString, "Fehler", MessageBoxButtons.OK, _
        MessageBoxIcon.Error)
End Try
```

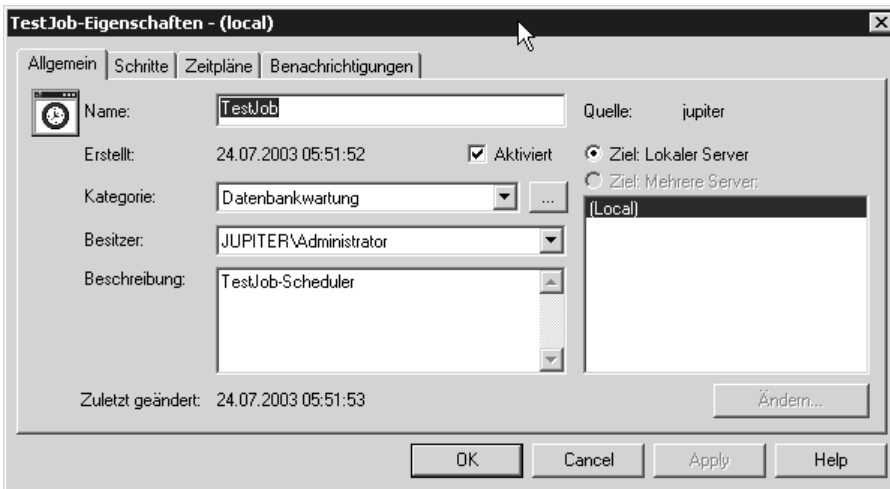


Abbildung 224: Nähere Angaben zum Testjob im Beispielprogramm

In den Abbildungen Abbildung 224 bis Abbildung 226 kann man erkennen, was man mit einer Zeile Programmcode und der hier entwickelten Klasse erreichen kann.

In Abbildung 225 ist der einzige Schritt, den dieser Job hat, so zu sehen, wie ihn der Enterprise-Manager wahrnimmt.

Wie man deutlich an diesem Beispiel erkennt, kann man die Dinge, die man sich im SQL-Server-Enterprise-Manager »zusammenklickt«, auch gut mittels eines Programms durchführen und so den Personen, die mit der MSDE arbeiten, effektiv unter die Arme greifen.

## 255 Auflistung der vorhandenen Jobaufträge

Hat man mehrere Aufträge dem SQL-Server übergeben, will man sich irgendwann einmal die Liste der Server-Jobs anzeigen lassen. Man weiß ja auch nicht, was die Kollegen während des Urlaubs so gemacht haben J.



Abbildung 225: Schrittdefinition des Programmes im Enterprise-Manager

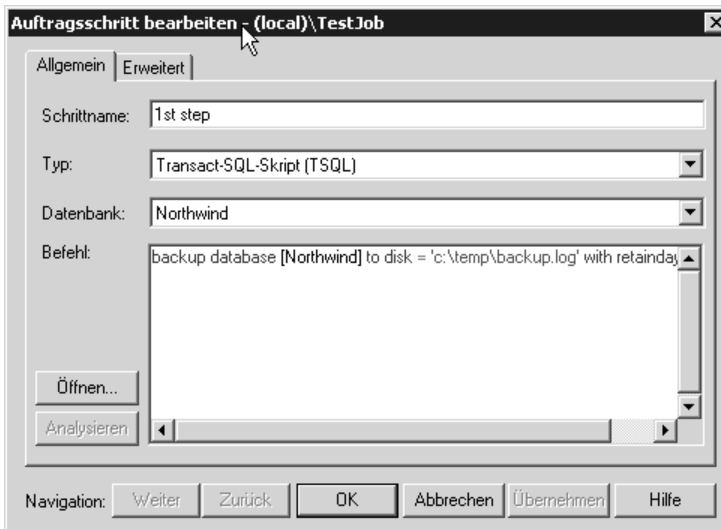


Abbildung 226: TSQL-Anweisung des Jobs

```

Public Function JobList() As ArrayList
    Dim mJob As SQLDMO.Job = New SQLDMO.Job
    Dim mCount As Integer
    Dim i As Integer
    Dim mJobList As ArrayList = New ArrayList

    If mServer.Status <> SQLDMO_SVCSTATUS_TYPE.SQLDMOSvc_Running Then
        Throw New System.ApplicationException( _
            "DMO: Server muss aktiv sein")
    End If

    mCount = mServer.JobServer.Jobs.Count
    For i = 1 To mCount

```

Listing 442: Liste der Server-Jobs

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Propor-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

---

```

    mJob = mServer.JobServer.Jobs.Item(i)
    mJobList.Add(mJob.Name)
Next
Return mJobList
End Function

```

---

*Listing 442: Liste der Server-Jobs (Forts.)*

Die Liste wird über eine `ArrayList` dem aufrufenden Programm zurückgegeben. Die Programmlogik dürfte mittlerweile geläufig sein. Interessiert man sich für einen konkreten Job, kann man sich mit ähnlichen Methoden wie in Kapitel [12.17](#) die Angaben über die entsprechenden Database Management Objects holen.

## 256 Tabellenindizes

In ADO.NET hat sich in der Verarbeitung von Datensätzen einiges grundlegend geändert. So gibt es keine serverseitigen Cursor mehr. Alles liegt beim Client. Eine Verbindung zum Server findet nur noch beim Lesen und Schreiben statt – nachdem typischerweise mehrere Datensätze bearbeitet wurden. Die Situation entspricht mehr dem Batch-Update des »alten« ADO.

Dies hat direkte Auswirkungen auf das Erstellen neuer Datensätze. Da ein Ausschnitt der Datenbank im Hauptspeicher des Clients vorgehalten wird (`Dataset`), werden neue Datensätze in diesem `Dataset` erstellt. Die eigentliche Datenbank bekommt von diesen Aktionen nichts mit. Erst wenn alle neuen und geänderten Datensätze beispielsweise mit der `Update`-Methode des `SqlDataAdapter`'s der Datenbank mitgeteilt werden, können diese abgespeichert werden. Da zwischenzeitlich eine andere Person ebenfalls einen von Ihnen bearbeiteten Datensatz geändert haben kann, wird das `Update` der Datensätze aufwändiger als bei den synchronen serverseitigen Cursor des ADO.

Kompliziert wird es, setzt man die beliebten Auto-Felder einer Datenbank ein. Jede größere Datenbank kennt solche speziellen Felder einer Tabelle, deren Wert sich automatisch um einen festgelegten Betrag erhöht, wird eine neue Zeile in die Tabelle eingefügt. Mit diesen Feldern wurden gerne Hauptschlüssel (Primary Key)-Felder definiert. Durch die synchrone Bearbeitung ergaben sich keine Probleme. Wenn Sie aber Datensätze ohne Verbindung zur Datenbank neu anlegen, fehlen Ihnen diese automatisch erzeugten Werte. Nun kann man sich beim SQL-Server von Microsoft diese Werte mit `@IDENTITY`, `IDENT_CURRENT` oder `SCOPE_IDENTITY` holen, doch liefern diese nur die letzte vergebene Nummer. Ein Ausweg liegt in der Benutzung des `OUTPUT`-Parameters in einer Stored-Procedure. Ein Beispiel ist in Listing 443 aufgeführt. Wird dem zuständigen `SqlCommand` der Parameter-Abfrage für den Parameter `NewID` mitgeteilt, dass die Eigenschaft `.Direction = ParameterDirection.Output` ist, erhält man die von der Datenbank erzeugte ID zurück. In einfachen Konstellationen mag dies noch praktikabel sein, aber schauen Sie mal auf die Abbildung 227. Wenn Sie jetzt alle automatischen Schlüssel haben wollen, die bei einem neuen Patienten erzeugt werden müssen, ahnen Sie, dass die automatischen Schlüssel bei ADO.NET nicht der Weisheit letzter Schluss sind. Wird der Patient auf einer Station aufgenommen (keine zentrale Patientenaufnahme), ergibt sich folgendes Bild: Da der Patient zum ersten Mal kommt, muss ein neuer Eintrag in die Tabelle `Patient` erfolgen. Es handelt sich gleichzeitig um einen Aufenthalt, also auch dort ein neuer Eintrag, wobei der Schlüssel für die Tabelle `Patient` bekannt sein muss. Der Patient wird auf eine Station gelegt, also ein Eintrag in die Tabelle `Verlegung`, wobei der ... Idealerweise hat der Patient auch die Einweisungsdiagnose des Hausarztes dabei, also ein Eintrag in die Tabelle `Diagnosen`, wobei ...

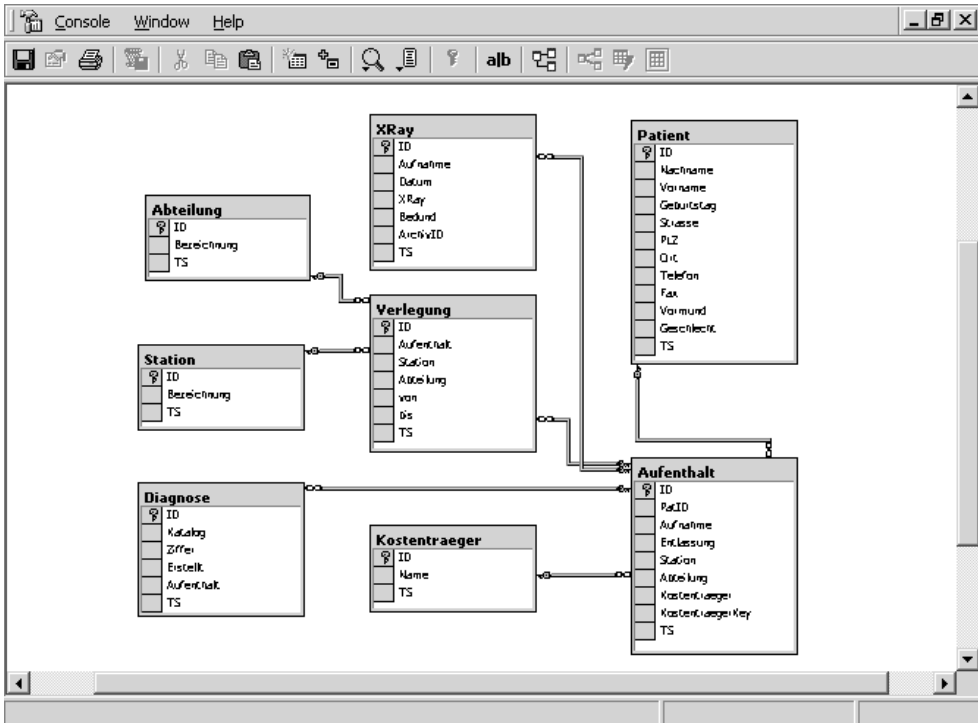


Abbildung 227: Stark vereinfachtes Datenbankschema für Patienten

Warum also nicht eindeutige Schlüssel auf der Client-Seite erzeugen, die anschließend problemlos in der Datenbank abgelegt werden können. Diese Schlüssel sollten so erstellt werden, dass sie mit keinem weiteren Schlüssel kollidieren können. Diese Möglichkeit gibt es, und Microsoft hat sogar einen eigenen Feldtyp in der Datenbank dafür hinterlegt: den GUID-Global Unique Identifier.

```
CREATE PROCEDURE InsertStation
(
  @BezeichnungIn Varchar(20)
  @NewID Integer OUTPUT
)

AS

INSERT INTO Station (Bezeichnung)
Values (@BezeichnungIn)

SET @NewID = SCOPE_IDENTITY()
```

Listing 443: Nutzung von SCOPE\_IDENTITY für einen Index

Wie eine solche Tabelle mit Hilfe von T-SQL erstellt wird, zeigt Listing 444. Das Feld ID wird mit dem Typ uniqueidentifier erstellt, einem 16 Byte langen Feld, welches einen GUID aufnehmen kann.

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Propor-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

---

```

CREATE TABLE [dbo].[XRay] (
    [ID] [uniqueidentifier] NOT NULL ,
    [Aufnahme] [uniqueidentifier] NULL ,
    [Datum] [datetime] NULL ,
    [XRay] [image] NULL ,
    [Befund] [varchar] (2000)
        COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
    [ArchivID] [varchar] (20)
        COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
    [TS] [timestamp] NULL
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO

```

---

*Listing 444: Tabelle mit GUID-Feld erstellen*

Wie ein solcher Wert erzeugt und abgespeichert wird, ist in Listing 444 zu sehen. Verfolgen Sie dafür den Weg der Variablen `DSGUID` durch die Funktion. Auf diese Weise hat man alle Probleme, die durch die automatischen Schlüssel entstehen können, umgangen, und dies bei einfacherer Handhabung.

## 257 Bilder in Tabellen abspeichern

Will man Bilder in einer Datenbank abspeichern, so ging dies lange Zeit nicht. Seit einiger Zeit hat der SQL-Server einen Datentyp, um diese Funktionalität zu realisieren. In Listing 443 wird das Tabellenfeld `Xray` mit diesem Datentyp `image` erstellt. Wie ein Bild in diesem Tabellenfeld abgespeichert wird, ist dem Listing 445 zu entnehmen.

---

```

Protected Sub btnCreate_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnCreate.Click

    Dim SQLQuery As String
    Dim DT As DataTable
    Dim DA As SqlDataAdapter
    Dim DR As DataRow
    Dim CB As SqlCommandBuilder
    Dim DSGUID As New Guid
    Dim PictureStream As FileStream
    Dim RAMFile As MemoryStream

    SQLQuery = "Select id, aufnahme, datum, xray, befund, archivid" _
        + " from xray where aufnahme = '" + PatID + "'"

    ' Dataadapter mit Command erstellen
    DA = New SqlDataAdapter(SQLQuery, Conn)

    ' CommandBuilder-Objekt erzeugen, setzt PK voraus
    CB = New SqlCommandBuilder(DA)

    ' Tabelle zur Verarbeitung
    DT = New DataTable

```

---

*Listing 445: Datensatz mit GUID und Bild erstellen*



```

' DataAdapter mit dem Schema der SQL-Tabelle füllen
' Daten werden hier nicht benötigt
DA.FillSchema(DT, SchemaType.Source)

' Neue Zeile erzeugen
DR = DT.NewRow()

' GUID als PK erzeugen
DSGUID = Guid.NewGuid

' Datei im Speicher anlegen
RAMFile = New MemoryStream

' Zugriff auf Bilddatei mittels Stream
PictureStream = New FileStream(PicName, FileMode.Open, _
    FileAccess.Read, FileShare.Read)

' Zwecks Lesen der Datei
'PictureReader = New StreamReader(PictureStream)
Dim PictureBuffer(CType(PictureStream.Length, Integer)) As Byte

' Abspeichern des Dateiinhalts in einem Byte-Array
' Größe der Datei darf den Maximalwert von Integer
' nicht überschreiten
PictureStream.Read(PictureBuffer, 0, _
    CType(PictureStream.Length, Integer))

' Zeile füllen
DR("id") = DSGUID
DR("aufnahme") = PatID
DR("datum") = dtPicture.Value
DR("befund") = txtBefund.Text
DR("archivid") = txtArchiv.Text
DR("xray") = PictureBuffer

' Zeile der Tabelle hinzufügen
DT.Rows.Add(DR)

' Den Insert-Befehl des CommandBuilders anzeigen
txtCB.Text = CB.GetInsertCommand.CommandText

' Im Server abspeichern
Try
    DA.Update(DT)
    DA.Dispose()
Catch ex As Exception
    Debug.WriteLine(ex.Message)
    Stop
End Try

End Sub

```

Der Klasse werden die Patientennummer `PatID` und der Verzeichnispfad des Bildes `PicName` übergeben. Die Funktion dient dem Hinzufügen eines Eintrages in die Tabelle `XRay`, so dass der Datenadapter nur die Tabellenstruktur von der Datenbank holen muss. Nach der Erzeugung eines neuen Primärschlüssels mit der Methode `GUID.NewGuid` wird ein `FileStream` auf die Bilddatei erstellt. Da der SQL-Server Bilder als Byte-Feld abspeichert, wird ein entsprechend dimensioniertes Byte-Array erzeugt und anschließend das Bild in diesem Byte-Array abgelegt. Dieses Array wird mit den anderen Feldern der Tabelle in der neuen Zeile gespeichert.

Der mit `CommandBuilder` erzeugte SQL-Befehl zum Abspeichern des Datensatzes wird in einem Textfeld dargestellt. Wie der Name der Methode `GetInsertCommand` vermuten lässt, existieren für die anderen Befehle ähnlich lautende Methoden:

Die Abbildung 228 zeigt die Eingabemaske für die Bilddaten der Datenbank.

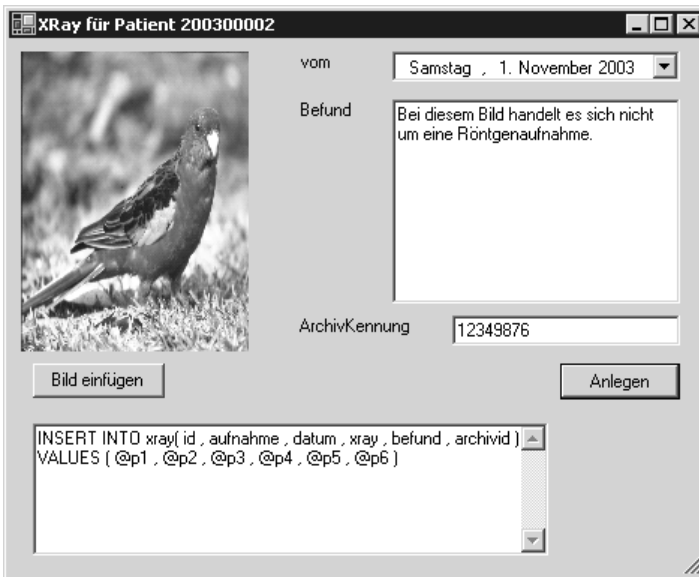


Abbildung 228: Datenbankmaske für das Speichern von Bildern

## 258 DataGrid füllen

Der Vollständigkeit halber ist hier noch die Funktion aufgeführt, die es der Maske aus Abbildung 228 ermöglicht, die Patientennummer zu übernehmen. Hierzu wird in Listing 446 die Verbindung zur Datenbank aufgenommen und eine SQL-Abfrage durchgeführt.

```
Private Sub btnLoad_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnLoad.Click

    Dim SQLQuery As String
    Dim DA As SqlDataAdapter
    Dim DS As DataSet
    Dim DT As DataTable
```

Listing 446: DataGrid mit dem Ergebnis einer Abfrage füllen

```

' Verbindung zur lokalen Datenbank
Conn = New _
    SqlConnection("Server=(local); Database=Southwind; " + _
        "Trusted_Connection=Yes")

Conn.Open()

' Für Demo-Zwecke reicht *, sonst ungünstig
SQLQuery = "Select * from aufenthalt"

' Datenbankbefehl erstellen
Dim Command As SqlCommand = New SqlCommand(SQLQuery, Conn)

' benanntes DataSet erstellen
DS = New DataSet("PatientenInfo")

' Dataadapter mit Command erstellen
DA = New SqlDataAdapter(Command)

' Dataset mit Daten des Adapers füllen, Tables(0) wird
' automatisch erstellt
DA.Fill(DS)

' Datagrid mit Dataset verbinden
dgCurrentPatient.DataSource = DS.Tables(0)

```

End Sub

Listing 446: DataGrid mit dem Ergebnis einer Abfrage füllen (Forts.)

Das DataSet wird über den SqlDataAdapter mit dem Ergebnis der Abfrage gefüllt. Dabei wird automatisch eine Tabelle erstellt, die über den Index 0 in der Tables-Auflistung des DataSet adressiert werden kann.

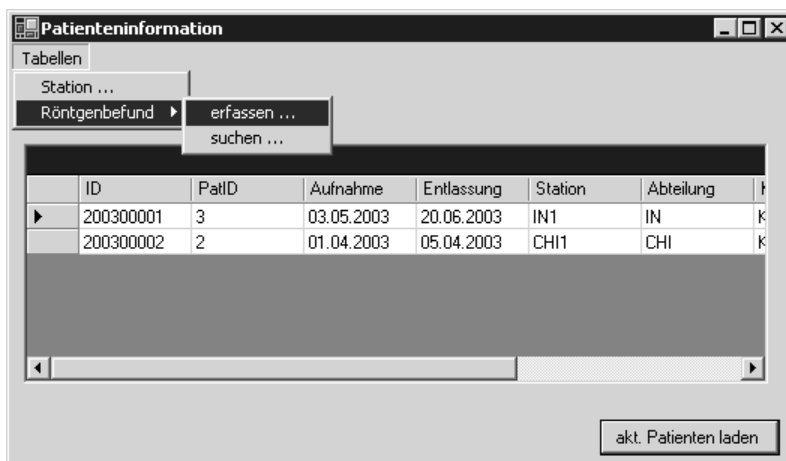


Abbildung 229: DataGrid für die Auswahl von Patienten

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

Diese Tabelle wird dem `DataGrid` als `Quelle` (`DataSource`) zugeordnet. Das Ergebnis ist in Abbildung 228 zu sehen. Zusätzlich ist der Menüeintrag zu erkennen, über den man zu der Bildschirmmaske in Abbildung 230 kommt.

## 259 MDAC-Version ermitteln

Arbeitet man in einem Programm mit Datenbanken, sollte man sich vergewissern, ob beim Kunden die entsprechenden Voraussetzungen gegeben sind. Zu diesen Voraussetzungen gehört auch die richtige MDAC-Version. Diese kann man aus der Registry erhalten. Sie steht unter dem Registry-Schlüssel `HKLM\Software\Microsoft\DataAccess`. Die Funktion `GetMDACVersion` aus Listing 447 ermittelt diese Version und gibt sie als `Typ Version` dem aufrufenden Programm zurück.

---

```
Function GetMDACVersion() As Version
    Dim mKey As RegistryKey
    Dim mVersion As String
    Dim mObj As Object
    Dim mParts As String()
    Dim mVer As Version

    mKey = Registry.LocalMachine.OpenSubKey _
        ("software\microsoft\dataaccess")

    mObj = mKey.GetValue("fullinstallver", "0.0.0.0")
    mVersion = CType(mObj, String)
    mKey.Close()

    mParts = mVersion.Split(Convert.ToChar("."))

    mVer = New Version(Convert.ToInt32(mParts(0)), _
        Convert.ToInt32(mParts(1)), Convert.ToInt32(mParts(2)), _
        Convert.ToInt32(mParts(3)))

    Return mVer
End Function
```

---

*Listing 447: Ermittlung der MDAC-Version*

Hierzu wird der Registry-Schlüssel über die Methode `OpenSubKey` der `Registry`-Klasse geöffnet und dem Objekt `mKey` zugeordnet. Über die Methode `GetValue` wird dann die Versionsangabe ermittelt. Sollte der Registry-Schlüssel nicht vorhanden sein, wird der Wert »0.0.0.0« zurückgeliefert, die Voreinstellung der Methode `GetValue`. Der Inhalt des Objektes `mObj` ist in jedem Fall eine Zeichenkette, obwohl das Objekt vom `Typ Object` ist. Dieser `Typ` wird von der Methode verlangt. In der Einstellung `strict on` führen alle anderen Angaben zu einem Fehler. Daher wird mit Hilfe der Methode `CType` die Version wieder in eine Zeichenkette umgewandelt.

Diese Zeichenkette wird am ».« aufgetrennt und in den `Typ Version` transformiert, der dann als Rückgabewert für die Funktion benutzt wird.

Ein Beispiel für eine solche Versionsermittlung ist in Abbildung 230 zu sehen. Die Konsolenausgabe erfolgte mit Hilfe der Funktion `Main` aus Listing 448.

```
Sub Main()  
  
    Dim mVersion As Version  
  
    mVersion = GetMDACVersion()  
  
    Console.WriteLine("MDAC - Version = {0}", mVersion.ToString)  
    Console.WriteLine()  
    Console.WriteLine("Major - Number = {0}", mVersion.Major.ToString)  
    Console.WriteLine("Minor - Number = {0}", mVersion.Minor.ToString)  
    Console.WriteLine("Build - Number = {0}", mVersion.Build.ToString)  
    Console.WriteLine("Revision - Number = {0}", mVersion.Revision)  
  
    Console.ReadLine()  
  
End Sub
```

Listing 448: Ausgabe der MDAC-Version auf der Konsole



Abbildung 230: Beispiel für die Ermittlung der MDAC-Version

## 260 Excel als Datenbank abfragen

In der Praxis kommt es sehr häufig vor, dass Daten über Excel-Datenblätter erfasst werden und später ausgewertet werden müssen. Kann und will man sich den Weg über einen Import in eine Datenbank sparen, kann Excel auch als Datenbank angesprochen werden. Die Datenbank wird durch die Arbeitsmappe repräsentiert, während die Tabellen durch die einzelnen Arbeitsblätter dargestellt werden. Die Zeilen und Spalten einer Datenbanktabelle in Excel wieder zu finden dürfte damit logisch nachvollziehbar sein.

Die Verbindung nach Excel wird in Listing 449 über eine `OleDbConnection` und ein `OleDbDataAdapter` realisiert. Bei der SQL-Abfrage ist zu beachten, dass die Tabelle, also das Excel-Arbeitsblatt, in eckigen Klammern eingeschlossen sein muss. Zusätzlich ist dem Namen des Arbeitsblattes ein »\$« anzuhängen. In der Excel-Tabelle stellt die erste Zeile die Feldnamen der Tabelle zur Verfügung.

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

```

Private Sub btnStart_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnStart.Click

    Dim DS As System.Data.DataSet
    Dim mComm As System.Data.OleDb.OleDbDataAdapter
    Dim mConn As System.Data.OleDb.OleDbConnection
    Dim mTable As DataTable
    Dim mRow As DataRow
    Dim mCol As DataColumn

    ' Verbindung nach Excel aufnehmen
    mConn = New System.Data.OleDb.OleDbConnection( _
        "provider=Microsoft.Jet.OLEDB.4.0; " & _
        "data source=" & _
        "e:\projekte\codebook\datenbanken\excel\küche.xls; " & _
        "Extended Properties=Excel 8.0;")

    ' Tabellen entsprechen Arbeitsblättern.
    mComm = New System.Data.OleDb.OleDbDataAdapter( _
        "Select * from [patient$]", mConn)

    ' Dataset für Tabelle erstellen
    DS = New System.Data.DataSet

    DataGrid1.ReadOnly = True

    Try
        mComm.Fill(DS, "KuechenBestellungen")
    Catch ex As Exception
        Throw New ApplicationException("Excel-VB.", ex)
    End Try

    ' Datagrid füllen
    DataGrid1.DataSource = DS.Tables("KuechenBestellungen")
    DataGrid1.Refresh()

    mConn.Close()

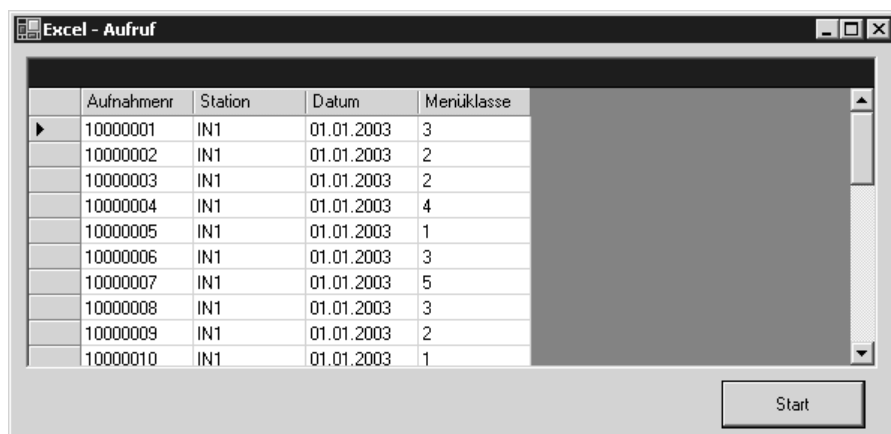
End Sub

```

---

*Listing 449: Excel-Arbeitsmappe als Datenbank abfragen*

Alles weitere verläuft nach dem bekannten Schema für das Füllen eines DataSets. In Abbildung 231 kann man das Ergebnis einer solchen Abfrage sehen.



|   | Aufnahmenr | Station | Datum      | Menüklasse |
|---|------------|---------|------------|------------|
| ▶ | 10000001   | IN1     | 01.01.2003 | 3          |
|   | 10000002   | IN1     | 01.01.2003 | 2          |
|   | 10000003   | IN1     | 01.01.2003 | 2          |
|   | 10000004   | IN1     | 01.01.2003 | 4          |
|   | 10000005   | IN1     | 01.01.2003 | 1          |
|   | 10000006   | IN1     | 01.01.2003 | 3          |
|   | 10000007   | IN1     | 01.01.2003 | 5          |
|   | 10000008   | IN1     | 01.01.2003 | 3          |
|   | 10000009   | IN1     | 01.01.2003 | 2          |
|   | 10000010   | IN1     | 01.01.2003 | 1          |

Abbildung 231: Excel-Tabelle in einem DataGrid

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Propor-  
tyGridDatei-  
systemNetz-  
werk

System

**Daten-  
banken**

XML

Wissen-  
schaftVer-  
schiede-  
nes





XML-Strukturen, wie sie vom W3C (World Wide Web Consortium, <http://www.w3.org/>) standardisiert werden, bilden einen wichtigen Stützpfeiler der .NET-Architektur. Dementsprechend finden sich im Framework viele Klassen, die den Umgang mit XML unterstützen und vereinfachen. Dabei ist die Bearbeitung nicht auf XML-Dateien beschränkt, sondern erstreckt sich auch z.B. auf Streams und Strings.

Anhand einiger praktischer Beispiele wird in diesem Kapitel der Umgang mit den wichtigsten XML-Klassen erläutert.

## 261 Schreiben von XML-Dateien mit dem XmlTextWriter

Mit der Klasse `XmlTextWriter` stellt Ihnen das Framework einen schnellen Mechanismus zur Verfügung, um XML-Strukturen in Streams oder Dateien zu speichern. Die Klasse arbeitet auf dem untersten Level und gestattet nur ein vorwärts gerichtetes Schreiben von Knoten. Elemente und Attribute werden mit verschiedenen Methoden geschrieben. Der Aufbau erfolgt meist nach folgendem Muster:

```
Dim xw As New XmlTextWriter(...)

' Versionsinformation schreiben
xw.WriteStartDocument()

' Root-Element beginnen
xw.WriteStartElement("Rootname")

... Elemente schreiben

' Root-Element abschließen
xw.WriteEndElement()

' Abschließender Aufruf für XmlWriter
xw.WriteEndDocument()

' Xml-Datei schließen
xw.Close()
```

`WriteStartDocument` und `WriteEndDocument` sichern einen syntaktisch korrekten Rahmen für die XML-Datei. Sowohl die XML-Version als auch das Encoding werden eingetragen. Ein aus `WriteStartElement` und `WriteEndElement` bestehendes Aufrufpaar bildet das Root-Element und umschließt die Methodenaufrufe für alle anderen Elemente und Attribute.

Beinhaltet ein Element weder Attribute noch Unterelemente, sondern lediglich Text, dann genügt ein Aufruf von `WriteElementString`. Andernfalls müssen Elemente mit einem Aufrufpaar von `WriteStartElement` und `WriteEndElement` angelegt werden. Nach dem Aufruf von `WriteStartElement` können Attribute und untergeordnete Elemente geschrieben werden.

Leere Elemente, die mit `WriteStartElement` und `WriteEndElement` geschrieben werden, werden in der verkürzten Syntax dargestellt:

---

```
<EinLeeresElement />
```

---

Wenn das nicht erwünscht ist, können Sie durch den Aufruf von `WriteFullEndElement` statt des abschließenden `WriteEndElement` die Ausgabe eines vollständigen End-Tags erzwingen:

---

```
<EinLeeresElement>
</EinLeeresElement>
```

---

Attribute werden meist mittels `WriteAttributeString` geschrieben. Alternativ können Sie auch hier ein Methodenpaar, bestehend aus `WriteStartAttribute` und `WriteEndAttribute`, aufrufen und die Daten getrennt ausgeben.

Text-Daten werden in der Regel mit `WriteString` ausgegeben. Zusätzlich werden andere Methoden für die Ausgabe von Base64-, BinHex- oder CDATA-kodierten binären Daten angeboten.

Eine Reihe von Eigenschaften (siehe Tabelle 39) gestatten Eingriffe in die Formatierung und geben Auskunft über den Status des Writers.

| Eigenschaft | Bedeutung                                                                                            |
|-------------|------------------------------------------------------------------------------------------------------|
| Formatting  | Gibt an, ob die Ausgabe mit oder ohne Einrückungen formatiert werden soll                            |
| IndentChar  | Zeichen, das für die Einrückung verwendet werden soll. Sinnvoll sind hier Leerzeichen und Tabulator. |
| Indentation | Anzahl der IndentChar-Zeichen für die Einrückung                                                     |
| WriteState  | Zustand des Writers                                                                                  |

Tabelle 39: Typische oft verwendete Eigenschaften der `XmlTextWriter`-Klasse

## Wandeln einer CSV-Datei in eine XML-Datei

Ein kleines Beispiel soll den Umgang mit der `XmlTextWriter`-Klasse verdeutlichen. Mit Hilfe eines Tabellenverarbeitungsprogramms wurde eine Textdatei mit Quizfragen erstellt. Jede Aufgabe wird in einer eigenen Zeile dargestellt, die folgende Struktur aufweist:

---

```
Schwierigkeitsgrad;Frage;Antwort1;Antwort2;Antwort3;Antwort4;Lösung
```

---

Alle Spalten werden mit Semikolons voneinander getrennt. Falls vorhanden, kann eine zusätzliche Spalte mit einer Erläuterung folgen. Zwei Beispiele für den Aufbau der Datei (die Zeilen sind hier wegen der eingeschränkten Seitenbreite umgebrochen worden):

---

```
9;Wem gehörte nie die Insel Kuba;Spanien;England;Frankreich;USA;C 12;Wo ist das
nördlichste Vorkommen von Pinguinen;südl. Polarkreis;südl. Wendekreis;Äquator;nördl.
Polarkreis;C;Galapagos Inseln
```

---

Diese Textdatei wird zeilenweise gelesen und jede Zeile mittels der String-Funktion `Split` in ihre Bestandteile zerlegt (Array Felder, siehe Listing 450). Nach den einleitenden Methodenaufrufen

(s.o.) wird für jedes Aufgaben-Element mit `WriteStartElement` das entsprechende XML-Element eingeleitet. Es folgen die Ausgaben von Schwierigkeitsgrad und Lösung als Attribut. Die Frage wird mit `WriteElementString` ohne Attribute gespeichert, während die vier Antwortmöglichkeiten jeweils mit der Folge `WriteStartElement`, `WriteAttributeString`, `WriteString` und `WriteEndElement` erzeugt werden.

Sofern vorhanden, wird eine Erläuterung mit `WriteElementString` angehängt. Nach Abschluss des Root-Elements und des Dokuments steht die XML-Datei zur Verfügung. Einen Ausschnitt aus dieser Datei, der den Inhalt der beiden Beispielzeilen wiedergibt, sehen Sie in Listing 451.

---

```
Public Sub CSVtoXML(ByVal csv As String, ByVal xml As String)

    ' XML-Datei mit XmlWriter zum Schreiben öffnen
    Dim xw As New XmlTextWriter(xml, _
        System.Text.Encoding.Default)

    ' Einrückung einschalten
    xw.Formatting = Formatting.Indented

    ' Versionsdeklaration schreiben
    xw.WriteStartDocument()

    ' Root-Element beginnen
    xw.WriteStartElement("Aufgaben")

    ' Kommentar
    xw.WriteComment("Liste der Aufgaben")

    ' Öffnen der CSV-Datei
    Dim sr As New StreamReader(csv, _
        System.Text.Encoding.Default)

    Dim zeile As String

    ' Datei zeilenweise bearbeiten
    Do
        ' Zeile lesen
        zeile = sr.ReadLine()

        ' auf Dateiende prüfen
        If zeile Is Nothing Then Exit Do

        ' Zeile in Spalten zerlegen, Trennzeichen ist das Semikolon
        Dim Felder() As String = zeile.Split(";";c)

        ' Neues Aufgabenelement einleiten
        xw.WriteStartElement("Aufgabe")

        ' Attribut 'Schwierigkeitsgrad'
        xw.WriteAttributeString("Schwierigkeitsgrad", Felder(0))
```

---

Listing 450: Schreiben einer XML-Datei mit Quizaufgaben aus einer CSV-Datei

---

Basics

---

Datum/  
Zeit

---

Anwen-  
dungen

---

Zeich-  
nen

---

Bild-  
bearbei-  
tung

---

Win-  
dows  
Forms

---

Controls

---

Proper-  
tyGrid

---

Datei-  
system

---

Netz-  
werk

---

System

---

Daten-  
banken

---

**XML**

---

Wissen-  
schaft

---

Ver-  
schiede-  
nes

---

```

' Attribut 'Lösung'
xw.WriteAttributeString("Lösung", Felder(6))

' Fragenelement schreiben
xw.WriteElementString("Frage", Felder(1))

' Vier Antwortmöglichkeiten
For i As Integer = 0 To 3
    ' Antwortelement einleiten
    xw.WriteStartElement("Antwort")
    ' Attribut 'ID'
    xw.WriteAttributeString("ID", "ABCD".Substring(i, 1))
    ' Text des Elements schreiben
    xw.WriteString(Felder(2 + i))
    ' Antwortelement abschließen
    xw.WriteEndElement()
Next

' Optionales Erläuterungselement
If Felder.Length > 7 Then
    If Felder(7) <> "" Then
        ' Erläuterungselement schreiben
        xw.WriteElementString("Erläuterung", Felder(7))
    End If
End If

' Aufgabenelement abschließen
xw.WriteEndElement()
Loop

' CSV-Datei schließen
sr.Close()

' Aufgabenelement abschließen
xw.WriteEndElement()

' Abschließender Aufruf für XmlWriter
xw.WriteEndDocument()

' Xml-Datei schließen
xw.Close()

```

End Sub

---

*Listing 450: Schreiben einer XML-Datei mit Quizaufgaben aus einer CSV-Datei (Forts.)*

---

```

<?xml version="1.0" encoding="Windows-1252"?>
<Aufgaben>
  <!--Liste der Aufgaben-->

  ...

```

---

*Listing 451: Ausschnitt aus der erzeugten XML-Datei*

```
<Aufgabe Schwierigkeitsgrad="9" Lösung="C">
  <Frage>Wem gehörte nie die Insel Kuba</Frage>
  <Antwort ID="A">Spanien</Antwort>
  <Antwort ID="B">England</Antwort>
  <Antwort ID="C">Frankreich</Antwort>
  <Antwort ID="D">USA</Antwort>
</Aufgabe>
<Aufgabe Schwierigkeitsgrad="12" Lösung="C">
  <Frage>Wo ist das nördlichste Vorkommen von Pinguinen</Frage>
  <Antwort ID="A">südl. Polarkreis</Antwort>
  <Antwort ID="B">südl. Wendekreis</Antwort>
  <Antwort ID="C">Äquator</Antwort>
  <Antwort ID="D">nördl. Polarkreis</Antwort>
  <Erläuterung>Galapagos Inseln</Erläuterung>
</Aufgabe>

...

</Aufgaben>
```

Listing 451: Ausschnitt aus der erzeugten XML-Datei (Forts.)

## 262 Lesen von XML-Dateien mit dem XmlTextReader

Analog zum XmlTextWriter stellt Ihnen das Framework mit der Klasse XmlTextReader die Basis-Funktionalität für das Lesen von XML-Strukturen zur Verfügung. Auch hier ist wieder nur eine vorwärts gerichtete Abarbeitung möglich. Einmal gelesene Knoten können nicht erneut durchlaufen werden, ohne den Reader neu zu starten.

Mit der unter dem Namen »depth first« bekannten Vorgehensweise wird ein Knoten zuerst in der Tiefe durchsucht, bevor die Geschwisterknoten bearbeitet werden. Das Lesen erfolgt nach dem Konzept des »akutellen Knotens«. Ein Knoten muss dabei nicht zwangsläufig ein Element sein. Tabelle 40 zeigt die Liste der Knotentypen, nach denen differenziert wird.

| Knotentyp      | Bedeutung                                         |
|----------------|---------------------------------------------------|
| Attribute      | Ein Attribut (z.B. ID= »A«)                       |
| CDATA          | Ein CDATA-Abschnitt                               |
| Comment        | Kommentar                                         |
| Document       | Das Stamm-Objekt                                  |
| Text           | Der Textinhalt eines Knotens                      |
| Whitespace     | Leerraum (Leerzeichen, Zeilenumbruch, Tabulator)  |
| XmlDeclaration | Die XML-Deklaration (z.B. <?xml version=»1.0«...> |

Tabelle 40: Die wichtigsten Knotentypen beim Lesen mit dem XmlTextReader

Ein typischer Aufbau zum Lesen einer XML-Datei mit Hilfe des `XmlTextReader`s sieht so aus:

---

```
' Neue Instanz anlegen
Dim xr As New XmlTextReader(...)

' Alle Knoten der Datei lesen
Do While xr.Read()

    ' Typ des Knotens prüfen
    Select Case xr.NodeType
        Case XmlNodeType.Element
            ' Bearbeitung Element...
        Case XmlNodeType.Attribute
            ' Bearbeitung Attribut ...
    End Select
Loop

' Reader wieder schließen
xr.Close()
```

---

Mit dem Aufruf der Methode `Read` wird jeweils der nächste Knoten eingelesen. Der Rückgabewert ist `True`, bis kein Knoten mehr gefunden wird. Die Eigenschaft `NodeType` gibt Auskunft über den Knotentyp (vergleiche Tabelle 40). Auf Basis dieser Information kann die Bearbeitung des Knotens erfolgen.

Eine Vielzahl von Lese-Methoden wie `GetAttribute`, `ReadElementString`, `ReadString` usw. bieten komfortable Zugriffsmöglichkeiten auf die untergeordneten Elemente eines Knotens. Eigenschaften wie `Name`, `Value`, `AttributeCount`, `HasAttributes`, `HasValue` sowie `NodeType` geben Auskunft über den Aufbau des Knotens. Die Namen der Eigenschaften und Methoden sind weitestgehend selbsterklärend und werden daher hier nicht weiter erläutert.

Anhand eines kleinen Beispiels soll erläutert werden, wie eine XML-Datei nach bestimmten Daten mithilfe eines `XmlTextReader`s durchsucht werden kann. Hierbei sollen aus der in Listing 451 gezeigten XML-Datei alle Aufgabensätze zu einem vorgegebenen Schwierigkeitsgrad ausgewählt und gespeichert werden. Anschließend wird per Zufallsgenerator eine Aufgabe ausgelost und zur Beantwortung angezeigt. Listing 452 zeigt die Implementierung.

Nach Öffnen der Datei wird diese in einer `Do`-Schleife Knoten für Knoten abgearbeitet. Handelt es sich bei dem Knoten um ein Element, dann muss untersucht werden, um welche Art von Element es sich handelt. In Frage kommen die Elemente `Aufgaben`, `Aufgabe`, `Frage`, `Antwort` und `Erläuterung`.

Hat das Element den Namen `Aufgabe`, dann wird eine neue Instanz der Hilfsklasse `Task` angelegt, die die Daten der Aufgabe speichern soll. Mit `GetAttribute` wird der Schwierigkeitsgrad ermittelt. Entspricht er nicht dem gesuchten Schwierigkeitsgrad, dann wird die Abarbeitung des aktuellen Elementes durch den Aufruf von `Skip` abgebrochen. Ansonsten wird auch das Attribut `Lösung` eingelesen und in einen Zahlenwert gewandelt.

Bei den Elementnamen `Frage`, `Antwort` und `Erläuterung` werden die jeweiligen Daten mit `GetAttribute` aus den Attributen bzw. mit `ReadString` aus dem Textinhalt des Knotens gelesen und in der `Task`-Instanz gespeichert.

Wird ein Knoten vom Typ `EndElement` mit dem Namen `Aufgabe` gefunden, dann wurde der Aufgaben-Knoten vollständig abgearbeitet bzw. übersprungen. Bei korrektem Schwierigkeitsgrad wird das `Task`-Objekt an die Liste `Tasks` (Instanz von `ArrayList`) angehängt.

Abschließend wird die Datei geschlossen und ein Datensatz zufällig ausgewählt und angezeigt (Abbildung 232).

---

```

' XML-Datei öffnen
Dim xr As New XmlTextReader("Aufgaben.xml")

Dim ts As Task

' Schwierigkeitsgrad
Dim degree As Integer = CInt(NUDDegree.Value)

' Aufgabenliste zum Schwierigkeitsgrad
Dim Tasks As New ArrayList

' Alle Knoten der Datei lesen
Do While xr.Read()

    ' Typ des Knotens prüfen
    Select Case xr.NodeType

        ' Element
        Case XmlNodeType.Element

            ' Elementnamen prüfen
            Select Case xr.Name

                ' Aufgabe: leitet eine neue Aufgabe ein
                Case "Aufgabe"

                    ' Neue Aufgabeninstanz
                    ts = New Task

                    ' Schwierigkeitsgrad einlesen
                    ts.Degree = XmlConvert.ToInt32( _
                        xr.GetAttribute("Schwierigkeitsgrad"))

                    ' Ist es der gesuchte?
                    If ts.Degree <> degree Then
                        ' Nein, weiter zur nächsten Aufgabe
                        xr.Skip()
                    Else
                        ' Lösung abfragen A->0 ... D->3
                        ts.CorrectAnswer = "ABCD".IndexOf( _
                            xr.GetAttribute("Lösung"))
                    End If

                ' Fragestellung
            End Select
        End Select
    End While

```

---

Listing 452: Lesen der Datensätze zu einem bestimmten Schwierigkeitsgrad mithilfe des `XmlTextReader`s

---

Basics

---

Datum/  
Zeit

---

Anwen-  
dungen

---

Zeich-  
nen

---

Bild-  
bearbei-  
tung

---

Win-  
dows  
Forms

---

Controls

---

Proper-  
tyGrid

---

Datei-  
system

---

Netz-  
werk

---

System

---

Daten-  
banken

---

**XML**

---

Wissen-  
schaft

---

Ver-  
schie-  
denes

---

```

        Case "Frage"
            ' Frage lesen
            ts.Question = xr.ReadString()

            ' Eine der vier Antworten
        Case "Antwort"
            ' Antwort lesen und Index ermitteln
            Dim i As Integer = "ABCD".IndexOf( _
                xr.GetAttribute("ID"))

            ts.Answers(i) = xr.ReadString()

            ' Optionale Erklärung
        Case "Erläuterung"
            ts.Description = xr.ReadString()

    End Select

    ' Ende-Tag eines Elementes
    Case XmlNodeType.EndElement
        ' Aufgabe vollständig eingelesen
        If xr.Name = "Aufgabe" Then
            ' Bei korrektem Schwierigkeitsgrad Aufgabe in Liste
            ' übernehmen
            If ts.Degree = degree Then Tasks.Add(ts)
        End If

    End Select

Loop

' Datei wieder schließen
xr.Close()

' Aufgabe zufällig auswählen und anzeigen
Dim n As Integer = New Random().Next(0, Tasks.Count)
CurrentTask = DirectCast(Tasks(n), Task)
LBQuestion.Text = CurrentTask.Question
...
Public Class Task
    Public Degree As Integer
    Public Question As String
    Public Answers(3) As String
    Public CorrectAnswer As Integer
    Public Description As String
End Class

```

---

*Listing 452: Lesen der Datensätze zu einem bestimmten Schwierigkeitsgrad mithilfe des XmlTextReaders (Forts.)*



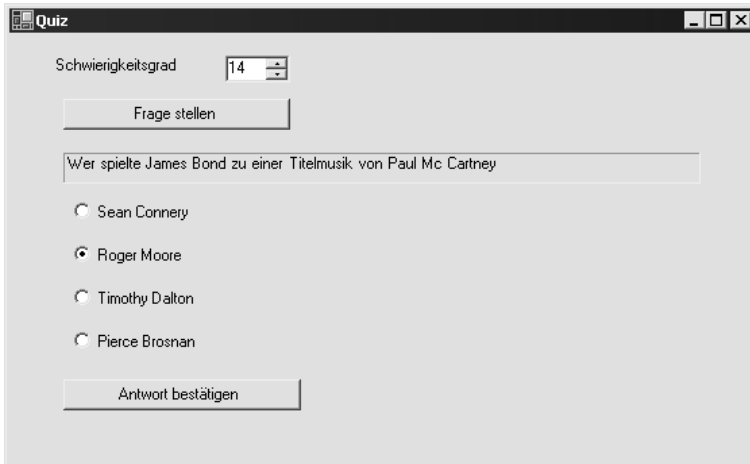


Abbildung 232: Quiz-Aufgabe aus XML-Datei lesen und anzeigen

## 263 Bilder und andere Binärdaten in XML-Dateien speichern

Da XML alle Informationen in Textform abspeichert, müssen binäre Daten grundsätzlich gewandelt werden. Für die Darstellung binärer Daten stehen unter XML die Formate CDATA, BinHex und Base64 zur Verfügung. Base64 wird häufig für die Speicherung von Bildern verwendet. Da das Prinzip für alle Formate gleich ist, soll hier nur der Umgang mit dem Base64-Format erläutert werden. Ferner spielt es keine Rolle, welche Art von Daten gespeichert werden sollen. Exemplarisch für beliebige binäre Daten wird daher hier ein Bild im JPEG-Format gespeichert.

Die Vorgehensweise zum Speichern binärer Daten erfolgt meist nach diesem Muster:

1. Speichern der Informationen in einem Stream
2. Wandeln des Streams in ein Byte-Array
3. Wandeln des Byte-Arrays in z.B. einen Base64-String

Im Beispiel in Listing 453 wird auf Basis einer Bilddatei ein Bitmap-Objekt angelegt. Das Bild wird durch den Aufruf von `Save` in einem angegebenen Format (hier JPEG) in einen MemoryStream geschrieben. Die `MemoryStream`-Instanz lässt sich mittels `ToArray` in ein Byte-Array wandeln. Durch den Aufruf von `WriteBase64` kann dann dieses Byte-Array an der gewünschten Position in der XML-Datei als Base64-String gespeichert werden.

---

*' XML-Datei mit XmlWriter zum Schreiben öffnen*

```
Dim xw As New XmlTextWriter("Bilder.xml", _
    System.Text.Encoding.Default)
```

*' Einrückung einschalten*

```
xw.Formatting = Formatting.Indented
```

*' Versionsdeklaration schreiben*

```
xw.WriteStartDocument()
```

---

Listing 453: Speichern eines Bildes in einer XML-Datei

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Propor-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

**XML**

Wissen-  
schaft

Ver-  
schiede-  
nes

---

```

' Root-Element beginnen
xw.WriteStartElement("Bilder")

' Kommentar
xw.WriteComment("Bilder als Base64")

' Bitmap lesen
Dim bm As New Bitmap("../Farn.jpg")

' MemoryStream anlegen
Dim ms As New MemoryStream

' Bitmap in MemoryStream speichern
bm.Save(ms, System.Drawing.Imaging.ImageFormat.Jpeg)

' In Byte-Array konvertieren
Dim bytes() As Byte = ms.ToArray()

' Element für Bild anlegen
xw.WriteStartElement("Bild1")

' Als Base64-String speichern
xw.WriteBase64(bytes, 0, bytes.Length)

' Element abschließen
xw.WriteEndElement()

' Aufgabenelement abschließen
xw.WriteEndElement()

' Abschließender Aufruf für XmlWriter
xw.WriteEndDocument()

' Xml-Datei schließen
xw.Close()

```

---

*Listing 453: Speichern eines Bildes in einer XML-Datei (Forts.)*

In Listing 454 sehen Sie die erzeugte XML-Datei (der Code für das Bild wurde abgeschnitten). Ein Bild wird so als Text-Element gespeichert und kann in gleicher Form wieder eingelesen werden (siehe nächstes Rezept).

---

```

<?xml version="1.0" encoding="Windows-1252"?>
<Bilder>
  <!--Bilder als Base64-->
  <Bild1>9j/4AAQSkZJRgABAgEAlgCWAAD/4QE2RXhpZgAATU0AKgAAAAg...
  </Bild1>
</Bilder>

```

---

*Listing 454: XML-Datei mit gespeichertem Bild*

Da die Binärinformationen in einen lesbaren Text konvertiert werden müssen, wird der benötigte Speicherplatz stets erheblich größer sein als der für die binäre Darstellung. Binärinformationen sollten daher nur in geringem Umfang in XML-Dateien aufgenommen werden.

Auch kümmert sich der `XmlTextWriter` nicht um die Formatierung der Base64-Zeichenketten. Die gesamte Zeichenfolge wird als eine einzige Zeile abgespeichert. Möchten Sie aus Gründen der Lesbarkeit die Daten ab einer bestimmten Spalte umbrechen, dann können Sie eine leicht veränderte Variante des obigen Codes verwenden (Listing 455).

Hier wird zunächst das Byte-Array in einen Base64-String gewandelt. Zu diesem Zweck wird die Methode `Convert.ToBase64String` aufgerufen. Dem generierten String werden im Abstand von 68 Zeichen Zeilenumbrüche hinzugefügt. Der so umgebrochene Text wird dann mittels `WriteString` im Element gespeichert.

```
...
' In Byte-Array konvertieren
Dim bytes() As Byte = ms.ToArray()

' In Base64-String konvertieren
Dim t As String = Convert.ToBase64String(bytes)
Dim wrapped As String
' Zeilenumbrüche vorsehen
Do
    wrapped = wrapped & Environment.NewLine & t.Substring(0, 68)
    t = t.Substring(68)
Loop While t.Length > 68
wrapped = wrapped & Environment.NewLine & t & Environment.NewLine

' Element für Bild anlegen
xw.WriteStartElement("Bild1")

' Umgebrochenen Base64-String speichern
xw.WriteString(wrapped)
...
```

*Listing 455: Variante von Listing 454, bei der der Base64-String in mehrere Zeilen umgebrochen wird*

Die XML-Datei erhält dann das in Listing 456 gezeigte Aussehen. Umgebrochen wurde hier nach der 68. Spalte. Leerzeichen und Zeilenumbrüche werden später beim Einlesen und Dekodieren des Base64-Strings ignoriert.

```
<?xml version="1.0" encoding="Windows-1252"?>
<Bilder>
  <!--Bilder als Base64-->
  <Bild1>
/9j/4AAQSkZJRgABAgEAlgcWAAD/4QE2RXhpZgAATU0AKgAAAagABwESAAMAAAABAAEA
AAEaAAUAAAABAAAyGEBAAUAAAABAAAagEoAAMAAAABAAIAAAExAAIAAAADAAAAcgEy
AAIAAAUAAAAj4dpAAQAAAAABAAAapAAAAAAAACWAAAAAQAAAJYAAAABQWRvYmUgUGhv
...
tDZARx96jmoorRfEIgNA6UUVYxB98fUUUUUDP//Z
</Bild1>
</Bilder>
```

*Listing 456: XML-Datei mit umgebrochenen Base64-Informationen*

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Proper-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

## 264 Bilder und andere Binärdaten aus XML-Dateien lesen

Wurden binäre Daten, wie im vorangegangenen Rezept beschrieben, z.B. im Base64-Format gespeichert, dann können sie beim Lesen aus einer XML-Datei wieder in ihre Ursprungsform zurückgewandelt werden. Das Bearbeitungsmuster entspricht in umgekehrter Reihenfolge dem des Schreibens binärer Daten:

1. Lesen des Base64-Strings aus einem Text-Element
2. Konvertierung in ein Byte-Array
3. eventuell Konvertierung in einen Stream
4. Konvertierung in das ursprüngliche Format

Listing 457 zeigt den benötigten Code, um die XML-Datei des vorherigen Beispiels (Listing 456) zu lesen und das enthaltene Bild in einer PictureBox anzuzeigen.

---

```
' XML-Datei öffnen
Dim xr As New XmlTextReader("Bilder.xml")

' Knoten auswählen
xr.ReadStartElement("Bilder")
xr.ReadStartElement("Bild1")

' Inhalt als String lesen
Dim b64 As String = xr.ReadString()

' Konvertierung in Byte-Array
Dim bytes() As Byte = Convert.FromBase64String(b64)

' Konvertierung in MemoryStream
Dim ms As New MemoryStream(bytes)

' Bitmap aus MemoryStream erstellen
Dim bm As New Bitmap(ms)

' Bitmap in PictureBox anzeigen
PictureBox1.Width = PictureBox1.Height * bm.Width \ bm.Height
PictureBox1.Image = bm

' XML-Datei schließen
xr.Close()
```

---

Listing 457: Lesen eines im Base64-Format gespeicherten Bildes aus einer XML-Datei

Nach Positionierung des Readers auf das gewünschte Element wird dessen Inhalt als String gelesen. Durch Aufruf von `Convert.FromBase64String` wird dieser Text dekodiert und die binären Daten in einem Byte-Array zurückgegeben. Die Wandlung in ein Bitmap-Objekt erfolgt hier wieder über einen `MemoryStream`. Bei anderen Datentypen können Sie diesen Schritt gegebenenfalls überspringen, wenn die Daten direkt aus dem Byte-Array extrahiert werden können.

Abschließend wird das Bild in einer PictureBox angezeigt (Abbildung 233) und die XML-Datei geschlossen.



Abbildung 233: Anzeigen eines aus einer XML-Datei gelesenen Fotos

Zwar bietet die Klasse `XmlTextReader` auch die Methode `ReadBase64` zum direkten Einlesen Base64-kodierter Daten an, jedoch erwartet die Methode die Angabe der Größe des Ziel-Arrays. Meist ist die Größe der Binärdaten wie in diesem Fall aber nicht bekannt, es sei denn, Sie haben diese Information zusätzlich in der XML-Datei gespeichert. Dann können Sie alternativ durch wiederholtes Aufrufen von `ReadBase64` die Daten häppchenweise in einer Schleife einlesen. Der Aufwand ist jedoch wesentlich größer als der Umweg über die Klasse `Convert`. Daher wurde dem Aufruf von `Convert.FromBase64String` im Beispiel der Vorzug gegeben.

## 265 XML-Dateien lesen mit XmlDocument

Wesentlich komfortabler als mit dem `XmlTextReader` lassen sich XML-Dateien mit einer Instanz der Klasse `XmlDocument` bearbeiten. `XmlDocument` bietet den strukturierten und wahlfreien Zugriff über das Document Object Model, kurz DOM, siehe auch Erläuterungen im Anhang. Durch Aufruf der Methode `Load` kann eine XML-Datei in einem einzigen Schritt geladen werden:

```
Private PresidentDoc As XmlDocument
PresidentDoc = New XmlDocument
PresidentDoc.Load("../presidents.xml")
```

Die `Load`-Methode verfügt über mehrere Überladungen, so dass Sie wahlweise aus einer Datei, einem Stream, einem `TextReader` oder aber auch einem `XmlReader`, wie z. B. dem oben beschriebenen `XmlTextReader`, lesen können. Nach dem Laden stehen die Daten in einem Cache zur Verfügung. Zur Bearbeitung kann innerhalb der baumförmigen Datenstruktur beliebig navigiert werden. Ein Beispiel hierfür ist die in Listing 458 gezeigte Methode `ProcessXmlElement`, die ein beliebiges XML-Element rekursiv analysiert und die Inhalte im Debug-Fenster ausgibt.

```
Public Sub ProcessXmlElement(ByVal element As XmlNode)

    ' Ausgabe des Elementnamens
    Debug.WriteLine("Element: " & element.Name)
    Debug.Indent()

    ' Attribute ausgeben
    For Each att As XmlAttribute In element.Attributes
        Debug.WriteLine("Attribut: " & att.Name & _
```

Listing 458: Rekursive Navigation durch den Unterbaum eines Elementes

---

```

    ", Value: " & att.Value)
Next

' Untergeordnete Knoten durchlaufen
For Each nd As XmlNode In element.ChildNodes
    ' Untergeordnete Elemente durchlaufen
    If TypeOf nd Is XmlElement Then _
        ProcessXmlElement(nd)

    ' Text-Knoten ausgeben
    If nd.NodeType = XmlNodeType.Text Then _
        Debug.WriteLine("Value: " & nd.Value)
Next
Debug.Unindent()

End Sub

```

---

*Listing 458: Rekursive Navigation durch den Unterbaum eines Elementes (Forts.)*

---



---

```

<?xml version="1.0" encoding="Windows-1252"?>
<Presidents>
  <President ID="25">
    <Name>William McKinley</Name>
    <Birthday>29.1.1843</Birthday>
    <Birthplace> Niles, Ohio</Birthplace>
    <YearsServed> 1897-1901</YearsServed>
    <Terms> Republican</Terms>
    <SpecialText>aa <br>1</br>bb<bx/></SpecialText>
  </President>
  <President ID="26">
    <Name>Theodore Roosevelt</Name>
    <Birthday>27.10.1858</Birthday>
    <Birthplace> New York, New York</Birthplace>
    <YearsServed> 1901-1909</YearsServed>
    <Terms> Republican</Terms>
  </President>
  ...
</Presidents>

```

---

*Listing 459: Ausschnitt aus der XML-Datei Presidents.xml*

---

Um die gesamte XML-Struktur zu durchlaufen, wird die Methode `ProcessXmlElement` mit der Referenz des Root-Knotens aufgerufen:

---

```
ProcessXmlElement(PresidentDoc.DocumentElement)
```

---

Dieser Aufruf erzeugt etwa folgende Ausgabe:

---

```

Element: Presidents
  Element: President
    Attribut: ID, Value: 25

```

---

```

Element: Name
    Value: William McKinley
Element: Birthday
    Value: 29.1.1843
Element: Birthplace
    Value: Niles, Ohio
Element: YearsServed
    Value: 1897-1901
Element: Party
    Value: Republican
Element: Terms
    Value: 1
Element: President
    Attribut: ID, Value: 26
...

```

Die Debug-Ausgaben werden hier nur zur Demonstration eingesetzt. Statt der Ausgaben können Sie natürlich beliebige andere Aktionen ausführen.

`ProcessXmlElement` ist allgemein formuliert und arbeitet mit beliebigen Inhalten. Als Aufrufparameter können Sie die Referenz eines beliebigen Elementes übergeben. Das muss nicht unbedingt das Stamm-Element des Baumes sein. Der folgende Aufruf

```

ProcessXmlElement(DirectCast( _
    PresidentDoc.DocumentElement.ChildNodes(11), XmlElement))

```

liefert dieses Ergebnis:

```

Element: President
    Attribut: ID, Value: 36
Element: Name
    Value: Lyndon B. Johnson
Element: Birthday
    Value: 27.08.1908
Element: Birthplace
    Value: Stonewall, Texas
Element: YearsServed
    Value: 1963-1969
Element: Party
    Value: Democratic
Element: Terms
    Value: 1-took over For Kennedy

```

Den ersten bzw. letzten Knoten einer Knotenliste finden Sie mit Hilfe der Eigenschaften `FirstChild` bzw. `LastChild`. Möchten Sie die Geschwisterknoten eines vorgegebenen Knotens durchlaufen, benutzen Sie die Eigenschaften `NextSibling` oder `PreviousSibling`. In einer `Do While`-Schleife können Sie dann alle Geschwisterknoten vorwärts oder rückwärts (Listing 460) durchlaufen und bearbeiten. Die Ausgabe des Beispielcodes sieht dann etwa so aus:

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Propor-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

---

```

43 George W. Bush
42 William Jefferson Clinton
41 George Bush
40 Ronald Reagan
39 Jimmy Carter

```

---

Sie wählen einen Unterknoten mit seinem Namen aus, indem Sie den Namen als Index übergeben. In Listing 450 wird beispielsweise der Knoten `Name` über `nd("Name")` abgerufen.

---

```

' Mit dem letzten Präsidenten-Element beginnen
Dim nd As XmlNode = PresidentDoc.DocumentElement.LastChild

' Solange Knoten gefunden werden
Do While Not nd Is Nothing
    ' Attribut ID ausgeben
    Debug.Write(nd.Attributes("ID").Value)
    ' Namen ausgeben
    Dim nel As XmlElement = nd("Name")
    Debug.WriteLine(" " & nel.FirstChild.Value)

    ' weiter mit dem Vorgänger
    nd = nd.PreviousSibling
Loop

```

---

Listing 460: Geschwisterknoten rückwärts durchlaufen

## 266 Hinzufügen, Entfernen und Ändern von Knoten mit XmlDocument

Die Klasse `XmlElement` besitzt keinen öffentlichen Konstruktor, kann also nicht mit `New` instanziiert werden. Stattdessen werden neue Elemente über die Methode `CreateElement` der Klasse `XmlDocument` erzeugt. Textknoten lassen sich mit `CreateTextNode` anlegen. Listing 461 zeigt ein Unterprogramm, das eine neue `XmlElement`-Instanz bereitstellt und dieser einen neuen Textknoten zuordnet.

---

```

Public Function CreateElement(ByVal doc As XmlDocument, _
    ByVal nodename As String, ByVal value As String) As XmlElement

    ' Neues Element anlegen
    Dim el As XmlElement = doc.CreateElement(nodename)

    ' Textknoten anhängen
    el.AppendChild(doc.CreateTextNode(value))

    ' Referenz des neuen Elementes zurückgeben
    Return el

End Function

```

---

Listing 461: Anlegen eines neuen Elementes mit einem untergeordneten Textknoten



Mittels `AppendChild` und `PrependChild` werden Knoten an eine Knotenliste am Ende bzw. am Anfang eingefügt. Alternativ stehen zum Einfügen die Methoden `InsertAfter` und `InsertBefore` zur Verfügung, um auch in der Listenmitte Elemente hinzufügen zu können. In Listing 462 wird demonstriert, wie die Präsidentenliste aus dem vorherigen Beispiel erweitert werden kann. Zunächst wird ein neues Element angelegt und mit `SetAttribute` das Attribut `ID` angelegt. Danach werden durch wiederholten Aufruf die Kindknoten `Name`, `Birthday` usw. angelegt und mit `AppendChild` dem Präsidenten-Knoten zugeordnet. Zuletzt wird mit `PrependChild` dieser Knoten an den Anfang der Präsidentenliste gestellt.

Die Methode `Save` kann die XML-Struktur wahlweise in einer Datei, einem String, einem Stream oder in einem `XmlWriter` (z.B. `XmlTextWriter`) speichern. Im Beispiel erfolgt die Speicherung in einer neuen Datei.

```
' Neuen Präsidentenknoten anlegen
Dim newPres As XmlElement = PresidentDoc.CreateElement("President")

' Attribut ID setzen
newPres.SetAttribute("ID", "24")

' Element Name
newPres.AppendChild(CreateElement(PresidentDoc, "Name", _
    "Grover Cleveland"))

' Element Birthday
newPres.AppendChild(CreateElement(PresidentDoc, "Birthday", _
    "18.3.1837"))

' Element Birthplace
newPres.AppendChild(CreateElement(PresidentDoc, _
    "Birthplace", "Caldwell, New Jersey"))

' Element YearsServed
newPres.AppendChild(CreateElement(PresidentDoc, "YearsServed", _
    "1885-1889, 1893-1897"))

' Element Party
newPres.AppendChild(CreateElement(PresidentDoc, "Party", _
    "Democratic"))

' Element Terms
newPres.AppendChild(CreateElement(PresidentDoc, "Terms", "2"))

' Neuen Präsidentenknoten als ersten in die Liste aufnehmen
PresidentDoc.DocumentElement.PrependChild(newPres)

' XML-Datei speichern
PresidentDoc.Save("Presidents2.xml")
```

Listing 462: Hinzufügen eines neuen Präsidenten-Elementes am Anfang der Liste

---

Basics

---

Datum/  
Zeit

---

Anwen-  
dungen

---

Zeich-  
nen

---

Bild-  
bearbei-  
tung

---

Win-  
dows  
Forms

---

Controls

---

Propert-  
yGrid

---

Datei-  
system

---

Netz-  
werk

---

System

---

Daten-  
banken

---

**XML**

---

Wissen-  
schaft

---

Ver-  
schiede-  
nes

Änderungen am Inhalt eines Textknotens können Sie über die Eigenschaft Value vornehmen:

---

```
' Knoten suchen (siehe XPath)
Dim fdr As XmlNode
fdr = PresidentDoc.SelectSingleNode("//President[@ID=32]")

' Text des untergeordneten Textknotens ändern
fdr("Name").FirstChild.Value = "Franklin Delano Roosevelt"
```

---

Alternativ kann der untergeordnete Knoten auch vollständig ersetzt werden:

---

```
' Das Element "Name"
Dim fdrn As XmlNode = fdr("Name")
' Der neue Knoten
Dim newName As XmlNode = PresidentDoc.CreateTextNode( _
    "Franklin Delano Roosevelt")

' Alten Knoten durch neuen ersetzen
fdrn.ReplaceChild(newName, fdrn.FirstChild)
```

---

Diese Variante ist vor allem dann sinnvoll, wenn ein Knoten durch einen neuen mit einer anderen Struktur ersetzt werden soll.

Löschen können Sie einen Knoten, über dessen Referenz Sie verfügen, durch den Aufruf der Methode RemoveChild:

---

```
PresidentDoc.DocumentElement.RemoveChild(fdr)
```

---

Hier wird der Knoten, auf den fdr verweist, gelöscht. Voraussetzung bei diesem Aufruf ist, dass der Knoten ein Unterknoten des Stamm-Elementes ist.

Durch den Aufruf von RemoveAll können Sie alle untergeordneten Knoten gleichzeitig entfernen.

## 267 XmlDocument mit XPath-Ausdrücken durchsuchen

Natürlich können Sie mit Hilfe der vielen Navigations-Methoden, die die Klassen XmlNode, XmlDocument und XmlElement zur Verfügung stellen, eine XmlDocument-Instanz beliebig durchlaufen und die Knoten auf Erfüllung selbst definierter Kriterien überprüfen. Einfacher geht es jedoch, wenn Sie eine spezielle Abfrage-Syntax verwenden, z. B. XPath.

Unter dem Titel »XML Path Language«, kurz XPath, definiert das W3C eine allgemeingültige Abfragesprache für XML-Strukturen. Die Definition finden Sie unter

<http://www.w3.org/TR/xpath>.

Auch das .NET Framework unterstützt XPath-Abfragen. Microsoft hält sich bei der Implementierung weitestgehend an die W3C-Empfehlung XPath 1.0. An dieser Stelle soll und kann hier nicht die komplexe XPath-Syntax erläutert werden. Hierzu gibt es neben der Definition zahlreiche Tutorials, z. B. von W3School unter <http://www.w3schools.com/xpath/default.asp>. Stattdessen wollen wir anhand einiger Beispiele zeigen, wie Sie XPath-Ausdrücke nutzen können, um ein XmlDocument-Objekt zu durchsuchen.

Die Klasse `XmlNode`, von der `XmlDocument` und in zweiter Generation auch `XmlElement` abgeleitet sind, definiert zwei Methoden für Abfragen mit XPath-Ausdrücken:

- ▶ `SelectSingleNode`
- ▶ `SelectNodes`

Beiden Methoden gemein ist, dass der XPath-Ausdruck eine Knotenliste zum Ergebnis haben muss. `SelectSingleNode` gibt, wie der Name vermuten lässt, genau einen Knoten zurück. Enthält die Knotenliste einen oder mehrere Knoten, wird stets der erste Knoten zurückgegeben. Ist die Liste leer, dann ist das Ergebnis `Nothing`.

`SelectedNodes` liefert hingegen immer ein Objekt vom Typ `XmlNode`, auch, wenn die Ergebnisliste leer ist. `XmlNodeList` ist eine Auflistungsklasse für `XmlNode`-Referenzen. Sie implementiert `IEnumerable` und kann somit in einer `For Each`-Schleife durchlaufen werden.

### Anwendung zum Testen von XPath-Ausdrücken

Erfahrungsgemäß ist der Umgang mit XPath-Ausdrücken innerhalb eines Programms nicht besonders einfach. Knotenlisten, die als Ergebnis zurückgegeben werden, lassen sich aufgrund ihrer teils stark verschachtelten Baumstruktur nur mühsam mit dem Debugger untersuchen und analysieren. Insbesondere für diejenigen, die gerne sehen möchten, welches Ergebnis ihre XPath-Abfrage liefert, ist das nachfolgend beschriebene Programm gedacht.

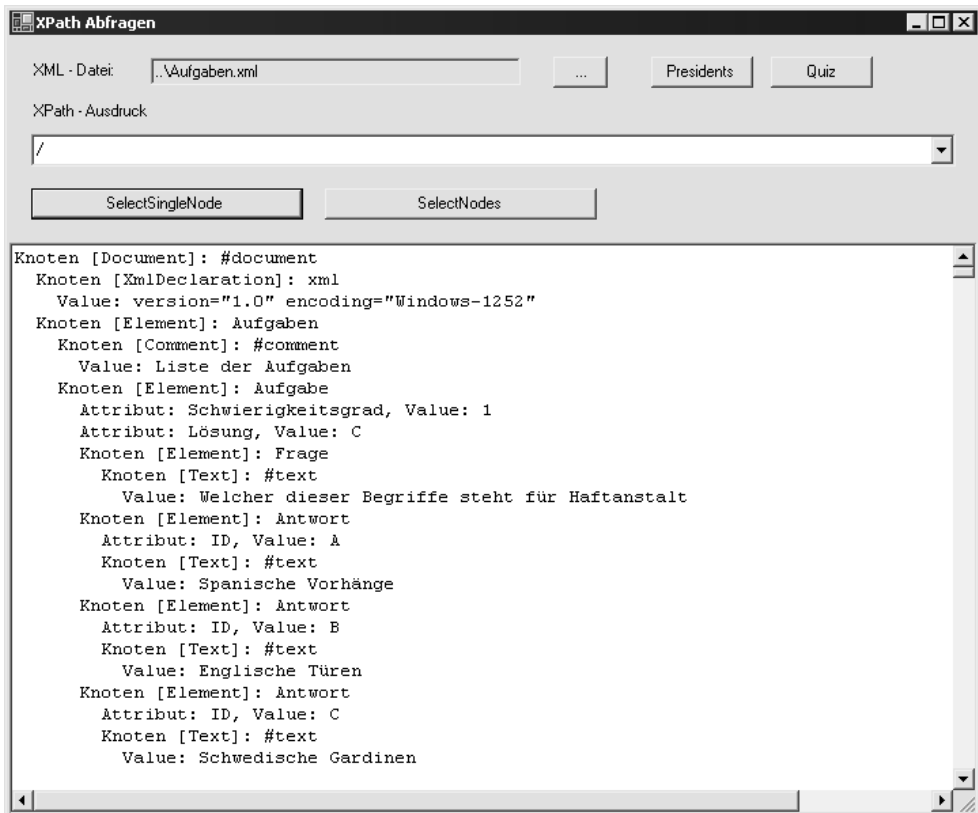


Abbildung 234: Analyse der Datei Aufgaben.xml

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Propor-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

Das Programm ermöglicht die Auswahl einer beliebigen XML-Datei oder einer der beiden Beispieldateien *Presidents.xml* oder *Aufgaben.xml*. Für die geladene Datei können in einer ComboBox vorbereitete Abfragen ausgewählt oder eigene eingegeben werden. Zwei Schaltflächen, `SELECT-SINGLENODE` und `SELECTNODES`, rufen die betreffenden Methoden auf und zeigen das Ergebnis in einer TextBox an. Abbildung 234 zeigt ein Beispiel für die Datei *Aufgaben.xml*.

Um die tatsächliche Knotenstruktur genauer untersuchen zu können, wird eine Variante der oben beschriebenen Methode `ProcessXmlElement` vorgesehen. Die neue Methode, `ProcessXmlNode`, siehe Listing 463, nimmt einen beliebigen Knoten als Parameter entgegen und gibt als Erstes dessen Typ und Namen aus. Anschließend werden alle Attribute ausgegeben, sofern der Knoten über Attribute verfügt. Existiert die Eigenschaft `Value` (z.B. bei Text- und Kommentarknoten), dann wird der zugehörige Wert ausgegeben. Letztlich erfolgt der rekursive Aufruf der Methode für alle existierenden Kindknoten.

---

```
Public Sub ProcessXmlNode(ByVal node As XmlNode)

    ' Ausgabe von Knotentyp und Knotenname
    Trace.WriteLine("Knoten [" & node.NodeType.ToString() & "]: " & _
        & node.Name)

    ' Nachfolgende Ausgaben einrücken
    Trace.Indent()

    ' Attribute ausgeben
    If Not node.Attributes Is Nothing Then
        For Each att As XmlAttribute In node.Attributes
            Trace.WriteLine("Attribut: " & att.Name & _
                ", Value: " & att.Value)
        Next
    End If

    ' Value ausgeben, sofern vorhanden
    If Not node.Value Is Nothing Then Trace.WriteLine _
        ("Value: " & node.Value)

    ' Untergeordnete Knoten durchlaufen
    For Each nd As XmlNode In node.ChildNodes
        ' Untergeordnete Elemente durchlaufen
        ProcessXmlNode(nd)
    Next

    ' Einrückungsstufe dekrementieren
    Trace.Unindent()

End Sub
```

---

Listing 463: `ProcessXmlNode` gibt ausführlich Auskunft über die Struktur des übergebenen Knotens

Anders als im vorherigen Beispiel werden hier auch Stamm-, Deklarations- und Kommentarknoten berücksichtigt und Textknoten explizit als Unterknoten kenntlich gemacht. So können Sie leichter ermitteln, wie Sie nach einer XPath-Abfrage gezielt auf die benötigten Informationen zugreifen können.

Die Ausgabe erfolgt wie zuvor mit Hilfe verschiedener statischer Methoden der Klasse `Trace`. Über ein Listener-Objekt (siehe Rezept 14.17 ((Debug- und Trace-Ausgaben an eine TextBox weiterleiten))) vom Typ `TraceToTextBox` wird die Ausgabe zusätzlich in einer TextBox sichtbar gemacht.

In den Click-Ereignis-Handlern der beiden `Select`-Schaltflächen (Listing 464) werden die jeweiligen `Select`-Methoden aufgerufen und das Ergebnis durch (wiederholtes) Aufrufen von `ProcessXmlNode` angezeigt. Im Falle von `SelectNodes` werden in der Ausgabe die einzelnen Knoten durch eine Reihe von Sternchen voneinander getrennt. Abbildung 235 zeigt eine Abfrage, bei der die Aufgabenliste nach Aufgaben-Elementen durchsucht wird, deren Attribut `Schwierigkeitsgrad` den Wert 15 hat. Als Ergebnis wird für jeden Treffer das Element `Fragen` zurückgegeben.

```
Private Sub BTNSelectSingleNode_Click(ByVal sender As _
    System.Object, ByVal e As System.EventArgs) Handles _
    BTNSelectSingleNode.Click

    ' Fenster löschen
    TBTrace.Clear()

Try
    ' Abfrage mit SelectSingleNode
    Dim xn As XmlNode = xdoc.SelectSingleNode(CBOXPath.Text)

    ' Ausgabe des Ergebnisknotens
    ProcessXmlNode(xn)

Catch ex As Exception
    ' Laufzeitfehler z.B., wenn Ergebnisknoten leer ist oder
    ' der XPath-Ausdruck fehlerhaft ist
    MessageBox.Show("Fehler: " & ex.Message)
End Try

End Sub

Private Sub BTNSelectNodes_Click(ByVal sender As _
    System.Object, ByVal e As System.EventArgs) Handles _
    BTNSelectNodes.Click

    ' Fenster löschen
    TBTrace.Clear()

Try
    ' Abfrage mit SelectNodes
    Dim xn1 As XmlNodeList = xdoc.SelectNodes(CBOXPath.Text)

    ' Ausgabe aller Knoten der Ergebnisliste
    For Each xn As XmlNode In xn1
        Debug.WriteLine("*****")
        ProcessXmlNode(xn)
    Next
```

Listing 464: XML-Struktur mit eingegebenem XPath-Ausdruck durchsuchen und Ergebnisse anzeigen

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tung  
Win-  
dows  
Forms

Controls

Proper-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken**XML**Wissen-  
schaftVer-  
schiede-  
nes

```

Catch ex As Exception
    ' Laufzeitfehler, wenn der XPath-Ausdruck fehlerhaft ist
    MessageBox.Show("Fehler: " & ex.Message)
End Try

```

```
End Sub
```

Listing 464: XML-Struktur mit eingegebenem XPath-Ausdruck durchsuchen und Ergebnisse anzeigen (Forts.)

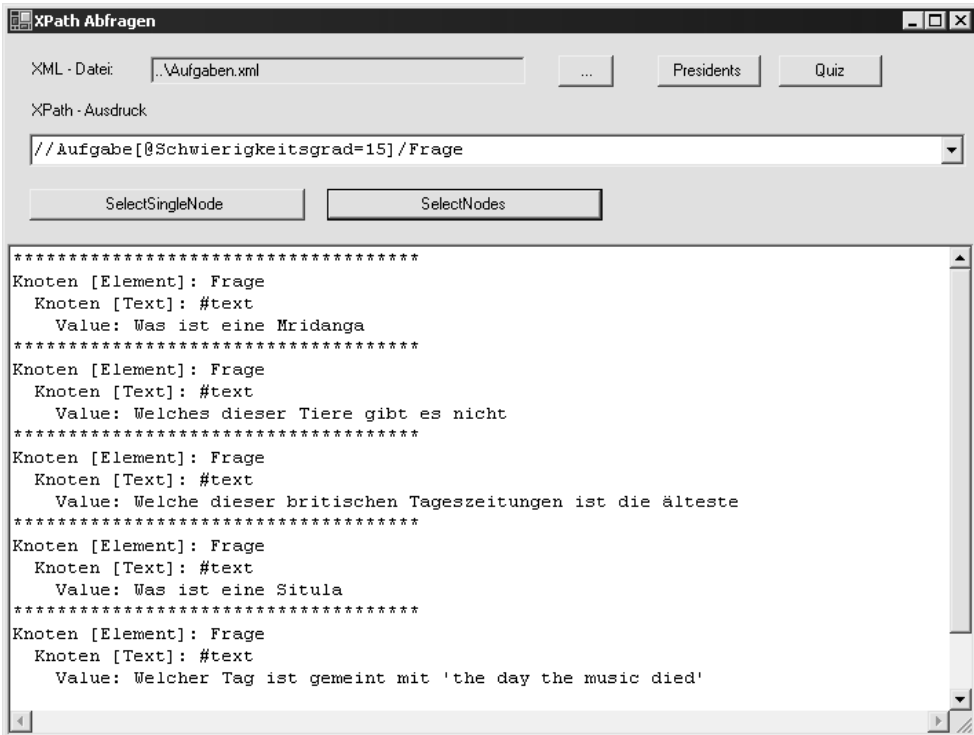


Abbildung 235: Die Ergebnisliste enthält fünf Knoten

Auch die im vorherigen Beispiel benutzte Abfrage

```
//President[@ID=32]
```

können Sie hier austesten. Sie liefert den Elementknoten vom Typ President zurück, dessen Attribut ID den Wert 32 hat. Die Abfrage

```
//President[contains(Birthplace,"New York")]
```

generiert eine Liste aller Präsidenten, bei denen der Geburtsort (Element Birthplace) die Zeichenkette „New York“ enthält (siehe Abbildung 236).

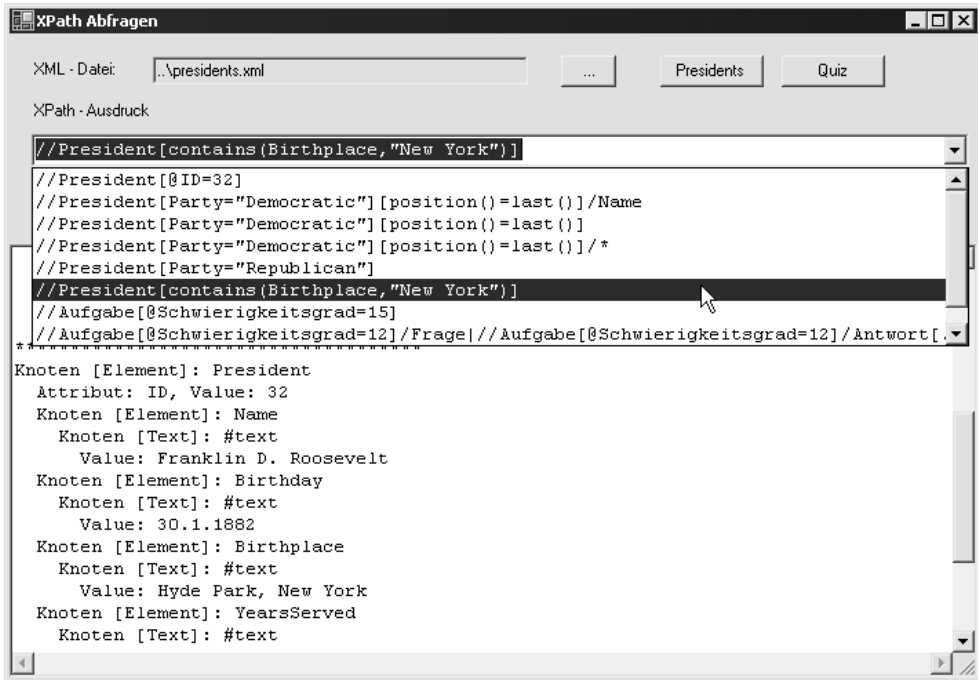


Abbildung 236: Sie können vorbereitete Abfragen auswählen oder neue eingeben

Hinweis

Die XPath-Abfragen in diesem Beispiel funktionieren nur, wenn die XML-Datei keinen Namespace definiert. Für XPath-Ausdrücke mit Namespace-Angaben siehe Rezept [13.8](#) ((XPath-Abfragen und XML-Namespaces)).

## 268 XPath-Abfragen und XML-Namespaces

Da die Namen von Elementen innerhalb einer XML-Struktur nicht unbedingt eindeutig sind, können sie Namensräumen zugeordnet werden. Insbesondere dann, wenn Daten aus verschiedenen Quellen zusammengefügt werden, sind Namensräume unerlässlich, um Mehrdeutigkeiten zu vermeiden. Beispielsweise wurde die Datei Quiz.xml (Listing 465) zusammengesetzt aus Kandidaten-Informationen und der Datei Aufgaben.xml. Das Element `Aufgabe` existiert in verschiedenen Kontexten. Zum einen innerhalb eines Elementes vom Typ `Person`, zum anderen innerhalb des Elementes `Aufgaben`.

Zur Unterscheidung werden zwei Namensräume definiert:

- Addison-Wesley/VBCodeBook/XML/Aufgaben
- Addison-Wesley/VBCodeBook/XML/Kandidat

Während der erste der Standard-Namensraum ist, wird der zweite explizit zugewiesen. Um die Texte innerhalb der XML-Struktur kurz zu halten, werden statt der Namensräume Platzhalter, auch Prefix genannt, verwendet. Für den zweiten Namensraum wurde hier der Prefix `Kandidat` definiert und eingesetzt.

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

```

<?xml version="1.0" encoding="utf-8" ?>
<Quiz xmlns="Addison-Wesley/VBCodeBook/XML/Aufgaben" xmlns:Kandidat="Addison-Wesley/
VBCodeBook/XML/Kandidat">
  <Kandidat:Person Name="Peter Müller">
    <Kandidat:Aufgabe Schwierigkeitsgrad="10">
      <Kandidat:Status>Korrekte Antwort gegeben</Kandidat:Status>
    </Kandidat:Aufgabe>
  </Kandidat:Person>
</Aufgaben>
<!--Liste der Aufgaben-->
<Aufgabe Schwierigkeitsgrad="1" Lösung="C">
  <Frage>Welcher dieser Begriffe steht für Haftanstalt</Frage>
  <Antwort ID="A">Spanische Vorhänge</Antwort>
  <Antwort ID="B">Englische Türen</Antwort>
  <Antwort ID="C">Schwedische Gardinen</Antwort>
  <Antwort ID="D">Sächsische Rollos</Antwort>
</Aufgabe>

...

<Aufgabe Schwierigkeitsgrad="12" Lösung="C">
  <Frage>Wo ist das nördlichste Vorkommen von Pinguinen</Frage>
  <Antwort ID="A">südl. Polarkreis</Antwort>
  <Antwort ID="B">südl. Wendekreis</Antwort>
  <Antwort ID="C">Äquator</Antwort>
  <Antwort ID="D">nördl. Polarkreis</Antwort>
  <Erläuterung>Galapagos Inseln</Erläuterung>
</Aufgabe>
</Aufgaben>
</Quiz>

```

---

*Listing 465: XML-Datei Quiz.xml definiert und benutzt Namensräume*

Beim Laden der XML-Datei mit `XmlDocument.Load` werden alle Prefixe aufgelöst und durch die zugeordneten Namensräume ersetzt. Der Namensraum jedes Elementes kann über die Eigenschaft `NamespaceURI` abgefragt werden. Die ursprünglichen Prefixe stehen jedoch für Abfragen nicht mehr zur Verfügung.

Sollten Sie schon einmal versucht haben, mit XPath-Abfragen in XML-Dateien zu suchen, in denen ein oder mehrere Namensräume definiert sind, ohne im XPath-Ausdruck den Namensraum anzugeben, dann wissen Sie, dass Sie als Ergebnis eine leere Knotenliste zurückerhalten. Beispielsweise der Ausdruck

---

```
//Aufgabe
```

---

gab für die XML-Datei *Aufgaben.xml* die Liste aller Elemente vom Typ *Aufgabe* zurück. Für die Datei *Quiz.xml* ist die Rückgabeliste jedoch leer. Und das, obwohl die Elemente *Aufgabe* innerhalb der Aufgabenliste zum Standard-Namensraum gehören.

Die Ursache hierfür ist in der Konformität zur XPath 1.0 Definition zu suchen. Diese Version kennt keinen Standard-Namensraum. Wird ein Element einem Namensraum zugeordnet, dann muss dieser bei der Suche mittels XPath auch angegeben werden, auch wenn es der Standard-Namensraum



ist. Nur, wenn der Standard-Namensraum nicht existiert (wie in den Beispielen im vorangegangenen Rezept), dann funktionieren die XPath-Abfragen ohne Angabe von Namensräumen.

Mit der Xpath-Version 2.0 soll das Problem behoben werden. Ob, wann und wie Microsoft diese Version implementieren wird, ist zurzeit noch unklar.

Damit die XPath-Ausdrücke nicht zu lang werden, definiert man in der Regel auch hier wieder Prefixe für die einzelnen Namensräume. Diese müssen mit denen in der XML-Datei nichts mehr zu tun haben und können frei festgelegt werden. Zur Definition sowie zur Auflösung bei XPath-Abfragen wird ein Objekt vom Typ `XmlNamespaceManager` benötigt.

In Listing 466 sehen Sie eine typische Konstruktion, um bei XPath-Abfragen Namensräume zu berücksichtigen. Nach Anlegen einer Instanz vom Typ `XmlNamespaceManager` werden die benötigten Namensräume unter Angabe der gewünschten Prefixe (hier A und B) hinzugefügt. Beim Aufruf von `SelectSingleNode` oder `SelectNodes` wird die Referenz des Namespace-Managers als zusätzlicher Parameter übergeben.

---

```
' Namespace-Manager anlegen
Dim nsmgr As New XmlNamespaceManager(xdoc.NameTable)
' Namensräume hinzufügen
nsmgr.AddNamespace("A", "Addison-Wesley/VBCodeBook/XML/Aufgaben")
nsmgr.AddNamespace("B", "Addison-Wesley/VBCodeBook/XML/Kandidat")
' XPath-Abfrage mit Prefix
Dim node As XmlNode = xdoc.SelectSingleNode("//B:Aufgabe", nsmgr)
ProcessXmlNode(node)
```

---

Listing 466: XPath-Abfrage mit Namespace-Definitionen

Der Code aus Listing 466 erzeugt die folgende Ausgabe:

---

```
Knoten [Element]: Kandidat:Aufgabe
  Attribut: Schwierigkeitsgrad, Value: 10
  Knoten [Element]: Kandidat:Status
    Knoten [Text]: #text
      Value: Korrekte Antwort gegeben
```

---

Um das XPath-Testprogramm auch für XML-Dateien mit Namensräumen einsetzen zu können, sind ein paar Erweiterungen notwendig. Das Laden der Dateien geschieht über die zentrale Methode `LoadXmlDocument` (Listing 467). Hier wird ein `XmlNamespaceManager`-Objekt angelegt und initialisiert. Die Referenz dieses Objektes wird als Membervariable der Fensterklasse allen anderen Methoden zur Verfügung gestellt und wird bei den Aufrufen von `SelectNodes` und `SelectSingleNode` übergeben.

Bei der Ermittlung der Namensräume wird davon ausgegangen, dass alle verwendeten Namensräume als Attribute des Stammknotens definiert werden. Diese Annahme ist nicht allgemeingültig, trifft aber auf die meisten Fälle zu. Namensräume, die an anderer Stelle definiert werden, werden vom Testprogramm nicht berücksichtigt.

---

```
Private Sub LoadXmlDocument(ByVal path As String)
' XML-Datei laden
xdoc.Load(path)
```

---

Listing 467: Laden einer XML-Datei und Initialisieren des Namespace-Managers

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

```

' Pfad anzeigen
Dim relpath As New Uri(Application.StartupPath)
TBPath.Text = relpath.MakeRelative(New Uri(path))
TBTrace.Text = "XML-Datei geladen: " & path

' Namespaceliste löschen
LVNamespaces.Items.Clear()

' Namespacemanager anlegen
nsMgr = New XmlNamespaceManager(xdoc.NameTable)

Dim i As Integer = 0

' Attribute nach Namensraumdefinitionen durchsuchen
For Each att As XmlAttribute In xdoc.DocumentElement.Attributes
    ' Ist das Attribut eine Namespace-Deklaration?
    If att.Name.StartsWith("xmlns") Then
        ' Namespace hinzufügen, Prefix Ni (i=1fd. Nummer)
        nsMgr.AddNamespace("N" & i.ToString(), att.Value)
        i += 1
    End If
Next

' ListView mit Namensräumen und Prefixen füllen
For Each nspc As String In nsMgr
    Dim lvi As ListViewItem = _
        LVNamespaces.Items.Add(nspc)
    lvi.SubItems.Add(nsMgr.LookupNamespace(nspc))
Next

End Sub

```

---

*Listing 467: Laden einer XML-Datei und Initialisieren des Namespace-Managers (Forts.)*

Alle Namensraumdefinitionen, die für den Stammknoten gefunden werden, werden dem Namespace-Manager bekannt gemacht. Jedem Namensraum wird ein Prefix, bestehend aus dem Buchstaben N und einer laufenden Nummer, zugeordnet. In einem ListView-Steuerelement werden abschließend alle Namensräume aufgeführt (siehe Abbildung 237).

Über das Kontrollkästchen NAMESPACE ANZEIGEN kann die Ausgabe des Namensraums für jeden Knoten ein- bzw. ausgeschaltet werden. Die Information wird bei Bedarf über die Eigenschaft `NamespaceURI` des jeweiligen Knotens ermittelt.

## 269 Schnellere Suche mit XPathDocument und XPathNavigator

Wollen Sie eine XML-Struktur nicht verändern, sondern nur durchlaufen oder durchsuchen, dann steht Ihnen mit der Klassenkombination `XPathDocument` und `XPathNavigator` eine wesentlich schnellere Variante zur Verfügung als mit der universelleren Klasse `XmlDocument`. `XPathDocument` und `XPathNavigator` sind für schnelle Lesezugriffe optimiert und bieten eine Vielzahl an Navigationsmöglichkeiten.

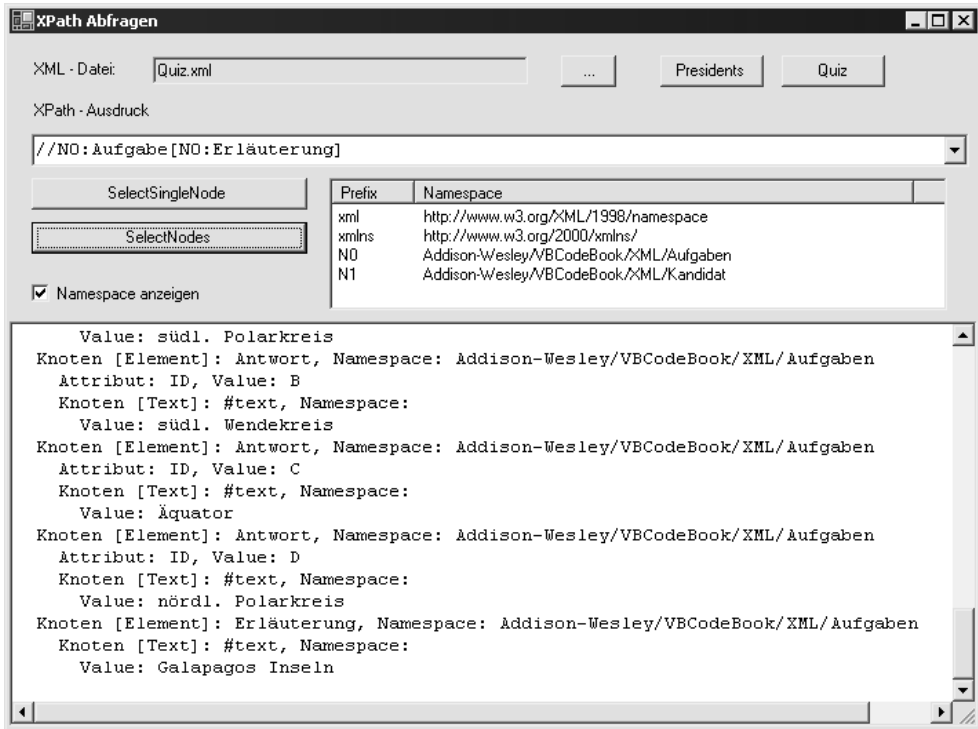


Abbildung 237: XPath-Suche mit Namespace-Definitionen

Bei der Instanziierung von `XPathDocument`-Objekten muss bereits die Datenquelle angegeben werden. Verschiedene Konstruktor-Überladungen ermöglichen das Laden aus Dateien, Streams und `TextReader`-Instanzen. Die Klasse `XPathNavigator` kennt als einzige öffentliche, nicht von `Object` geerbte Methode `CreateNavigator`. Diese Methode legt eine neue `XPathNavigator`-Instanz an und gibt deren Referenz zurück. Eine typische Aufruffolge zum Anlegen eines `XPathNavigator`-Objektes sieht so aus:

```
Private xpDoc As XPathDocument
Private xpNav As XPathNavigator
...
xpDoc = New XPathDocument("..\Presidents.xml")
xpNav = xpDoc.CreateNavigator()
```

Auch mehrere unabhängige Navigator-Objekte können Sie mit `CreateNavigator` erzeugen. Ein Navigator-Objekt bietet eine Vielzahl an Möglichkeiten, Knoten zu selektieren. Mit 14 verschiedenen teilweise überladenen `MoveTo`-Methoden stehen Ihnen zahlreiche Varianten für die Navigation von Knoten zu Knoten zur Verfügung. Mehrere Eigenschaften wie `Value` und `Name` geben Auskunft über den aktuellen Knoten.

Wesentlich interessanter ist allerdings die Suche mit XPath-Ausdrücken, wie Sie sie bereits von der `XmlDocument`-Klasse kennen. `XPathNavigator` stellt hierzu die Methode `Select` mit zwei Überladungen bereit. In der ersten Variante übergeben Sie den Ausdruck als String (Listing 468).

---

```

Dim xni As XPathNodeIterator
' XPath-Suche gibt NodeIterator zurück
xni = xpNet.Select("//President[@ID>35]/Name")

' Liste der gefundenen Knoten durchlaufen
Do While xni.MoveNext()
    Debug.WriteLine(xni.Current.Value)
Loop

```

---

*Listing 468: Suche mit XPathNavigator.Select und einem als String übergebenen XPath-Ausdruck*

Abfragen beziehen sich immer auf den aktuellen Knoten, auf den der Navigator verweist. Der Rückgabewert der Select-Methode ist vom Typ XPathNodeIterator und erlaubt, die Knotenliste in einer While-Schleife zu durchlaufen. MoveNext positioniert auf das nächste (beim ersten Aufruf auf das erste) Element und liefert einen booleschen Wert, der angibt, ob noch weitere Knoten existieren. Die Eigenschaft Current gibt die Referenz des jeweils aktuellen Knotens an, so dass die Ausgabe aus Listing 468 in etwa so aussieht:

---

```

Lyndon B. Johnson
Richard M. Nixon
Gerald R. Ford
Jimmy Carter
Ronald Reagan
George Bush
William Jefferson Clinton
George W. Bush

```

---

Für komplexe XPath-Ausdrücke lässt der XPathNavigator noch eine schnellere Abfragevariante zu: die Kompilierung des XPath-Ausdrucks (Listing 469).

---

```

' Definition und Kompilierung eines XPath-Ausdrucks
Dim xpExp As XPathExpression
xpExp = xpNet.Compile("//President[@ID>35]/Name")

' Abfrage mit kompiliertem Ausdruck
Dim xni As XPathNodeIterator
xni = xpNet.Select(xpExp)

' Liste der gefundenen Knoten durchlaufen
Do While xni.MoveNext()
    Debug.WriteLine(xni.Current.Value)
Loop

```

---

*Listing 469: Suche mit XPathNavigator.Select und einem kompilierten XPath-Ausdruck*

Das Ergebnis dieser Variante ist identisch mit dem des vorigen Beispiels. Auch die Ausführungsgeschwindigkeit wird sich kaum messbar unterscheiden. Erst bei großen XML-Dateien und komplexen, oft benötigten Ausdrücken kann die Vorkompilierung ihre Vorteile ausspielen.

## XPathNavigator und Namensräume

Die in Rezept 13.8 ((XPath-Abfragen und XML-Namenspaces)) dargestellte Problematik der Namensräume bei XPath-Abfragen in XmlDocument-Instanzen trifft gleichermaßen auf die XPathNavigator-Klasse zu. Auch hier müssen Sie, sobald in der XML-Datei ein Namensraum angegeben wird, eine XmlNamespaceManager-Instanz anlegen, die Namensräume hinzufügen und im XPath-Ausdruck (durch Prefixe) angeben. XPathNavigator.Select lässt allerdings nicht die zusätzliche Angabe eines Namespace-Managers zu. Stattdessen müssen Sie hier den Ausdruck kompilieren und mittels SetContext dem XPathExpression-Objekt die XmlNamespaceManager-Instanz zuordnen (Listing 470).

```
' Dokument laden
Dim xpd As New XPathDocument("../Quiz.xml")

' Navigator anlegen
Dim xpnv As XPathNavigator = xpd.CreateNavigator()

' Namespace-Manager anlegen
Dim nsmgr As New XmlNamespaceManager(xpnv.NameTable)

' Namensräume und Prefixe hinzufügen
nsmgr.AddNamespace("A", "Addison-Wesley/VBCodeBook/XML/Aufgaben")
nsmgr.AddNamespace("K", "Addison-Wesley/VBCodeBook/XML/Kandidat")

' XPath-Ausdruck kompilieren
Dim xpExp As XPathExpression
xpExp = xpnv.Compile("//K:Person/@Name")

' Namespace-Manager zuweisen
xpExp.SetContext(nsmgr)

' Abfrage ausführen
Dim xni As XPathNodeIterator
xni = xpnv.Select(xpExp)

' Liste der gefundenen Knoten durchlaufen
Do While xni.MoveNext()
    Debug.WriteLine(xni.Current.Value)
Loop
```

Listing 470: Vorkompilierte XPath-Abfrage unter Angabe von Namensräumen

## Vorgefertigte Knotenlisten

Für einige typische Knotenlisten benötigen Sie keine XPath-Abfrage, sondern können sie mit Hilfe der jeweiligen Select-Methode direkt abrufen. Die folgenden drei Methoden geben eine Knotenliste als XPathNodeIterator-Referenz zurück:

- ▶ **SelectChildren** generiert die Liste der Kindknoten
- ▶ **SelectAncestors** generiert die Liste der Vorgänger
- ▶ **SelectDescendants** generiert die Liste der Nachfolger

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Proper-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

Jede dieser Methoden erwartet einen Parameter vom Typ `XPathNodeType`, mit dem Sie (als Bitkombination) die Auswahl auf bestimmte Knotentypen einschränken können.

### XPath-Ausdrücke auswerten, die keine Knotenliste zurückgeben

Die `Select`-Methode erlaubt nur XPath-Ausdrücke, die eine Knotenliste zum Ergebnis haben, denn schließlich ist der Rückgabewert vom Typ `XPathNodeListIterator`. XPath-Ausdrücke müssen aber nicht zwangsläufig eine Knotenliste generieren, sondern können auch andere Ergebnisse liefern. In der W3C-Dokumentation sind eine Reihe von XPath-Funktionen definiert, die z.B. für Berechnungen genutzt werden können. Ein Ausdruck wie

---

```
count(//President[contains(Birthplace,'New York')])
```

---

berechnet die Anzahl der Knoten und würde für unsere Beispieldatei *Presidents.xml* den Wert 2 zurückgeben. Für derartige Auswertungen können Sie die Methode `Evaluate` einsetzen:

---

```
' Wie viele Präsidenten wurden in New York geboren?
Dim result As Object = xpNav.Evaluate( _
    "count(//President[contains(Birthplace,'New York')])")
```

---

`Evaluate` gibt eine allgemeine `Object`-Referenz zurück. Sie müssen selbst die notwendigen Typ-Umwandlungen vornehmen. Alle gültigen XPath-Ausdrücke können Sie mit `Evaluate` verwenden. Auch Ausdrücke, die eine Knotenliste zurückgeben, wie beim Aufruf von `Select`:

---

```
result = xpNav.Evaluate("//President/Name")
```

---

Als Ergebnis erhalten Sie hier wieder eine Referenz eines `XPathNodeListIterator`-Objektes.

## 270 XmlView-Steuerelement zur strukturierten Darstellung von XML-Dateien

Die Baumstruktur einer XML-Datei legt es nahe, die enthaltenen Informationen in einem `TreeView`-Steuerelement darzustellen. Attribute eines Knotens können dann beispielsweise in einem verbundenen `ListView`-Control dargestellt werden. Mit Hilfe der oben vorgestellten Navigationsmöglichkeiten können Sie ein solches Vorhaben leicht realisieren.

In diesem Rezept wollen wir den Aufbau eines `UserControls` vorstellen, das zusätzlich zu den genannten Fähigkeiten den Knoten im `TreeView`-Control Icons und `ToolTips` zuordnet. Für die Daten der in Listing 471 abgedruckten XML-Datei soll eine Darstellung wie in Abbildung 238 realisiert werden. Die Zuordnung von Icons und Hilfetexten erfolgt durch eine zweite XML-Datei (Listing 472). Dort wird auch mit Hilfe des Attributs `DefaultAttribute` festgelegt, welches Attribut eines Datenknotens für die Textanzeige in der `TreeView` herangezogen werden soll.

Eine beliebige XML-Datei, die die Daten in Form von Attributen bereithält, kann so in einer ansprechenden Form mit der `TreeView/ListView`-Kombination dargestellt werden. Lediglich die Definitionsdatei und die Icon-Dateien müssen zusätzlich bereitgestellt werden. Das Steuerelement lässt sich in Fenstern und anderen Steuerelementen einsetzen.

```
<?xml version="1.0" encoding="utf-8" ?>
<Firma xmlns="http://tempuri.org/Firma.xsd" Name="Meier GmbH">
  <Abteilung Bezeichnung="Software Entwicklung" ...
    <Abteilungsleiter Geburtsdatum="1.1.1950" Nachname="Meier" ...
      <Mitarbeiter Geburtsdatum="2.2.1965" Nachname="Schmitz" ...
        <Mitarbeiter Geburtsdatum="3.3.1966" Nachname="Hase" ...
          <Computer ID="Entwicklungsserver" Betriebssystem="NT4 Server" />
          <Unterabteilung Bezeichnung="Xml-Design" Kurzbezeichnung=...
            <Abteilungsleiter Nachname="Bär" Vorname="Yogi" Telefon=...
              <Mitarbeiter Geburtsdatum="4.4.1973" Nachname="Bacher" ...
                <Computer ID="XmlTest PC" Betriebssystem="WinXP"/>
              </Unterabteilung>
            </Abteilung>
          <Abteilung Bezeichnung="Vertrieb" Kurzbezeichnung="V">
            <Abteilungsleiter Geburtsdatum="5.5.1962" Nachname= ...
              <Mitarbeiter Nachname="Stenz" Vorname="Julius" Telefon="201" />
              <Computer ID="KundenDB" Betriebssystem="Win2003 Server"/>
            </Abteilung>
          </Firma>
```

Listing 471: FirmaMeier.xml beschreibt die Firmenstruktur mit Abteilungen, Mitarbeitern und Computern

```
<?xml version="1.0" encoding="utf-8" ?>
<Definitions xmlns="http://tempuri.org/XmlTreeViewDef.xsd">
  <EntryDefinition Tagname="Firma" Icon="factory.ico"
    DefaultAttribute="Name"/>
  <EntryDefinition Tagname="Abteilung" Icon="grouph.ico"
    DefaultAttribute="Bezeichnung"/>
  <EntryDefinition Tagname="Unterabteilung" Icon="group.ico"
    DefaultAttribute="Bezeichnung"/>
  <EntryDefinition Tagname="Abteilungsleiter" Icon="userh.ico"
    DefaultAttribute="Nachname"/>
  <EntryDefinition Tagname="Mitarbeiter" Icon="user.ico"
    DefaultAttribute="Nachname"/>
  <EntryDefinition Tagname="Computer" Icon="computer.ico"
    DefaultAttribute="ID"/>
</Definitions>
```

Listing 472: MeierTreeViewDef.xml ordnet den Elementen Icons und Hilfetexte zu

Das UserControl besteht im Wesentlichen aus der TreeView (DataTree), der ListView (DataDetail) und einem Splitter zur dynamischen Größenanpassung. Einige private Member-Variablen halten die für die Navigation und die visuelle Darstellung benötigten Informationen bereit:

```
' Pfad der XML-Datei, die die Daten enthält
Private source As String
' Pfad der XML-Datei, die die Zuordnungen enthält
Private entryDefinition As String
' XmlDocument-Instanz für die Navigation in den Daten
Private doc As XmlDocument = New XmlDocument
```

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Propor-  
tivityGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

```

' XmlDocument für die Navigation in der Definitionsdatei
Private edDoc As XmlDocument
' Namespace-Manager hierzu
Private edDocNsmgr As XmlNamespaceManager
' Stammknoten der Daten in der TreeView
Private treeRoot As TreeNode
' Hashtable als Zuordnungstabelle
Private imageIndex As New Hashtable

```

---

Zwei öffentliche Eigenschaften, `XmlSource` und `XmlEntryDefinition` (Listing 473), definieren die Schnittstelle nach außen. Ihnen wird ein eigener `UITypeEditor` zugeordnet, damit bereits im Eigenschaftsfenster die Dateien über der `FileOpen`-Dialog zugeordnet werden können.

---

```

<Description("Anzuzeigende XML-Datei"), _
    Editor(GetType(XMLFilenameEditor), _
        GetType(System.Drawing.Design.UITypeEditor))> _
Public Property XmlSource() As String
    Get
        Return source
    End Get
    Set(ByVal Value As String)
        source = Value
        Me.Reload()
    End Set
End Property

...
Public Property XmlEntryDefinition() As String
...
End Property

Public Class XMLFilenameEditor
    Inherits System.Windows.Forms.Design.FileNameEditor

    Protected Overrides Sub InitializeDialog(ByVal openFileDialog _
        As System.Windows.Forms.OpenFileDialog)
        ' Filter für XML-Dateien verwenden
        openFileDialog.Filter = "XML-Dateien (*.xml) |*.xml"
    End Sub
End Class

```

---

Listing 473: Eigenschaften `XmlSource` und `XmlEntryDefinition` zur Auswahl der XML-Dateien

Bei Änderung der Eigenschaft `XmlSource` wird die Methode `Reload` aufgerufen (Listing 474). Sie lädt das XML-Dokument und liest die Informationen aus der Definitionsdatei, sofern diese vorhanden ist. Alle Icon-Dateien werden eingelesen und die Bilder der Liste `TreeViewImages` hinzugefügt. `imageIndex` verweist auf ein `Hashtable`-Objekt, über das der Index eines Bildes in der Liste aus dem Elementnamen ermittelt werden kann. Dem `TreeView`-Control wird die Bilderliste zugeordnet, so dass den Knoten nur noch ein Bildindex zugeordnet werden muss, um das zugehörige Symbol anzeigen zu können.



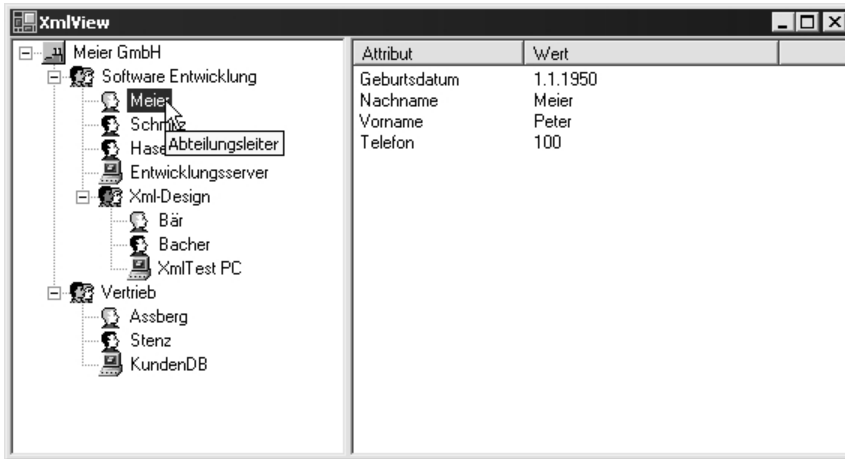


Abbildung 238: Anzeige einer XML-Datei im XmlView-Steuerelement

```

Public Sub Reload()
    Try
        ' Datendokument laden
        doc.Load(source)

        ' Pfad für Icon-Dateien ermitteln
        Dim basePath As String = _
            Directory.GetParent(entryDefinition).FullName & "\"

        ' Wenn die Definitionsdatei existiert
        If entryDefinition.Length > 0 Then
            ' Hashtable leeren
            imageIndex.Clear()
            ' Definitionsdatei laden, Namensraum berücksichtigen
            edDoc = New XmlDocument
            edDoc.Load(entryDefinition)
            edDocNsmgr = New XmlNamespaceManager(edDoc.NameTable)
            edDocNsmgr.AddNamespace("x", _
                "http://tempuri.org/XmlTreeViewDef.xsd")

            ' Bilderliste leeren
            TreeViewImages.Images.Clear()

            ' Definitionsdatei durchlaufen
            For Each element As XmlElement _
                In edDoc.DocumentElement.ChildNodes

                ' Bild auswählen und laden
                TreeViewImages.Images.Add(New Drawing.Icon(basePath & _
                    element.Attributes("Icon").Value))

                ' Verweis in Hashtable aufnehmen. Key=Elementname,

```

Listing 474: Reload initialisiert das Steuerelement

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Proper-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken**XML**Wissen-  
schaftVer-  
schie-  
denes

---

```

        ' Value = Index
        imageIndex(element.Attributes("Tagname").Value) = _
            TreeViewImages.Images.Count-1

    Next

    ' Imageliste zuordnen
    DataTree.ImageList = TreeViewImages

Else
    ' Keine Imageliste, wenn keine Definitionsdatei angegeben ist
    DataTree.ImageList = Nothing

End If

' Neu zeichnen
Refresh()

Catch ex As Exception
    ' Im Fehlerfall Exception abfangen und TreeView leeren
    Debug.WriteLine(ex.Message)
    DataTree.Nodes.Clear()
End Try

End Sub

```

---

*Listing 474: Reload initialisiert das Steuerelement (Forts.)*

In der überschriebenen Methode `Refresh` (Listing 475) wird das `TreeView`-Control neu aufgebaut. Zunächst wird der Stammknoten neu angelegt und das Symbol zugeordnet, dann werden dessen Kindknoten aufgebaut. Die Ermittlung der anzuzeigenden Texte, die ja aus einem festgelegten Attribut entnommen werden sollen, übernimmt die Methode `GetXmlValue` (Listing 476), den Aufbau der Kindknoten `FillSubTree` (Listing 477).

---

```

Public Overrides Sub Refresh()
    Try
        ' Knotenliste leeren
        DataTree.Nodes.Clear()

        ' Stammknoten hinzufügen
        treeRoot = DataTree.Nodes.Add(GetXmlValue(doc.DocumentElement))

        ' Bildindex ermitteln
        treeRoot.ImageIndex = CInt(imageIndex(doc.DocumentElement.Name))
        treeRoot.SelectedImageIndex = treeRoot.ImageIndex

        ' Tag verweist auf das Element
        treeRoot.Tag = doc.DocumentElement

        ' Kindknoten anzeigen und öffnen
    
```

---

*Listing 475: Neuorganisation des TreeView-Steuerelementes*

```

        FillSubTree(treeRoot)
        treeRoot.Expand()

    Catch ex As Exception
        Debug.WriteLine(ex.Message)
    End Try

End Sub

```

Listing 475: Neuorganisation des TreeView-Steuerelementes (Forts.)

GetXmlValue ermittelt für das angegebene Element, welches Attribut angezeigt werden soll. Der Wert dieses Attributes wird gelesen und als Funktionswert zurückgegeben. FillSubTree legt für jeden Unterknoten einen TreeView-Knoten an und weist die Texte und Symbole zu. Die Tag-Eigenschaft eines jeden Knotens verweist auf das zugehörige XmlElement-Objekt. So kann bei Auswahl eines Knotens direkt auf die untergeordneten Daten zugegriffen werden.

```

Private Function GetXmlValue(ByVal element As XmlElement) As String

    ' Elementname ermitteln
    Dim tn As String = element.Name
    ' Eintrag in Definitionsliste suchen
    Dim xp As String = "x:EntryDefinition[@TagName='" & tn & "']"
    Dim el As XmlNode = edDoc.DocumentElement.SelectSingleNode(xp, _
        edDocNsmgr)
    ' Ermitteln, welches Attribut den Text liefern soll
    Dim att As String = el.Attributes("DefaultAttribute").Value
    ' Dieses Attribut lesen
    Return element.Attributes(att).Value

End Function

```

Listing 476: Attributwert ermitteln, der in der TreeView angezeigt werden soll

```

Private Sub FillSubTree(ByVal treeRoot As TreeNode)

    ' Elternknoten aus Tag-Referenz lesen
    Dim rootElement As XmlElement = _
        DirectCast(treeRoot.Tag, XmlElement)

    ' Aktuellen Zweig löschen
    treeRoot.Nodes.Clear()

    ' Alle unmittelbaren Nachfolger in der XML-Datendatei durchlaufen
    For Each element As XmlElement In rootElement.ChildNodes
        ' TreeView-Knoten anlegen und Bild zuordnen
        Dim node As TreeNode = treeRoot.Nodes.Add(GetXmlValue(element))
        node.Tag = element
        node.ImageIndex = CInt(imageIndex(element.Name))
        node.SelectedIndex = node.ImageIndex
    Next

```

Listing 477: Anlegen untergeordneter Knoten in FillSubTree

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung  
Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

 Next

 End Sub
 

---

*Listing 477: Anlegen untergeordneter Knoten in FillSubTree (Forts.)*

Wählt der Anwender einen Eintrag in der TreeView aus, dann wird im AfterSelect-Ereignis (Listing 478) dieser Knoten erweitert und aufgeklappt, sofern er Unterknoten besitzt. Alle zugehörigen Attribute werden mittels ShowDetails (Listing 479) in der ListView angezeigt.

---

```
Private Sub Dattree_AfterSelect(ByVal sender As System.Object, _
    ByVal e As System.Windows.Forms.TreeViewEventArgs) _
    Handles DataTree.AfterSelect

    ' Unterknoten anlegen und aufklappen
    FillSubTree(e.Node)
    e.Node.Expand()

    ' Details zu ausgewähltem Knoten in der ListView anzeigen
    ShowDetails(DirectCast(e.Node.Tag, XmlElement))
```

 End Sub
 

---

*Listing 478: Erweitern eines Knotens und Anzeigen der Eigenschaften nach Auswahl durch den Anwender*

---

```
Private Sub ShowDetails(ByVal element As XmlElement)

    ' ListView leeren
    DataDetail.Items.Clear()

    ' Alle Attribute des ausgewählten Knotens durchlaufen
    For Each att As XmlAttribute In element.Attributes
        ' Attributnamen und -wert in ListView darstellen
        Dim lvi As ListViewItem = DataDetail.Items.Add(att.Name)
        lvi.SubItems.Add(att.Value)
    Next
```

 End Sub
 

---

*Listing 479: Auflisten aller zum ausgewählten Knoten gehörenden Attribute*

Damit zu jedem Knoten die Elementnamen als ToolTip angezeigt werden können, muss bei jeder Mausbewegung über dem TreeView-Steuerelement ermittelt werden, über welchem Knoten sich der Mauszeiger befindet (Listing 480). Der Elementname dieses Knotens wird dann durch den Aufruf von SetToolTip dem ToolTip-Control, das der ListView zugeordnet ist, zugewiesen.

---

```
Private Sub Dattree_MouseMove(ByVal sender As System.Object, _
    ByVal e As System.Windows.Forms.MouseEventArgs) _
    Handles DataTree.MouseMove
```

---

*Listing 480: Kontinuierliche Festlegung des ToolTip-Textes bei Mausbewegungen*

```
' Knoten der TreeView aus den Mauskoordinaten ermitteln
Dim node As TreeNode = DataTree.GetNodeAt(e.X, e.Y)

' Ignorieren, wenn der Cursor nicht über einem Knoten steht
If node Is Nothing Then Exit Sub

' ToolTip-Text festlegen
ToolTipTag.SetToolTip(DataTree, DirectCast(node.Tag, _
    XmlElement).Name)

End Sub
```

Listing 480: Kontinuierliche Festlegung des ToolTip-Textes bei Mausbewegungen (Forts.)

## 271 Nachrichten aus RSS-Kanälen aufbereiten

Viele Nachrichtenagenturen und Website-Betreiber bieten Kurzfassungen ihrer Informationen in speziellen XML-Dateien an. Weit verbreitet ist das RSS-Format. RSS steht für *RDF SiteSummary*, eine einfache Variante der *RDF*-Spezifikation. *RDF* wiederum steht für Resource Description Framework und beschreibt den Aufbau solcher Dateien. Die Definition des W3C finden Sie unter:

<http://www.w3.org/RDF>

RSS gibt es in verschiedenen Versionen. Aktuell ist die Version 2.0, zu der Sie unter <http://blogs.law.harvard.edu/tech/rss> eine Beschreibung finden können. Ein Beispiel, wie eine solche Datei aufgebaut ist, sehen Sie in Listing 481. Innerhalb des <channel>-Elementes werden weitere Detailinformationen zur Quelle wie Titel, Beschreibung und Sprache definiert. Artikel, auf die verwiesen wird, werden in <item>-Elementen näher beschrieben. Üblich sind hier das Datum der Veröffentlichung, der Titel, eine Kurzbeschreibung sowie der Link zur HTML-Seite des Artikels. Weitere optionale Angaben sind möglich. In der o.g. Beschreibung finden Sie weitere Angaben über erlaubte XML-Elemente.

```
<rss version="2.0">
  <channel>
    <title>MSDN: Visual Basic</title>
    <link>http://msdn.microsoft.com/vbasic/</link>
    <description>The latest technical information On Visual Basic...
    <language>en-us</language>
    <ttl>1440</ttl>
    <item>
      <title>Blogging: Design Your Own Weblog Application from Sc...
      <pubDate>Fri, 26 Sep 2003 07:00:00 GMT</pubDate>
      <description>In this article the author builds a full-featu...
      <link>http://msdn.microsoft.com/msdnmag/issues/03/10/Bloggi...
    </item>
    ...
  </channel>
</rss>
```

Listing 481: Beispiel einer RSS-Datei mit Artikeln zu VB.NET

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

Inzwischen gibt es zahlreiche Programme, die einen komfortablen Umgang mit diesen Informationsquellen, auch Newsfeeder genannt, ermöglichen. In diesem Rezept wollen wir Ihnen zeigen, wie Sie selbst die Informationen in Ihren Windows-Applikationen nutzen können.

Zunächst wird eine Liste der verfügbaren RSS-Quellen benötigt. Diese wird in Form einer XML-Datei (*RSS.xml*, siehe Listing 482) bereitgestellt. Für jeden Channel wird der Titel und der zugehörige URL gespeichert. Beim Start des Programms wird diese Liste gelesen und die Titel in einer ListBox angezeigt (Listing 483).

---

```
<?xml version="1.0" encoding="utf-8" ?>
<Channels>
  <Channel>
    <Title>MSDN: Visual Basic</Title>
    <Link>http://msdn.microsoft.com/vbasic/rss.xml</Link>
  </Channel>
  <Channel>
    <Title>MSDN Just Published</Title>
    <Link>http://msdn.microsoft.com/rss.xml</Link>
  </Channel>
  <Channel>
    <Title>MSDN: Visual C#</Title>
    <Link>http://msdn.microsoft.com/vcsharp/rss.xml</Link>
  </Channel>
  ...
  <Channel>
    <Title>dotnetpro News Feed</Title>
    <Link>http://www.dotnetpro.de/rss/newstopten.ashx</Link>
  </Channel>
</Channels>
```

---

*Listing 482: RSS.xml hält die Liste der RSS-Kanäle bereit*

---

```
Imports System.Xml
Imports System.Xml.XPath

...

' Dokument und Navigator für Quellenliste
Protected RssSourcesDoc As XPathDocument
Protected RssSourcesNav As XPathNavigator

...

' XML-Datei mit Channelliste laden und Navigator erzeugen
RssSourcesDoc = New XPathDocument( _
    Application.StartupPath & "..\rss.xml")
RssSourcesNav = RssSourcesDoc.CreateNavigator()

' Titel der Channels suchen
Dim ni As XPathNodeIterator = _
    RssSourcesNav.Select("//Channel/Title")
```

---

*Listing 483: Lesen und Anzeigen der Kanalliste im Load-Ereignis*

```
' und in ListBox anzeigen
Do While ni.MoveNext()
    LBFeeder.Items.Add(ni.Current.Value)
Loop
```

Listing 483: Lesen und Anzeigen der Kanalliste im Load-Ereignis (Forts.)

Aus der Liste der RSS-Quellen kann der Anwender eine Quelle wählen. Zu dieser Quelle wird der Link ermittelt und die Datei geladen. Abbildung 239 zeigt die Registerkarte QUELLEN mit der Liste der Channels und der Detailbeschreibung des Channels MSDN VISUAL BASIC.

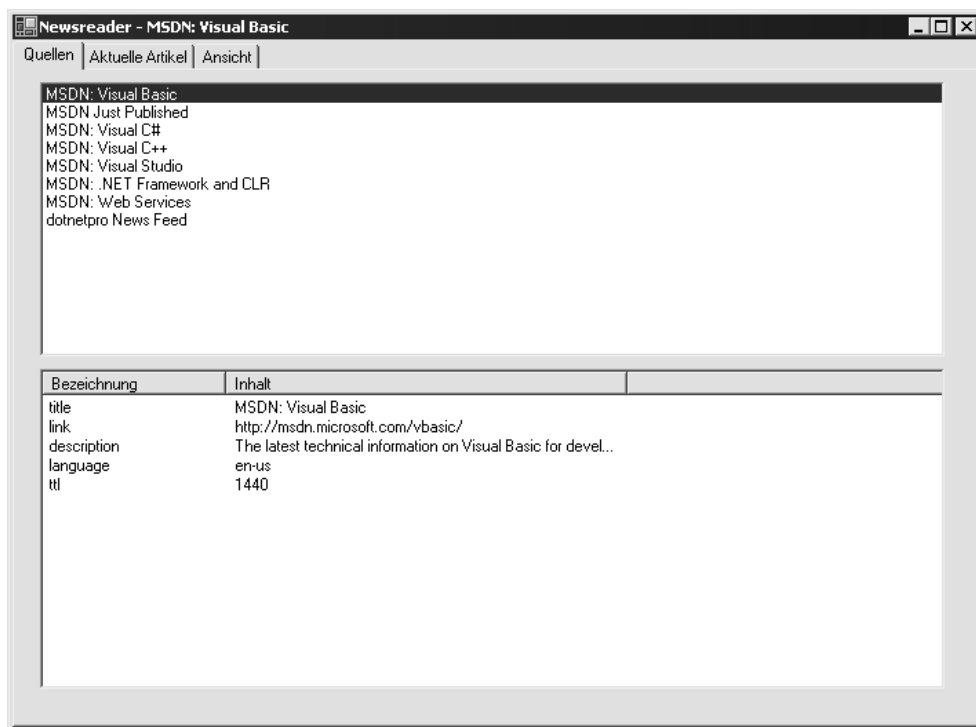


Abbildung 239: Der Newsreader zeigt die Liste der RSS-Quellen und Details zur ausgewählten Quelle an

Ein einfacher Klick auf einen der Listeneinträge ruft die Methode `LoadFeeder` (Listing 484) auf. Aus dem angezeigten Text des ausgewählten Eintrags wird über eine XPath-Abfrage mit `Evaluate` der Link zur Quelldatei bestimmt. Diese Datei wird dann geöffnet und ein `XPathNavigator`-Objekt dazu angelegt. Die Referenzen der beiden Objekte werden für spätere Zugriffe in den Membervariablen `RssChannelDoc` und `RssChannelNav` gespeichert.

Um die Detailinformationen zur ausgewählten Nachrichtenquelle zu erhalten, werden im nächsten Schritt alle Elemente gesucht, die nicht vom Typ `<item>` sind. Hierzu wird der XPath-Ausdruck `//channel/*[local-name()!='item']` verwendet. Alle gefundenen Elemente werden mit Namen und Wert in die Detailliste eingetragen (siehe Abbildung 239, untere ListView).

---

Protected Sub LoadFeeder()

If LBFeeder.SelectedItem Is Nothing Then Return

*' Cursor merken und umschalten*

Dim cur As Cursor = Me.Cursor

Cursor = Cursors.WaitCursor

*' Liste leeren*

LVDetails.Items.Clear()

*' Titelzeile setzen*

Me.Text = "Newsreader-" &amp; LBFeeder.SelectedItem.ToString()

*' Link zum Channel ermitteln*

Dim link As String = CStr(RssSourcesNav.Evaluate( \_  
"String(//Channel[Title='" & LBFeeder.SelectedItem.ToString() \_  
& "']/Link)")

*' XML-Dokument laden, Navigator anlegen*

RssChannelDoc = New XPathDocument(link)

RssChannelNav = RssChannelDoc.CreateNavigator()

*' Alle Einträge ungleich <item> suchen*

Dim ni As XPathNodeIterator = RssChannelNav.Select( \_  
"//channel/\*[local-name()!='item']")

*' und mit Name und Wert in Detailliste anzeigen*

Do While ni.MoveNext()

Dim lvi As ListViewItem = LVDetails.Items.Add(ni.Current.Name)  
lvi.SubItems.Add(ni.Current.Value)

Loop

*' Titelliste leeren*

LVTitles.Items.Clear()

*' XPath-Ausdrücke kompilieren*

expTitle = RssChannelNav.Compile("String(title)")

expLink = RssChannelNav.Compile("String(link)")

expDescription = RssChannelNav.Compile("String(description)")

expDate = RssChannelNav.Compile("String(pubDate)")

*' Nun die <item>-Einträge suchen*

ni = RssChannelNav.Select("//channel/item")

*' und jeweils Datum der Veröffentlichung und Titel in die Liste  
' eintragen*

Do While ni.MoveNext()

Try

Dim day As DateTime = \_  
DateTime.Parse(CStr(ni.Current.Evaluate(expDate)))



```

        Dim lvi As ListViewItem = LVTitles.Items.Add( _
            day.ToShortDateString())
        lvi.SubItems.Add(CStr(ni.Current.Evaluate(expTitle)))
        lvi.Tag = ni.Current.Clone()
    Catch
    End Try

Loop

' Cursor zurücksetzen
Me.Cursor = cur

End Sub

```

Listing 484: LoadFeeder lädt Detailinformationen zur Quelle und die Artikelliste (Forts.)

Für die spätere Suche werden einige XPath-Ausdrücke kompiliert und gespeichert (Membervariablen `expTitle`, `expLink`, `expDescription` und `expDate`). Dann erfolgt der Aufbau der Artikelliste, einer `ListView` auf der zweiten Registerkarte. Hierfür werden alle `<item>`-Elemente ermittelt und für jedes Element mit Hilfe der vorkompilierten Ausdrücke das Datum der Veröffentlichung sowie der Titel des Artikels gelesen und in die Liste aufgenommen. Bei jedem `ListView`-Eintrag wird der Tag-Eigenschaft eine Kopie des zum Knoten gehörenden `XPathNavigator`-Objektes zugewiesen. So kann später schnell auf die gespeicherten Informationen zugegriffen werden, ohne erneut das `<item>`-Element suchen zu müssen.

Zusätzlich wird das `DoubleClick`-Ereignis der `ListBox` abgefangen und dort mit

```
TabControl1.SelectedIndex = 1
```

auf die nächste Registerkarte weitergeschaltet. So kann der Anwender schnell zum nächsten Register navigieren, ohne selbst auf die Kartenreiter klicken zu müssen.

Abbildung 240 zeigt die zweite Registerkarte, die alle in der Quelldatei aufgeführten Artikel auflistet. Bei Auswahl eines Eintrags wird im unteren Bereich des Fensters die zum Artikel gehörende Kurzbeschreibung ausgegeben. Im `SelectedIndexChanged`-Ereignis-Handler der `ListView LVTitles` (Listing 485) wird über die in der Tag-Eigenschaft gespeicherte Referenz das Navigator-Objekt abgerufen, das auf den zugehörigen Knoten verweist. Über dieses wird mit einer gespeicherten Abfrage die Beschreibung des Artikels geladen und in der schreibgeschützten `TextBox` angezeigt.

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Propert-  
yGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

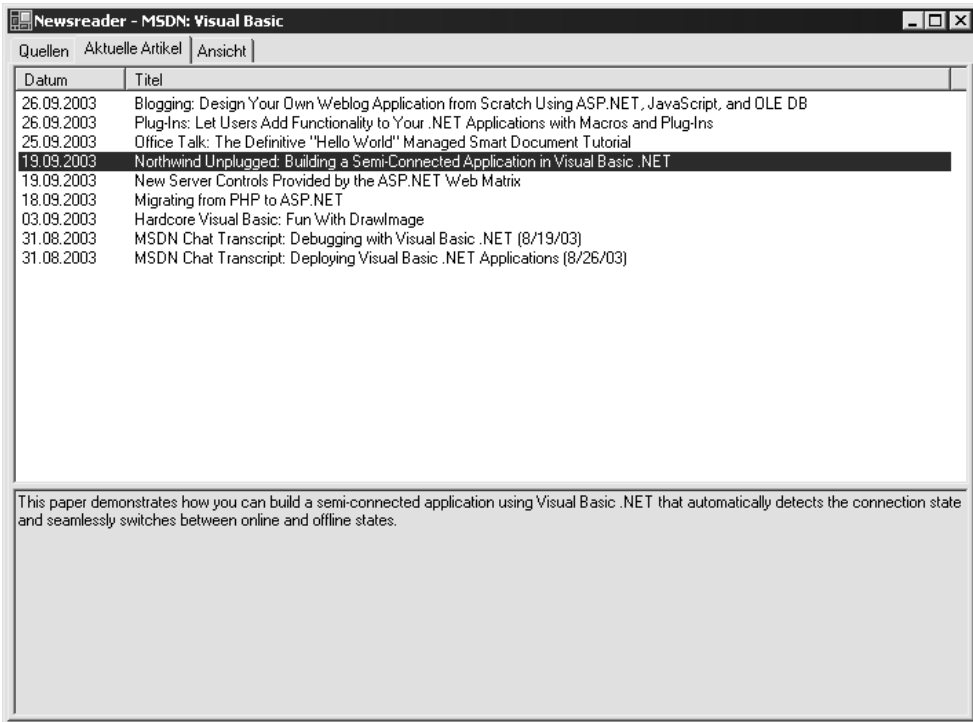


Abbildung 240: Anzeigen der aktuellen Artikel eines Channels und der Kurzbeschreibung zum ausgewählten Artikel

```
Private Sub LVTitles_SelectedIndexChanged(ByVal sender As _
    System.Object, ByVal e As System.EventArgs) _
    Handles LVTitles.SelectedIndexChanged

    If LVTitles.SelectedItems.Count < 1 Then Return

    ' Tag verweist auf den Navigator für das <item>-Element
    Dim nav As XPathNavigator = DirectCast( _
        LVTitles.SelectedItems(0).Tag, XPathNavigator)

    ' Beschreibung des Artikels laden und anzeigen
    TBDescription.Text = CStr(nav.Evaluate(expDescription))

End Sub
```

Listing 485: Bei Auswahl eines Artikels wird dessen Kurzbeschreibung angezeigt

Ein Doppelklick auf einen Artikel soll diesen Artikel im WebBrowser-Steuerelement auf der dritten Registerkarte laden und darstellen. Listing 486 zeigt die dafür notwendige Implementierung. Wieder wird über die Tag-Eigenschaft des ListView-Eintrags der Navigator abgerufen und mittels `Evaluate` eine XPath-Abfrage, hier zum Link der HTML-Seite, abgesetzt. Der erhaltene Link wird dem WebBrowser-Steuerelement zur Navigation übergeben.

```

Private Sub LVTitles_DoubleClick(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles LVTitles.DoubleClick

    If LVTitles.SelectedItems.Count < 1 Then Return

    ' Tag verweist auf den Navigator für das <item>-Element
    Dim nav As XPathNavigator = DirectCast( _
        LVTitles.SelectedItems(0).Tag, XPathNavigator)

    ' Link laden und im Browser-Steuerelement anzeigen
    Dim link As String = CStr(nav.Evaluate(expLink))
    AxWebBrowser1.Navigate(link)

    ' Tabseite umschalten
    TabControl1.SelectedIndex = 2

    ' Titelzeile setzen
    Me.Text = "Newsreader-" & LBFeeder.SelectedItem.ToString() _
        & "-" & LVTitles.SelectedItems(0).SubItems(1).Text

End Sub

```

Listing 486: Bei Doppelklick Artikel auswählen und auf Browser-Ansicht umschalten



Abbildung 241: Der letzte Schritt: Laden des Artikels im WebBrowser-Control

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Proper-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

Abschließend wird auf die dritte Registerkarte umgeschaltet. So kommt der Benutzer mit nur zwei Doppelklicks von der Liste der Datenquellen bis zur HTML-Darstellung des gesuchten Artikels. In Abbildung 241 sehen Sie die dritte Registerkarte, auf der der gewählte Artikel in der HTML-Ansicht angezeigt wird.

## 272 Das Wichtigste der Tagesschau im UserControl

Auch in Ihren Windos-Anwendungen können Sie die neuesten Nachrichten präsentieren. Ein Benutzersteuerelement lädt die aktuellen Schlagzeilen von einem Newsfeeder (z.B. von der Tagesschau unter <http://www.tagesschau.de/xml/tagesschau-meldungen>) und zeigt sie in Form von Link-Label-Steuerelementen an (siehe Abbildung 242). Über die ToolTip-Anzeige kann der Anwender schnell eine Kurzfassung des Artikels lesen. Durch Klick auf die LinkLabel-Schaltfläche wird die zugehörige Nachricht im Browser angezeigt.

Als Nachrichtenquelle sind alle RDF- und RSS-Anbieter geeignet (siehe vorheriges Rezept). Der Anbieter tagesschau.de steht hier nur stellvertretend für viele andere. Achten Sie auch auf den Aufbau der XML-Dateien, der recht unterschiedlich ausfallen kann. Viele optionale Elemente und Namensräume erschweren einen allgemeingültigen Ansatz.

Voraussetzung für die Nutzbarkeit des UserControls ist natürlich, dass eine Verbindung zum Internet besteht. Für Anwender, die nicht über eine ständige Internetverbindung verfügen, sondern die Verbindung immer von Hand auf- und abbauen müssen, ist das Tool weniger geeignet. Beachten Sie auch bitte die Nutzungsbestimmungen des Newsfeed-Betreibers (siehe Kasten).

### Hinweis

Wenn Sie Inhalte fremder Datenquellen in Ihren Programmen nutzen wollen, beachten Sie bitte die Bestimmungen des Anbieters, die meist als Kommentar in den XML-Dateien enthalten sind. tagesschau.de z.B. lässt ausschließlich nicht-kommerzielle Nutzungen zu und verlangt, dass die Schlagzeilen mit den hinterlegten Meldungen verlinkt werden. Auch müssen Sie in der Regel die Quelle angeben, entweder, wie im Beispiel, durch ein Logo oder durch explizite Nennung des Anbieters.



Abbildung 242: Aktuelle Schlagzeilen mit Link und Kurzbeschreibung

Die öffentliche Methode `Reload` (Listing 487) wird beim Laden des Steuerelementes bzw. bei Bedarf (z.B. über einen Timer) aufgerufen. Zunächst werden eventuell von einem früheren Aufruf

bereits vorhandene `LinkLabel`-Steuerelemente entfernt (siehe `RemoveLinkLabels`, Listing 488). Anschließend wird die XML-Datei geladen und ein Navigator-Objekt angelegt.

Beachten Sie bei Verwendung der XML-Nachrichtendateien insbesondere die definierten Namensräume, da Sie diese bei XPath-Abfragen berücksichtigen müssen. Hier wird der Namespace `http://purl.org/rss/1.0/` als Standard-Namespace dem Namespace-Manager hinzugefügt (Prefix: `x`). Eine Reihe kompilierter XPath-Ausdrücke, die den definierten Prefix verwenden, erleichtern den Zugriff auf die benötigten Elemente.

Das Logo des Anbieters wird in einer `PictureBox` angezeigt, die im Steuerelement links oben angeordnet ist. Die `SizeMode`-Eigenschaft dieser `PictureBox` wird auf `AutoSize` eingestellt, so dass die Größe automatisch angepasst wird. Der Pfad der Bilddatei wird aus `image/url` ermittelt und das Bild über `Webrequest / Webresponse` geladen.

---

```
Public Sub Reload()

    RemoveLinkLabels()

    ' RDF-Datei laden
    Dim NewsDoc As New XPathDocument( _
        "http://www.tagesschau.de/xml/tagesschau-meldungen/")

    ' Navigator anlegen
    Dim NewsNav As XPathNavigator = NewsDoc.CreateNavigator()

    ' Namespace-Manager und Standardnamespace anlegen
    Dim nsmgr As New XmlNamespaceManager(NewsNav.NameTable)
    nsmgr.AddNamespace("x", "http://purl.org/rss/1.0/")

    ' XPath-Suchausdrücke kompilieren
    ' <link>
    Dim expLink As XPathExpression = NewsNav.Compile("String(x:link)")
    expLink.SetContext(nsmgr)

    ' <url>
    Dim expImgUrl As XPathExpression = NewsNav.Compile( _
        "String(//x:image/x:url)")

    ' <title>
    expImgUrl.SetContext(nsmgr)
    Dim expTitle As XPathExpression = _
        NewsNav.Compile("String(x:title)")
    expTitle.SetContext(nsmgr)

    ' <description>
    Dim expDescription As XPathExpression = _
        NewsNav.Compile("String(x:description)")
    expDescription.SetContext(nsmgr)

    ' <item>
    Dim expItem As XPathExpression = NewsNav.Compile("//x:item")
```

---

Listing 487: Laden der XML-Nachrichtendatei und Aufbau der Links

---

Basics

---

Datum/  
Zeit

---

Anwen-  
dungen

---

Zeich-  
nen

---

Bild-  
bearbei-  
tung

---

Win-  
dows  
Forms

---

Controls

---

Propert-  
yGrid

---

Datei-  
system

---

Netz-  
werk

---

System

---

Daten-  
banken

---

**XML**

---

Wissen-  
schaft

---

Ver-  
schiede-  
nes

---

```

expItem.SetContext(nsmgr)

' Bilddatei für Logo ermitteln
Dim imgPath As String = CStr(NewsNav.Evaluate(expImgUrl))

' Datei laden
Dim wr As WebRequest = WebRequest.Create(imgPath)
' Auf Response warten
Dim ws As WebResponse = wr.GetResponse()
' Bitmap aus Stream erstellen
Dim s As Stream = ws.GetResponseStream()

' Logo anzeigen (Autosize passt Größe der PictureBox an)
PBLogo.Image = New Bitmap(s)

' Position für nächstes LinkLabel
Dim pos As Integer = PBLogo.Top + PBLogo.Height + 10

' Tabindex für nächstes LinkLabel
Dim ti As Integer = 1

' Liste der <item>-Elemente durchlaufen
Dim ni As XPathNodeIterator = NewsNav.Select(expItem)

Do While ni.MoveNext()

    ' neues LinkLabel anlegen
    Dim lnkLbl As New LinkLabel

    ' Position setzen
    lnkLbl.Location = New System.Drawing.Point(0, pos)

    ' Maximale Breite
    lnkLbl.Width = Me.ClientSize.Width

    ' Tab-Eigenschaften
    lnkLbl.TabIndex = ti
    lnkLbl.TabStop = True

    ' Text des LinkLabels = Titel der Nachricht
    lnkLbl.Text = CStr(ni.Current.Evaluate(expTitle))

    ' ToolTipText des LinkLabels = umgebrochener Text der
    ' Beschreibung (<description>)
    Me.ToolTip1.SetToolTip(lnkLbl, _
        BreakString(CStr(ni.Current.Evaluate(expDescription))))

    ' Tag-Eigenschaft verweist auf Link zur Nachricht
    lnkLbl.Tag = CStr(ni.Current.Evaluate(expLink))

    ' Organisatorisches

```

---

```

Me.Controls.Add(lnkLbl)
AddHandler lnkLbl.Click, AddressOf LLClick
pos += lnkLbl.Height
ti += 1

```

Loop

```

' Einstellung für vertikale Bildlaufleiste
Me.AutoScrollMinSize = New Size(0, pos)

```

End Sub

*Listing 487: Laden der XML-Nachrichtendatei und Aufbau der Links (Forts.)*

In einer Schleife über alle <item>-Objekte werden die LinkLabel-Steuer-elemente erzeugt und positioniert. Der Schlagzeilentitel wird der Text-Eigenschaft zugewiesen, der Link auf die verbundene HTML-Seite der Tag-Eigenschaft. Eine Kurzfassung der Nachricht aus dem Element <description> wird als ToolTip-Text hinterlegt. Das Click-Ereignis eines jeden LinkLabels wird an den gemeinsamen Ereignis-Handler LLClick (siehe Listing 489) gebunden.

```

Private Sub RemoveLinkLabels()

    ' Vorhandene LinkLabels entfernen
    Dim c As Control
    For i As Integer = Controls.Count - 1 To 0 Step -1
        If TypeOf Controls(i) Is LinkLabel Then Controls.RemoveAt(i)
    Next

End Sub

```

*Listing 488: Entfernen vorhandener LinkLabels*

Bei Klick auf eines der LinkLabel-Steuer-elemente wird der Handler LLClick aufgerufen. sender verweist auf das auslösende Steuer-element und kann auf LinkLabel gecastet werden. Der Tag-Eigenschaft wurde zuvor der URL der Nachrichtenseite zugeordnet, so dass dieser als String gelesen werden kann. Um ein ähnliches Verhalten wie im Browser zu simulieren, wird das LinkLabel als »besucht« gekennzeichnet und erhält dadurch eine andere Textfarbe. Im letzten Schritt wird über Process.Start die Nachrichtenseite im Standard-Browser angezeigt.

```

Private Sub LLClick(ByVal sender As Object, ByVal e As EventArgs)

    ' sender ist ein LinkLabel
    Dim lnkLbl As LinkLabel = DirectCast(sender, LinkLabel)

    ' Tag verweist auf HTML-Seite
    Dim link As String = CStr(lnkLbl.Tag)

    ' LinkLabel As "besucht" kennzeichnen
    lnkLbl.LinkVisited = True

```

*Listing 489: Bei Mausklick auf Label die Nachricht im Browser anzeigen*

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung  
Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

```
' Seite im Browser öffnen  
Process.Start(link)
```

```
End Sub
```

---

*Listing 489: Bei Mausklick auf Label die Nachricht im Browser anzeigen (Forts.)*

## 273 XML-Dateien validieren

Als Grundvoraussetzung für die Bearbeitung von XML-Dateien mit `XmlTextReader`, `XmlDocument`, `XPathDocument` usw. gilt, dass es sich um wohlgeformte, also syntaktisch richtige XML-Notationen handelt. Stellen die Framework-Methoden einen Fehler fest, lehnen sie die weitere Bearbeitung ab und lösen eine Exception aus. Weitere Überprüfungen finden nicht automatisch statt.

Oft ist es aber notwendig, dass eine XML-Datei nach einem vorgegebenen Bauplan aufgebaut ist, damit sich verarbeitende Programme auf die Einhaltung einer Struktur verlassen können. Auch die Einhaltung von Datentypen ist oft relevant, damit sich Elementwerte schnell und sicher umwandeln lassen.

Um den Aufbau einer XML-Datei definieren zu können, verwendet man ein Schema. Zwei Arten von Schemata sind heute gebräuchlich. Zum einen die inzwischen veraltete, aber immer noch häufig anzutreffende *DTD* (Document Type Definition), zum anderen die *XSD* (XML Schema Definition), die zukünftig DTDs ersetzen soll. Näheres dazu finden Sie beim W3C unter

<http://www.w3.org/TR/xmlschema-0/>

<http://www.w3.org/TR/xmlschema-1/>

<http://www.w3.org/TR/xmlschema-2/>

Microsoft unterstützt in den XML-Klassen beide Arten von Schemata. Im Umgang mit der Entwicklungsumgebung erkennt man allerdings sehr bald, dass der Schwerpunkt bei XSD-Schemadateien liegt, die auch für andere Zwecke (Datenbankdesign) zum Einsatz kommen. Bei zukünftigen Entwicklungen sollte man daher XSD den Vorzug gegenüber DTD geben.

Auch XSD-Dateien sind gültige XML-Dateien, gehorchen also der XML-Syntax. Sie beschreiben den Aufbau einer XML-Datei und legen somit die Richtlinien fest, die eine XML-Datei erfüllen muss, damit sie im Sinne des Schemas gültig ist. Den Aufbau von XSD-Dateien im Detail zu erklären würde den Rahmen dieses Buches bei weitem sprengen. Stattdessen soll anhand eines Beispiels die Validierung vorhandener XML-Dateien mit vorhandenen Schemata erklärt werden.

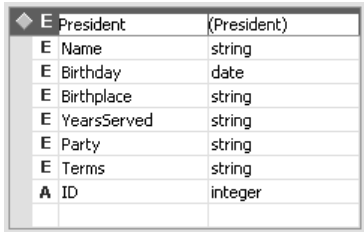
Für die bereits in mehreren Rezepten verwendete XML-Datei *Presidents.xml* wurde ein Schema definiert. Die XML-Notation dieses Schemas sehen Sie in Listing 490. Visual Studio unterstützt noch eine andere Ansicht, die die Sichtweise eines Schemas als Datenbankstruktur grafisch aufbereitet. Ein Beispiel hierfür sehen Sie in Abbildung 243. In den vorangegangenen Beispielen wurden alle Element- und Attribute grundsätzlich als String verarbeitet. In der Schemadefinition wurde nun abweichend davon für das Attribut `ID` der Typ `integer` und für das Element `Birthplace` der Typ `date` verwendet. Die bisher verwendete deutsche Notation für das Datum wurde ersetzt durch die ISO-Notation. Listing 491 zeigt einen Ausschnitt der geänderten Datei. Als Namensraum wird die Schemadefinition angegeben.



```
<?xml version="1.0" ?>
<xs:schema id="Presidents"
  targetNamespace="http://tempuri.org/presidents.xsd"
  xmlns:mstns=http://tempuri.org/presidents.xsd
  xmlns="http://tempuri.org/presidents.xsd"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:msdata="urn:schemas-microsoft-com:xml-msdata"
  attributeFormDefault="qualified" elementFormDefault="qualified">

  <xs:element name="Presidents" msdata:IsDataSet="True"
    msdata:Locale="de-DE" msdata:EnforceConstraints="False">
    <xs:complexType>
      <xs:choice maxOccurs="unbounded">
        <xs:element name="President">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Name" type="xs:String" minOccurs="0"
                msdata:Ordinal="0" />
              <xs:element name="Birthday" type="xs:Date"
                minOccurs="0" msdata:Ordinal="1" />
              <xs:element name="Birthplace" type="xs:String"
                minOccurs="0" msdata:Ordinal="2" />
              <xs:element name="YearsServed" type="xs:String"
                minOccurs="0" msdata:Ordinal="3" />
              <xs:element name="Party" type="xs:String"
                minOccurs="0" msdata:Ordinal="4" />
              <xs:element name="Terms" type="xs:String"
                minOccurs="0" msdata:Ordinal="5" />
            </xs:sequence>
            <xs:attribute name="ID" form="unqualified"
              type="xs:Integer" />
          </xs:complexType>
        </xs:element>
      </xs:choice>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Listing 490: XSD-Schema für Presidents.xml



| ◆ E | President   | {President} |
|-----|-------------|-------------|
| E   | Name        | string      |
| E   | Birthday    | date        |
| E   | Birthplace  | string      |
| E   | YearsServed | string      |
| E   | Party       | string      |
| E   | Terms       | string      |
| A   | ID          | integer     |

Abbildung 243: Das Schema in der Dataset-Ansicht

```

<?xml version="1.0" encoding="Windows-1252"?>
<Presidents xmlns="http://tempuri.org/presidents.xsd">
  <President ID="25">
    <Name>William McKinley</Name>
    <Birthday>1843-01-29</Birthday>
    <Birthplace>Niles, Ohio</Birthplace>
    <YearsServed>1897-1901</YearsServed>
    <Party>Republican</Party>
    <Terms>1</Terms>
  </President>
  ...
</Presidents>

```

Listing 491: Presidents.xml mit Angabe der Schemadatei und Element <Birthday> im ISO-Format

Eine so aufgebaute XML-Datei lässt sich mit Hilfe der Entwicklungsumgebung bereits ohne eigene Software auf Gültigkeit überprüfen. Wird die XML-Datei im Visual Studio ausgewählt und angezeigt, dann steht im Menü XML der Unterpunkt XML-Daten überprüfen zur Verfügung. Ein Beispiel für das Ergebnis einer solchen Überprüfung zeigt Abbildung 244. In der Aufgabenliste finden Sie die festgestellten Fehler und können mit einem Doppelklick auf eine Fehlermeldung direkt zu der betreffenden Zeile navigieren.

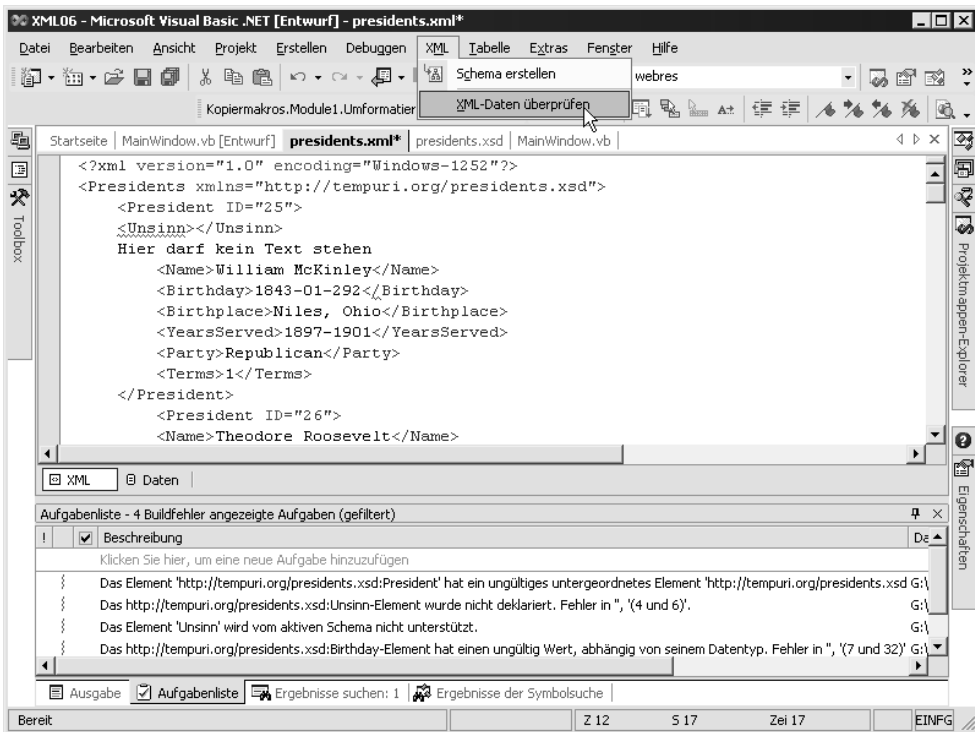


Abbildung 244: XML-Datei mit Hilfe der Entwicklungsumgebung validieren

Programmatisch lässt sich die gleiche Überprüfung mit Hilfe der Klasse `XmlValidatingReader` durchführen. Der Einsatz dieses Readers wird dabei zwischen das Öffnen der Datei mit dem `XmlTextReader` und der weiteren Verarbeitung geschoben. Im Beispiel in Listing 492 wird die XML-Datei zunächst mit einer `XmlTextReader`-Instanz geöffnet. Hierauf setzt die Instanz des `XmlValidatingReader` auf. Diese wiederum wird genutzt, um die Datei in ein `XmlDocument`-Objekt zu laden.

Der `XmlValidatingReader` überprüft die XML-Datei knotenweise beim Einlesen. Das heißt, die Validierung findet erst statt, wenn das Dokument geladen wird, und nicht bereits beim Anlegen des Readers. Vor dem Laden des Dokumentes muss das Schema zugewiesen werden. Dieser Vorgang geschieht nicht automatisch!

Syntaktische Fehler werden beim Laden direkt mit einer Exception quittiert und müssen im Catch-Block abgefangen werden. Verstöße gegen das Schema werden hingegen von einem Ereignis-Handler (`XSDErrorHandler`, siehe Listing 493) bearbeitet. Der Handler wird mit `AddHandler` an das `ValidationEventHandler`-Ereignis gebunden. Beide Fehlerarten führen im Beispielprogramm zur Ausgabe einer `MessageBox`, die Ursache und Position näher beschreibt.

```
Imports System.Xml
Imports System.Xml.Schema

...

' XML-Datei mit XmlTextReader öffnen
Dim xtr As New XmlTextReader( _
    Application.StartupPath & "..\presidents.xml")

' ValidatingReader setzt auf XmlTextReader auf
Dim xvr As New XmlValidatingReader(xtr)

' Try/Catch zum Abfangen von XML-Syntaxfehlern
Try

    ' Schemadatei hinzufügen
    xvr.Schemas.Add("http://tempuri.org/presidents.xsd", _
        Application.StartupPath & "..\presidents.xsd")

    ' Validierungstyp einstellen
    xvr.ValidationType = ValidationType.Schema

    ' Eventhandler für Validierungsfehler
    AddHandler xvr.ValidationEventHandler, AddressOf XSDErrorHandler

    ' Neue XmlDocument-Instanz anlegen und mit ValidatingReader
    ' öffnen
    Dim xdoc As New XmlDocument
    xdoc.Load(xvr)

Catch ex As Exception
    ' Syntaxfehler in der XML-Datei
    MessageBox.Show(ex.Message, "XML Syntaxfehler")
```

Listing 492: XML-Datei gegen ein XSD-Schema prüfen

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

```

    ErrorCount += 1

End Try

' Abschlussbehandlung
xvr.Close()

If ErrorCount = 0 Then
    MessageBox.Show("Keine Fehler")
Else
    MessageBox.Show(String.Format("{0} Fehler", ErrorCount))
End If

```

---

*Listing 492: XML-Datei gegen ein XSD-Schema prüfen (Forts.)*

---

```

Private Sub XSDErrorHandler (ByVal sender As Object, _
    ByVal e As ValidationEventArgs)

    ' XML-Datei entspricht nicht dem Schema
    MessageBox.Show(BreakString(e.Message), "XML Validierungsfehler")
    ErrorCount += 1

End Sub

```

---

*Listing 493: Validierungsfehler abfangen*

Nachfolgend finden Sie einige Beispiele für Fehler und deren Erkennung. Verstöße gegen die XML-Syntax führen zum Auslösen einer Exception. Die ungültigen Zeichen im folgenden Ausdruck

---

```

<?xml version="1.0" encoding="Windows-1252"?>
<Presidents xmlns="http://tempuri.org/presidents.xsd">
    <President ID="25">
        <<<<<
        <Name>William McKinley</Name>

```

---

führen zur Ausgabe der in Abbildung 245 gezeigten Fehlermeldung.



*Abbildung 245: Keine korrekte XML-Syntax*

Im nächsten Beispiel ist die XML-Datei wohlgeformt, also syntaxfehlerfrei, definiert aber im Element `<President>` ein Unterelement `<NameOfPet>`, das im Schema nicht definiert ist:

```
<?xml version="1.0" encoding="Windows-1252"?>
<Presidents xmlns="http://tempuri.org/presidents.xsd">
  <President ID="25">
    <Name>William McKinley</Name>
    <Birthday>1843-01-29</Birthday>
    <Birthplace>Niles, Ohio</Birthplace>
    <YearsServed>1897-1901</YearsServed>
    <Party>Republican</Party>
    <Terms>1</Terms>
    <NameOfPet>Pluto</NameOfPet>
  </President>
```

Die Meldung für diesen Fehler sehen Sie in Abbildung 246.

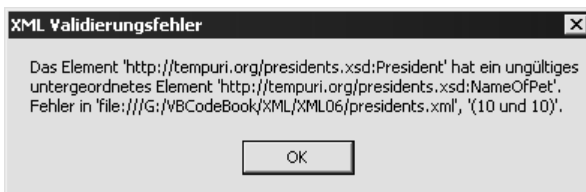


Abbildung 246: <NameOfPet> ist nicht im Schema definiert

Das abschließende Beispiel enthält zwei Fehler:

```
<?xml version="1.0" encoding="Windows-1252"?>
<Presidents xmlns="http://tempuri.org/presidents.xsd">
  <President ID="25">
    <Name>William McKinley</Name>
    <Birthday>1843-01-290</Birthday>
    <Birthplace>Niles, Ohio</Birthplace>
    <YearsServed>1897-1901</YearsServed>
    <Party>Republican</Party>
    <Terms>1</Terms>
    zusätzliche Informationen
  </President>
```

Erstens enthält das Element <Birthday> einen Wert, der nicht in ein Datum gewandelt werden kann, zweitens wurde für das Element <President> ein Textknoten angegeben, der im Schema nicht definiert wurde. Die Fehlermeldungen hierzu zeigen Abbildung 247 und Abbildung 248.



Abbildung 247: Ein Wert entspricht nicht seinem Datentyp

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Propor-  
tityGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

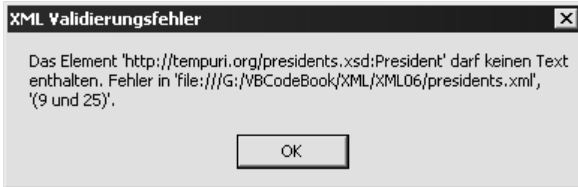


Abbildung 248: Auch die Inhalte werden geprüft

## 274 XSL-Transformationen

XSL (Extensible Stylesheet Language)-Transformationen dienen der Umwandlung einer XML-Datei in ein beliebiges Format. Sie sind nicht nur für Web-Anwendungen, wo man sie häufig antrifft, sondern auch für Windows-Programme interessant. So lassen sich über eine XSLT-Datei die Daten aus XML-Dateien für beliebige Zwecke formatieren. Eine typische Transformation ist die Ausgabe in HTML, um sie für WebBrowser darstellbar zu machen. Aber auch andere Formate (RTF, PDF etc.) sind möglich. Ein weiteres Einsatzgebiet ist die Konvertierung in eine XML-Datei mit einer anderen Struktur.

Auch XSLT ist ein bücherfüllendes Thema, das hier nicht näher behandelt werden kann. Einzelheiten finden Sie unter:

<http://www.w3.org/TR/xslt>

Mit einem kleinen Beispiel wollen wir demonstrieren, wie Sie mit Hilfe einer XSL-Transformation aus der Präsidentenliste eine HTML-Datei generieren können. Eine einfache Transformationsdatei sehen Sie in Listing 494. Sie besteht aus zwei Match-Knoten. Der erste wird aufgerufen für das Stammelement der XML-Datei (<Presidents>), der zweite für jeden enthaltenen <President>-Knoten. Um die Namen eindeutig zuzuordnen, wird dem Standard-Namensraum der XML-Datei hier das Prefix `pres` zugeordnet.

---

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:pres="http://tempuri.org/presidents.xsd">

  <!-- Match für Stammknoten -->
  <xsl:template match="/">
    <!-- Kopf der HTML-Datei mit Tabellenbeginn-->
    <html>
      <head>
        <title>Präsidententabelle </title>
      </head>
      <body>
        <h1>US-Präsidenten des 20. Jahrhunderts</h1>
        <TABLE id="Table1" style="Z-INDEX: 101" cellSpacing="1"
          cellPadding="1" width="600" border="1">
          <TR style="FONT-SIZE: 12pt; FONT-FAMILY: 'Arial Black'">
            <td align="center" width="30">Nr
            </td>
            <TD width="200">Name</TD>
```

---

Listing 494: XSL-Transformation für `President.xml` nach `President.html`

```

        <TD>Amtszeit</TD>
        <TD>Partei</TD>
    </TR>

    <!-- Präsidentendaten hier einfügen -->
    <xsl:apply-templates Select="pres:Presidents"/>

</TABLE>
</body>
</html>
</xsl:template>

<!-- Match für Präsidenten-Knoten -->
<xsl:template match="pres:President">
    <!-- Tabellenzeile zusammensetzen -->
    <TR style="FONT-SIZE: 10pt; FONT-FAMILY: Arial">
        <TD align="center"><xsl:value-of Select="@ID"/></TD>
        <TD ><xsl:value-of Select="pres:Name"/></TD>
        <TD><xsl:value-of Select="pres:YearsServed"/></TD>
        <TD><xsl:value-of Select="pres:Party"/></TD>
    </TR>
</xsl:template>

</xsl:stylesheet>

```

*Listing 494: XSL-Transformation für President.xml nach President.html (Forts.)*

Listing 495 zeigt die für die Ausführung notwendigen Aktionen. Benötigt wird eine Instanz der Klasse `XsltTransform`. Über diese wird das Stylesheet geladen und anschließend die Transformation ausgeführt. Die verwendete Überladung der Methode `Transform` benötigt den Pfad der XML-Quelldatei, den Pfad der Zieldatei und die Referenz eines `XmlResolver`-Objektes. Letzteres kann dazu verwendet werden, Informationen über Zugriffsrechte zu übergeben. Alle Überladungen, die kein `XmlResolver`-Objekt als Parameter erwarten, sind in der Dokumentation als »veraltet« gekennzeichnet und sollten nicht mehr benutzt werden. Wenn keine besonderen Zugriffsrechte benötigt werden, kann statt einer `XmlResolver`-Referenz auch `Nothing` übergeben werden.

Das Ergebnis der Transformation, interpretiert vom Internet Explorer, zeigt Abbildung 249. Die Formatierung der Tabelle lässt sich leicht durch Änderung der HTML-Tags in der XSLT-Datei anpassen.

```

Imports System.Xml
Imports System.Xml.Xsl

...

' Neues Transformationsobjekt
Dim xslt As New XsltTransform

Try
    ' Transformationsdatei laden

```

*Listing 495: Ausführen der XSL-Transformation*

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

```

xslt.Load(Application.StartupPath & "..\Presidents1.xslt")

' Transformation ausführen
xslt.Transform(Application.StartupPath & "..\Presidents.xml", _
    "Presidents1.html", Nothing)

Catch ex As Exception
    MessageBox.Show(ex.Message)
End Try

```

Listing 495: Ausführen der XSL-Transformation (Forts.)

| Nr | Name                      | Amtszeit  | Partei     |
|----|---------------------------|-----------|------------|
| 25 | William McKinley          | 1897-1901 | Republican |
| 26 | Theodore Roosevelt        | 1901-1909 | Republican |
| 27 | William Howard Taft       | 1909-1913 | Republican |
| 28 | Woodrow Wilson            | 1913-1921 | Democratic |
| 29 | Warren G. Harding         | 1921-1923 | Republican |
| 30 | John Calvin Coolidge      | 1923-1929 | Republican |
| 31 | Herbert Hoover            | 1929-1933 | Republican |
| 32 | Franklin D. Roosevelt     | 1933-1945 | Democratic |
| 33 | Harry S. Truman           | 1945-1953 | Democratic |
| 34 | Dwight D. Eisenhower      | 1953-1961 | Republican |
| 35 | John F. Kennedy           | 1961-1963 | Democratic |
| 36 | Lyndon B. Johnson         | 1963-1969 | Democratic |
| 37 | Richard M. Nixon          | 1969-1974 | Republican |
| 38 | Gerald R. Ford            | 1974-1977 | Republican |
| 39 | Jimmy Carter              | 1977-1981 | Democratic |
| 40 | Ronald Reagan             | 1981-1989 | Republican |
| 41 | George Bush               | 1989-1993 | Republican |
| 42 | William Jefferson Clinton | 1993-2001 | Democratic |
| 43 | George W. Bush            | Present   | Republican |

Abbildung 249: Mit Hilfe einer XSL-Transformation erzeugte Tabelle

Fehler, die bei der Ausführung der Transformation bzw. beim Laden der Transformationsdatei festgestellt werden, lösen eine Exception aus, die abgefangen werden sollte. Unter Umständen ist es sinnvoll, die XML-Datei wie oben gezeigt vorher zu validieren.



## 275 XSL-Transformation mit Parametern

Im vorangegangenen Beispiel wird auf die eigentliche Transformation kein Einfluss genommen. Die gesamte Abbildung ist in der XSLT-Datei festgeschrieben. Oft muss man jedoch der Transformation zusätzliche Parameter mit auf den Weg geben. Sei es, dass diese für zusätzliche Ausgaben benötigt werden oder dass sie zur Steuerung des Transformationsablaufs Verwendung finden sollen.

Eine mögliche Aufrufvariante für eine XSL-Transformation mit Parametern zeigt Listing 496. Hier wird die XML-Datei über eine `XPathDocument`-Instanz geladen. Das Anlegen des `XsltTransform`-Objektes und das Laden der XSLT-Datei erfolgen wie zuvor.

Zur späteren Übergabe der Parameter wird eine Instanz von `XsltArgumentList` angelegt. Mit `AddParam` wird ein Parameter festgelegt. Hier wird unter dem Parameternamen `Party` der in einer `ComboBox` ausgewählte Parteiname (`Democratic` oder `Republican`) hinzugefügt. Diese Parameterliste wird beim Aufruf von `Transform` zusätzlich übergeben.

```
' XML-Datei laden und Navigator erzeugen
Dim xPDoc As New XPathDocument( _
    Application.StartupPath & "..\Presidents.xml")
Dim xPNav As XPathNavigator = xPDoc.CreateNavigator()

' Neues Transformationsobjekt
Dim xslt As New XsltTransform

' Neue Parameterliste
Dim params As New XsltArgumentList

' Writer für Zieldatei
Dim writer As New XmlTextWriter("Presidents2.html", _
    System.Text.Encoding.Default)

' Ausgewählte Partei als Parameter hinzufügen
params.AddParam("Party", "", CBOParty.SelectedItem)

Try
    ' Transformationsdatei laden
    xslt.Load(Application.StartupPath & "..\Presidents2.xslt")

    ' Transformation ausführen
    xslt.Transform(xPDoc, params, writer, New XmlUrlResolver)

Catch ex As Exception
    MessageBox.Show (ex.Message)
End Try

' Zieldatei schließen
writer.Close()
```

Listing 496: XSL-Transformation mit Parameterübergabe

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

Im Stylesheet müssen die Parameter definiert werden. Das geschieht mit dem Tag `<xsl:param>`. Unter dem dort angegebenen Namen kann dann im Dokument mit einem vorangestellten `$`-Zeichen auf den Parameter zugegriffen werden (z.B. `$Party`). Listing 497 zeigt die erforderlichen Änderungen der XSLT-Datei.

Mit einem `<xsl:if>`-Knoten wird die Ausgabe gesteuert. Nur die Daten der Präsidenten, die der vorgegebenen Partei angehören, werden in die Liste aufgenommen. Das Ergebnis der Transformation sehen Sie in Abbildung 250.

---

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:pres="http://tempuri.org/presidents.xsd">

  <!-- Parameter zur Auswahl der Partei -->
  <xsl:param name="Party"/>

  <!-- Match für Stammknoten -->
  <xsl:template match="/">
    ...
  </xsl:template>

  <!-- Match für Präsidenten-Knoten -->
  <xsl:template match="pres:President">

    <!-- Nur Präsidenten mit der vorgegebenen Parteizugehörigkeit
         berücksichtigen -->
    <xsl:If test="pres:Party=$Party">

      <!-- Tabellenzeile zusammensetzen -->
      <TR style="FONT-SIZE: 10pt; FONT-FAMILY: Arial">
        <TD align="center"><xsl:value-of Select="@ID"/></TD>
        <TD ><xsl:value-of Select="pres:Name"/></TD>
        <TD><xsl:value-of Select="pres:YearsServed"/></TD>
        <TD><xsl:value-of Select="pres:Party"/></TD>
      </TR>

    </xsl:If>
  </xsl:template>

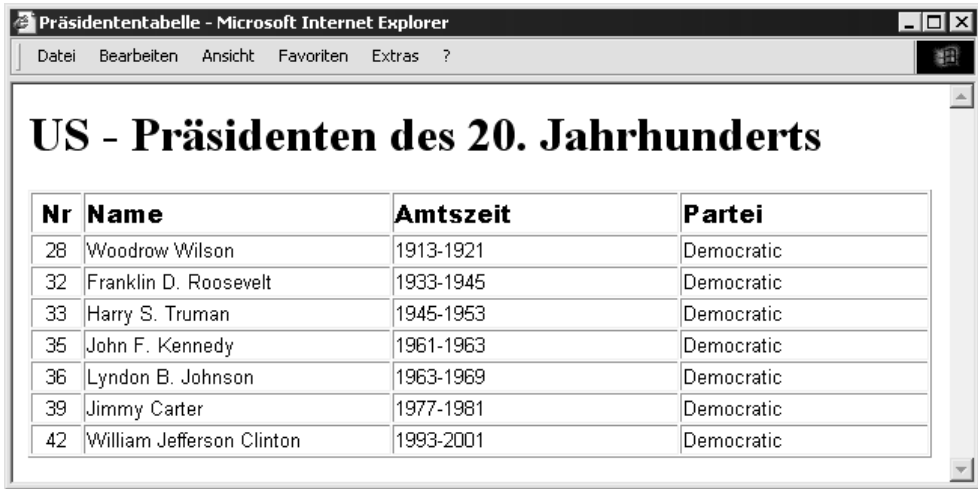
</xsl:stylesheet>
```

---

*Listing 497: Auswertung eines übergebenen Parameters im Stylesheet*

## 276 Einer XSL-Transformation zusätzliche Funktionen bereitstellen

Neben den XPath-Funktionen können Sie in einer XSLT-Datei eine Reihe von Funktionen verwenden, die XSL von Haus aus mitliefert. Viele Operationen lassen sich aber mit den zur Verfügung stehenden Funktionen nicht oder nur sehr umständlich durchführen. Beispiele hierfür sind Berechnungen von Bildgrößen oder Kalenderinformationen.



| Nr | Name                      | Amtszeit  | Partei     |
|----|---------------------------|-----------|------------|
| 28 | Woodrow Wilson            | 1913-1921 | Democratic |
| 32 | Franklin D. Roosevelt     | 1933-1945 | Democratic |
| 33 | Harry S. Truman           | 1945-1953 | Democratic |
| 35 | John F. Kennedy           | 1961-1963 | Democratic |
| 36 | Lyndon B. Johnson         | 1963-1969 | Democratic |
| 39 | Jimmy Carter              | 1977-1981 | Democratic |
| 42 | William Jefferson Clinton | 1993-2001 | Democratic |

Abbildung 250: Über einen Parameter wurde der Aufbau der Liste auf eine Partei eingeschränkt

.NET hält eine verblüffend einfache Lösung für dieses Problem bereit: Programmieren Sie die benötigten Funktionen doch in VB.NET!

Alles, was Sie hierfür benötigen, ist eine Klasse, die die gewünschten Funktionen als öffentliche Methoden bereitstellt. Innerhalb dieser Methoden haben Sie die volle .NET-Funktionalität zur Verfügung. Sie arbeiten mit der gewohnten Syntax und können die Funktionen auch noch debuggen – eine Möglichkeit, die bei Funktionen innerhalb des Stylesheets nicht gegeben ist.

Als Beispiel soll die Präsidententabelle um eine fünfte Spalte erweitert werden, in der der Name des Wochentags aufgeführt wird, an dem der jeweilige Präsident geboren wurde. Zur Bestimmung des Wochentags wird eine Methode im VB.NET-Programm bereitgestellt. Hierfür wird die benötigte Methode (`GetNameOfDay`) innerhalb einer neuen Klasse implementiert (Listing 498). Mit folgendem Code werden die öffentlichen Methoden der Klasse in der Transformation verfügbar gemacht:

```
' DayComputation-Objekt für Berechnungen übergeben
Dim dayComp As New DayComputation
params.AddExtensionObject("urn:DayComputation", dayComp)
```

`AddExtensionObject` übergibt die Referenz einer Instanz von `DayComputation` und definiert einen Namensraum für die Verwendung. Dieser Namensraum kann frei gewählt werden und dient dazu, Namensgleichheiten zu vermeiden. Üblich ist eine Notation wie `urn:Klassenname`.

```
Public Class DayComputation

    Public Function GetNameOfDay(ByVal day As String) As String

        ' In DateTime-Struktur umwandeln
        Dim d As DateTime = DateTime.Parse(day)
```

Listing 498: Die Klasse `DayComputation` stellt die Methode `GetNameOfDay` für den Gebrauch innerhalb einer XSL-Transformation zur Verfügung

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Propor-  
tGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

---

```
' Wochentag als Text zurückgeben
Return d.ToString("dddd")
```

```
End Function
```

```
End Class
```

---

*Listing 498: Die Klasse DayComputation stellt die Methode GetNameOfDay für den Gebrauch innerhalb einer XSL-Transformation zur Verfügung (Forts.)*

Das geänderte Stylesheet sehen Sie in Listing 499. Die Namensraumdefinition wird um einen Namespace für die verfügbaren Methoden erweitert (DayComputation), und zwar so, wie er bei AddExtensionObject übergeben wird. Der Klassenname wird hier als Prefix definiert. Um die Funktion aufzurufen, muss der Prefix angegeben werden (z.B. DayComputation:GetNameOfDay(pres:Birthday)).

Für jeden <President>-Knoten wird GetNameOfDay aufgerufen und das Geburtsdatum als String-Parameter übergeben. In der VB.NET-Methode wird der String umgewandelt, der Name des Wochentags bestimmt und als String zurückgegeben. Das Ergebnis wird als Wert in der Tabelle eingefügt. Abbildung 251 zeigt die HTML-Ansicht der erzeugten Datei.

---

```
<?xml version="1.0" encoding="UTF-8" ?>

<!-- Zusätzlich wird ein Namespace für die Funktionen definiert -->
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:pres="http://tempuri.org/presidents.xsd"
  xmlns:DayComputation="urn:DayComputation">

  <!-- Parameter zur Auswahl der Partei -->
  <xsl:param name="Party"/>

  <!-- Match für Stammknoten -->
  <xsl:template match="/">
    ...
  </xsl:template>

  <!-- Match für Präsidenten-Knoten -->
  <xsl:template match="pres:President">
    ...
    <!-- Tabellenzeile zusammensetzen -->
    <TR style="FONT-SIZE: 10pt; FONT-FAMILY: Arial">
      <TD align="center"><xsl:value-of Select="@ID"/></TD>
      <TD ><xsl:value-of Select="pres:Name"/></TD>
      <TD><xsl:value-of Select="pres:YearsServed"/></TD>
      <TD><xsl:value-of Select="pres:Party"/></TD>

      <!-- VB-Methode aufrufen, um den Wochentag abzufragen -->
      <TD><xsl:value-of Select=
        "DayComputation:GetNameOfDay(pres:Birthday)" /></TD>
```

---

*Listing 499: Nutzung von Methoden, die in VB.NET programmiert wurden, in einem Stylesheet*

```

</TR>
...
</xsl:template>

</xsl:stylesheet>

```

Listing 499: Nutzung von Methoden, die in VB.NET programmiert wurden, in einem Stylesheet (Forts.)

| Nr | Name                 | Amtszeit  | Partei     | Geboren an einem |
|----|----------------------|-----------|------------|------------------|
| 25 | William McKinley     | 1897-1901 | Republican | Sonntag          |
| 26 | Theodore Roosevelt   | 1901-1909 | Republican | Mittwoch         |
| 27 | William Howard Taft  | 1909-1913 | Republican | Dienstag         |
| 29 | Warren G. Harding    | 1921-1923 | Republican | Donnerstag       |
| 30 | John Calvin Coolidge | 1923-1929 | Republican | Donnerstag       |
| 31 | Herbert Hoover       | 1929-1933 | Republican | Montag           |
| 34 | Dwight D. Eisenhower | 1953-1961 | Republican | Dienstag         |
| 37 | Richard M. Nixon     | 1969-1974 | Republican | Donnerstag       |
| 38 | Gerald R. Ford       | 1974-1977 | Republican | Montag           |
| 40 | Ronald Reagan        | 1981-1989 | Republican | Montag           |
| 41 | George Bush          | 1989-1993 | Republican | Donnerstag       |
| 43 | George W. Bush       | Present   | Republican | Dienstag         |

Abbildung 251: Die Wochentage wurden durch Rückruf einer VB.NET-Methode ermittelt

## 277 Parallelbetrieb von XmlDocument und DataSet

Vorausgesetzt, für die zu bearbeitende XML-Datei existiert ein gültiges Schema und dieses Schema lässt eine Sicht der Daten in Form von Tabellen zu, dann können die Daten auch in ein DataSet geladen werden. Der Vorteil besteht darin, dass Sie mit den Daten in einer anderen Sichtweise umgehen können. Statt der Baumstruktur einer XML-Datei arbeiten Sie mit der Tabellenstruktur einer Datenbank. Somit stehen Ihnen alle Methoden der Klasse DataSet zur Verfügung, um Abfragen oder Manipulationen an den Daten vorzunehmen. Insbesondere wenn es um die Bearbeitung relationaler Strukturen geht, kann ein DataSet seine Vorteile ausspielen.

Der Clou besteht nun darin, dass sich die beiden Sichtweisen nicht gegenseitig ausschließen, sondern dass Ihnen das Framework einen Synchronisationsmechanismus zur Verfügung stellt, der es Ihnen erlaubt, wechselweise die XML-Baum-Ansicht und die Tabellen-Ansicht zu verwenden. Änderungen in der einen Sicht werden automatisch in der anderen nachgeführt. So können Sie beliebig zwischen Baum- und Tabellensicht hin- und herwechseln, je nachdem, welche für eine bestimmte Aufgabenstellung besser geeignet ist.

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung  
Win-  
dows  
Forms

Controls

Propor-  
tGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

Für diese parallele Bearbeitung sieht das Framework die Klasse `DataSet` für die Tabellensicht und die Klasse `XmlDataDocument` für die XML-Sicht vor. Um die Synchronisation einer `DataSet` und einer `XmlDataDocument`-Instanz sicherzustellen, stehen Ihnen zwei Vorgehensweisen zur Auswahl.

1. Erst das `DataSet`-Objekt anlegen und laden, dann mit diesem eine `XmlDataDocument`-Instanz generieren.
2. Erst das `XmlDataDocument`-Objekt anlegen und laden und anschließend mit diesem die `DataSet`-Instanz anlegen

Beide Wege führen zum gleichen Ziel. Betrachten wir zunächst die erste Variante. Basis für das Beispiel sind die bereits vorgestellten Dateien *Presidents.xml* (Listing 501) und *Presidents.xsd* (Listing 500). Um XML-Dateien in ein `DataSet`-Objekt zu laden, stehen Ihnen die Methoden `ReadXmlSchema` und `ReadXml` zur Verfügung. In jedem Fall muss das Schema geladen werden, damit die Zuordnung der Daten aus den XML-Elementen in die Tabellen des `DataSet`s vorgenommen werden kann. Danach kann die XML-Datei geladen werden:

---

```
' Anlegen einer leeren DataSet-Instanz
Dim dsDoc As New DataSet

' Lesen des XML-Schemas
dsDoc.ReadXmlSchema(Application.StartupPath & "..\Presidents.xsd")

' Laden der Daten in die Tabellenstruktur
dsDoc.ReadXml(Application.StartupPath & "..\Presidents.xml")

' Anlegen und synchronisieren einer XmlDataDocument-Instanz
Dim xdDoc As New XmlDataDocument(dsDoc)
```

---

Die Synchronisation einer neuen `XmlDataDocument`-Instanz wird vom Framework vorgenommen, wenn Sie dem Konstruktor die Referenz des `DataSet`s als Parameter übergeben. Über `dsDoc` und `xdDoc` stehen Ihnen dann die Tabellensicht bzw. die XML-Struktursicht zur Verfügung.

In der anderen Variante wird zuerst ein leeres `XmlDataDocument`-Objekt angelegt. Danach wird über die Eigenschaft `DataSet` das synchronisierte `DataSet`-Objekt abgerufen. Für dieses muss auch in dieser Variante das Schema der XML-Datei geladen werden. Erst dann kann die XML-Datei geladen werden.

---

```
' Leeres XmlDataDocument-Objekt anlegen
Dim xdDoc As New XmlDataDocument

' DataSet abrufen
Dim dsDoc As DataSet = xdDoc.DataSet

' Schema für die Zuordnung laden
dsDoc.ReadXmlSchema(Application.StartupPath & "..\Presidents.xsd")

' XML-Datei laden
xdDoc.Load(Application.StartupPath & "..\Presidents.xml")
```

---

Für das so erhaltene DataSet können Sie die enthaltenen Tabellen abfragen. Der Code

```
' Tabellen anzeigen
Debug.WriteLine("Tabellen des Datasets: ")
For Each dt As DataTable In dsDoc.Tables
    Debug.WriteLine(dt.TableName)
Next
```

führt zur Ausgabe von

```
Tabellen des Datasets:
President
```

Mit Hilfe der Select-Methode können Sie ein Array von DataRow-Objekten generieren, die einer angegebenen Bedingung genügen. Mit

```
' Alle Tabellenzeilen der "demokratischen" Präsidenten selektieren
Dim rows() As DataRow = dsDoc.Tables("President").Select( _
    "Party='Democratic'")

' Array durchlaufen und ID und Name des Präsidenten ausgeben
For Each row As DataRow In rows
    Debug.WriteLine(row("ID").ToString() & " " & _
        row("Name").ToString())
Next
```

werden alle Tabellenzeilen ausgewählt, für die in der Spalte Party der Wert Democratic eingetragen ist. Die Ausgabe sieht dann so aus:

```
28 Woodrow Wilson
32 Franklin D. Roosevelt
33 Harry S. Truman
35 John F. Kennedy
36 Lyndon B. Johnson
39 Jimmy Carter
42 William Jefferson Clinton
```

Exakt die gleiche Ausgabe können Sie auch über das XmlDocument-Objekt erreichen. Eine XPath-Abfrage liefert eine Liste vom Typ XmlNodeList. Diese wird durchlaufen und die Texte für das Attribut ID sowie für das Element Name abgerufen und ausgegeben:

```
' Namensraum verwalten
Dim nsMgr As New XmlNamespaceManager(xdDoc.NameTable)
nsMgr.AddNamespace("pres", "http://tempuri.org/presidents.xsd")

' XPath-Abfrage
Dim liste As XmlNodeList = xdDoc.SelectNodes( _
    "//pres:President[pres:Party='Democratic']", nsMgr)
```

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Proper-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

---

```
' Ergebnisliste durchlaufen und ID und Name des Präsidenten
' ausgeben
For Each nd As XmlNode In liste
    Debug.WriteLine(nd.Attributes("ID").Value & " " & _
        nd("Name").InnerText)
Next
```

---

Während in diesem Beispiel der Aufwand für beide Lösungen vergleichbar ist, gibt es Situationen, in denen Sie entweder über `XmlDataDocument` oder über `DataSet` wesentlich schneller zum Ziel kommen. Beide Sichten haben einen anderen Ursprung und verfolgen einen anderen Ansatz.

## 278 Klassenhierarchie aus XML-Schema erzeugen

Vielleicht kennen Sie bereits die Möglichkeit, mit Hilfe von Visual Studio aus einem Schema ein typisiertes Dataset zu generieren. Die Entwicklungsumgebung benutzt dazu ein kleines Helferlein namens `xsd.exe`. Mit Hilfe dieses Tools können Sie auch eine Klassenhierarchie erzeugen, die sich über den `XmlSerializer` serialisieren und deserialisieren lässt.

Wenn Sie die EINGABEAUFFORDERUNG über VISUAL STUDIO .NET 2003 COMMAND PROMPT starten, ist der Pfad für `xsd.exe` bereits verfügbar und Sie können das Programm direkt aufrufen:

---

```
xsd presidents.xsd /c /l:VB
```

---

Der Parameter `/c` weist `xsd.exe` an, aus der angegebenen XSD-Datei Klassen zu generieren, `/l:VB` gibt an, dass diese in VB-Syntax kodiert werden sollen. Die generierte Datei heißt in diesem Beispiel `presidents.vb` und hat den in Listing 500 dargestellten Aufbau (gekürzt und formatiert).

---

```
Imports System.Xml.Serialization

'
'Der Quellcode wurde von xsd automatisch generiert.
'Version=1.1.4322.573.
'

<System.Xml.Serialization.XmlTypeAttribute([Namespace]:= _
    "http://tempuri.org/presidents.xsd"), _
System.Xml.Serialization.XmlRootAttribute([Namespace]:= _
    "http://tempuri.org/presidents.xsd", IsNullable:=False)> _
Public Class Presidents

    <System.Xml.Serialization.XmlElementAttribute("President")> _
    Public Items() As PresidentsPresident
End Class

<System.Xml.Serialization.XmlTypeAttribute([Namespace]:= _
    "http://tempuri.org/presidents.xsd")> _
Public Class PresidentsPresident

    Public Name As String
```

---

Listing 500: Mit `xsd.exe` erzeugte Klassen auf Basis der Schemadatei `Presidents.xsd`



```

<System.Xml.Serialization.XmlElementAttribute(DataType="Date")> _
Public Birthday As Date

<System.Xml.Serialization.XmlIgnoreAttribute()> _
Public BirthdaySpecified As Boolean

Public Birthplace As String

Public YearsServed As String

Public Party As String

Public Terms As String

<System.Xml.Serialization.XmlAttributeAttribute( _
    DataType="Integer")> _
Public ID As String

End Class

```

*Listing 500: Mit xsd.exe erzeugte Klassen auf Basis der Schemadatei Presidents.xsd (Forts.)*

In den generierten Klassen sind bereits zahlreiche Attribute zur Steuerung der Serialisierung eingetragen. Mit `XmlAttributeAttribute` wird beispielsweise gesteuert, dass die Member-Variable `ID` als Attribut gespeichert wird. Auch die Namensräume werden berücksichtigt.

Interessant ist die Umsetzung des Elementes `Birthday`. Verwendet wird hier der Datentyp `Date` (identisch mit `DateTime`). Da es sich um einen Wertetyp handelt und somit `Birthday` nicht den Wert `Nothing` annehmen kann, wird ein zusätzlicher Mechanismus vorgesehen, um angeben zu können, ob das Element vorhanden ist oder nicht. Hierzu wird automatisch die Member-Variable `BirthdaySpecified` eingefügt. Sie ist mit einem `XmlIgnore`-Attribut gekennzeichnet, so dass sie bei der Serialisierung nicht berücksichtigt wird.

`xsd.exe` unterstützt noch eine Reihe weiterer Operationen, z.B. den umgekehrten Weg, nämlich die Generierung eines Schemas aus einer Klassenstruktur. Einzelheiten dazu finden Sie in der MSDN-Dokumentation.

## 279 Serialisierung mit Hilfe der Klasse XmlSerializer

Das Framework stellt mehrere Klassen zur Verfügung, die es erlauben, Objekte zu *serialisieren* und zu *deserialisieren*. Unter Serialisieren versteht man, den Inhalt eines Objektes in einen Byte-Strom (Stream) zu wandeln. Deserialisieren ist der umgekehrte Vorgang, der die Bytes aus einem Stream wieder zu einem Objekt zusammenfügt.

Eine der Klassen, die Objekte serialisieren können, ist `XmlSerializer`. Die `Serialize`-Methode dieser Klasse erzeugt eine XML-Struktur der Objektdaten und gibt diese auf einen Stream aus. Serialisiert werden alle öffentlichen Member-Variablen und alle öffentlichen Eigenschaften. Private und geschützte Member einer Klasse werden nicht berücksichtigt.

Da der Serialisierer die Klassen, deren Instanzen serialisiert werden sollen, nicht kennt, sondern via Reflection die Struktur der Klasse untersucht, müssen einige Voraussetzungen erfüllt sein, damit der `XmlSerializer` eine Klasse serialisieren und wieder deserialisieren kann.

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Proper-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

Jede Klasse, deren Objekte serialisiert werden sollen, muss einen Standard-(parameterlosen) Konstruktor besitzen. Bei der Deserialisierung wird unter Verwendung des Standard-Konstruktors eine Instanz der Klasse angelegt und anschließend den öffentlichen Membern Werte zugewiesen. Eine Reihe von Frameworkklassen (z.B. `Bitmap`, `Hashtable` etc. ) sind nicht serialisierbar. Die Typen aller zu serialisierenden Bestandteile einer Klasse müssen selbst auch serialisierbar sein.

Ist wenigstens eine der Bedingungen nicht erfüllt, führt der Versuch, mit `XmlSerializer` Objekte zu (de-)serialisieren, zu Laufzeitfehlern. In jedem Fall sollten die Serialisierungsoperationen in `Try/Catch`-Blöcken abgesichert werden.

Den typischen Ablauf einer Serialisierung sehen Sie in Listing 501. Im Beispiel wird die automatisch generierte Klasse `Presidents` (s. Listing 500) aus dem vorangegangenen Beispiel verwendet. Als Ausgabe-Stream wird zuerst eine Instanz der Klasse `StreamWriter` angelegt. Anschließend wird eine Instanz von `XmlSerializer` erzeugt. Dem Konstruktor wird dabei das Typ-Objekt des zu serialisierenden Typs übergeben. Die eigentliche Serialisierung erfolgt durch den Aufruf der Methode `Serialize`. Übergeben werden die Referenz des Streams und des zu serialisierenden Objektes. Da die Ausgabe in eine Datei erfolgt, muss der Stream geschlossen werden. Dies geschieht im Beispiel im `Finally`-Block, damit die Datei sowohl im Fehler- als auch im Erfolgsfall geschlossen wird.

---

```
Imports System.Xml
Imports System.Xml.Serialization
Imports System.IO

...

Protected USPresidents As Presidents

...

Dim sw As StreamWriter
Try

    ' StreamWriter für die Ausgabe mit XmlSerializer anlegen
    sw = New StreamWriter( _
        Application.StartupPath & "..\Presidents.xml")

    ' XmlSerializer-Instanz für Klasse Presidents
    Dim xs As New XmlSerializer(GetType(Presidents))

    ' Instanz von Presidents serialisieren
    xs.Serialize(sw, USPresidents)

Catch ex As Exception
    Debug.WriteLine(ex.Message)
Finally
    'StreamWriter wieder schließen
    If Not sw Is Nothing Then sw.Close()
End Try
```

---

Listing 501: Serialisieren der Präsidentenliste mit der `XmlSerializer`-Klasse

Das Gegenstück dazu, die Deserialisierung, sehen Sie in Listing 502. Analog zum `StreamWriter` aus Listing 501 wird hier ein `StreamReader` eingesetzt, da die Daten aus einer Datei gelesen werden sollen. Die Instanziierung des `XmlSerializer`-Objektes erfolgt genauso wie bei der Serialisierung. Wieder wird dem Konstruktor mitgegeben, um welchen Typ es sich handeln wird.

Nach Öffnen der Datei mit Hilfe des `StreamReaders` erfolgt die Deserialisierung durch den Aufruf der Methode `Deserialize`. Diese instanziert ein neues Objekt von der Klasse `Presidents`, überträgt die Werte und gibt die Referenz des Objektes zurück. Aus syntaktischen Gründen ist hier eine Typ-Umwandlung erforderlich.

```
' XmlSerializer-Instanz für Typ Presidents anlegen
Dim xs As New XmlSerializer(GetType(Presidents))
Dim sr As StreamReader
Try
    ' Streamreader-Instanz zum Lesen der Datei anlegen
    sr = New StreamReader( _
        Application.StartupPath & "..\Presidents.xml")

    ' Deserialisieren
    USPresidents = DirectCast(xs.Deserialize(sr), Presidents)

Catch ex As Exception
    Debug.WriteLine(ex.Message)

Finally
    ' Streamreader wieder schließen
    If Not sr Is Nothing Then sr.Close()
End Try
```

Listing 502: Deserialisieren der Präsidentenliste mit der `XmlSerializer`-Klasse

Prinzipiell können Sie jede Ihrer eigenen Klassen serialisieren, vorausgesetzt, sie erfüllt die oben genannten Bedingungen. Über Attribute lässt sich die Serialisierung im Detail steuern. Nachfolgend werden die wichtigsten Attribute beschrieben.

### XmlIgnore

Member-Variablen oder Eigenschaften, die mit diesem Attribut gekennzeichnet werden, werden von der Serialisierung ausgeschlossen. Sie können dieses Attribut verwenden, um gezielt Elemente nicht zu serialisieren, die aus anderen Gründen öffentlich sein müssen. Beispiel:

```
<XmlIgnore(>> Public Password As String
```

Die Membervariable `Password` ist öffentlich, wird aber nicht in die Serialisierung einbezogen.

### XmlRoot

Mit diesem Attribut können Sie Name und Namensraum des Stammelementes vorgeben. Beispiel:

```
XmlRoot("Personendaten", Namespace:="urn:PersonalData")> _
Public Class Person
...
End Class
```

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Proper-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

beginnt eine XML-Datei mit

---

```
<?xml version="1.0" encoding="utf-8"?>
<Personendaten xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance" xmlns="urn:PersonalData">
...
```

---

## XmlElement

Mit `XmlElement` können Sie den Namen und den Namensraum eines Elementes bei der Serialisierung vorgeben. Die öffentliche Member-Variable `Number`

---

```
<XmlElement("Nummer")> Public Number As Integer
```

---

wird in der XML-Darstellung durch das Element `Nummer` repräsentiert:

---

```
<Nummer>123</Nummer>
```

---

## XmlAttribute

Wenn ein Wert nicht als XML-Element, sondern als Attribut serialisiert werden soll, verwenden Sie `XmlAttribute`:

---

```
<XmlAttributeAttribute("Nummer")> Public Number As Integer
```

---

führt öffentliche Member-Variable `Number` als Attribut `Nummer` im Stammelement auf:

---

```
<Personendaten xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance" Nummer="123">
```

---

## XmlEnum

`XmlEnum` können Sie verwenden, um andere Bezeichnungen für Enum-Konstanten zu definieren. Beispiel:

---

```
Public Enum SpecialColor
    <XmlEnum("Rot")> Red
    <XmlEnum("Gelb")> Yellow
    <XmlEnum("Grün")> Green
End Enum

...

<XmlElement("Ampel")> Public Trafficlight As SpecialColor = _
    SpecialColor.Red
```

---

wird serialisiert zu:

---

```
<Ampel>Rot</Ampel>
```

---

## XmlText

Jeweils ein Element einer Klasse kann als Textknoten definiert werden.

```
<XmlText()> Public NickName As String = "Teddy"
```

wird z. B. so serialisiert:

```
<Personendaten xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance">
  Teddy
</Personendaten>
```

## XmlArray und XmlElement

Arrays und Auflistungen, sofern sie nicht `IDictionary` implementieren, werden automatisch mit- samt ihren Elementen serialisiert. Sie können mit den beiden Attributen steuern, wie die Abbil- dung erfolgen soll. Im Beispiel wird eine Liste einmal mit dem Attribut `XmlArray`, das andere Mal mit dem Attribut `XmlElement` markiert:

```
<XmlArray("ListeAlsArray")> Public Liste As New ArrayList
<XmlElement("ListeAlsElement")> Public Liste2 As ArrayList

...

Liste.Add("Erstes Element")
Liste.Add("Zweites Element")
Liste.Add("Drittes Element")
Liste2 = Liste
```

Beide Referenzvariablen verweisen auf dieselbe Liste. Die Serialisierung ergibt:

```
<ListeAlsArray>
  <anyType xsi:type="xsd:String">Erstes Element</anyType>
  <anyType xsi:type="xsd:String">Zweites Element</anyType>
  <anyType xsi:type="xsd:String">Drittes Element</anyType>
</ListeAlsArray>

<ListeAlsElement xsi:type="xsd:String">Erstes
  Element</ListeAlsElement>
<ListeAlsElement xsi:type="xsd:String">Zweites
  Element</ListeAlsElement>
<ListeAlsElement xsi:type="xsd:String">Drittes
  Element</ListeAlsElement>
```

Im ersten Fall werden alle Elemente als Kind-Elemente von `ListeAlsArray` gekapselt, im zweiten Fall wird für jedes Array-Element ein XML-Element `ListeAlsElement` angelegt.

## XmlInclude

Findet der XmlSerializer beim Serialisieren statt eines Objektes vom Typ A ein Objekt vom Typ B, wobei B von A mittelbar oder unmittelbar abgeleitet ist, dann wird ein Laufzeitfehler ausgelöst. Beispiel: Gegeben seien die Klassen `Address` und `OfficeAddress`:

---

```
Public Class Address

    Public City As String

    Public Sub New(ByVal city As String)
        Me.City = city
    End Sub
    Public Sub New()
    End Sub

End Class

Public Class Officeaddress
    Inherits Address

    Public Company As String

    Public Sub New()
    End Sub
    Public Sub New(ByVal company As String, ByVal city As String)
        MyBase.New(city)
        Me.Company = company
    End Sub

End Class
```

---

Die Klasse `Person` definiert eine Member-Variable vom Typ `Address`:

---

```
Public Class Person
    ...
    Public Address1, Address2 As Address
    ...
End Class
```

---

und weist ihr eine Instanz von `Officeaddress` zu:

---

```
Address1 = New Officeaddress("Microsoft", "Redmont")
```

---

Die syntaktisch und semantisch richtige Zuweisung kann der XmlSerializer nicht auflösen. Damit die Serialisierung möglich ist, müssen bei der Definition der übergeordneten Klasse alle Klassen, die für die Serialisierung in Frage kommen, bekannt gemacht werden:

---

```
<XmlInclude(GetType(Officeaddress))> _
Public Class Person
    ...
End Class
```

---

Falls Sie Namensräume verwenden, achten Sie darauf, dass die Definitionen zusammenpassen. Die Serialisierung ergibt z. B. folgende XML-Struktur:

```
<Person ...>
  <Address1 xsi:type="Officeaddress">
    <City>Redmont</City>
    <Company>Microsoft</Company>
  </Address1>
  ...
</Person>
```

### Grenzen des XmlSerializer

Der XmlSerializer wurde konzipiert, um Daten in WebServices serialisieren zu können. Wie Sie bereits gesehen haben, werden nur öffentliche Daten serialisiert. Ebenfalls gibt es eine Reihe von zu erfüllenden Bedingungen und Einschränkungen. Mit dem XmlSerializer können Sie nur Daten, aber keine Referenzen serialisieren. Betrachten Sie folgendes Beispiel:

```
Public P1 As New Person
Public P2 As Person = P1
```

Vor der Serialisierung verweisen beide Referenzvariablen P1 und P2 auf dasselbe Objekt. Nach einer Deserialisierung wird jedoch sowohl für P1 als auch für P2 eine neue Instanz von Person angelegt. Die Werte sind zwar identisch, P1 und P2 verweisen aber dann auf unterschiedliche Objekte.

Wenn die korrekte Auflösung von Referenzen für Sie wichtig ist, dann müssen Sie auf einen anderen Serialisierer (z. B. SoapFormatter) ausweichen.

Auch die Serialisierung einiger Wertetypen ist problematisch. Während DateTime automatisch im ISO-Format ausgegeben wird, lässt sich ein TimeSpan-Wert nicht serialisieren. Wenn Sie im Zweifel sind, ob der XmlSerializer für Ihre Zwecke geeignet ist, prüfen Sie an einem Testprogramm, ob alle Ihre benötigten Datentypen richtig serialisiert und wieder deserialisiert werden.

## 280 Unbekannte XML-Inhalte bei der Deserialisierung mit dem XmlSerializer

Bei der Deserialisierung behandelt der XmlSerializer nur die Elemente, die er für die Wiederherstellung der Objekte benötigt. Existieren zusätzliche Knoten, z. B. zusätzliche Attribute oder untergeordnete Elemente, dann werden diese für die Deserialisierung nicht berücksichtigt.

Im Normalfall werden Sie die zusätzlichen Knoten ignorieren können. Sie können sich bei Bedarf aber vom XmlSerializer informieren lassen, wenn unbekannte Knoten gefunden werden. Hierzu stehen drei Ereignisse zur Verfügung, die Sie mit eigenen Handlern verknüpfen können:

- ▶ UnknownNode
- ▶ UnknownAttribute
- ▶ UnknownElement

Als Beispiel dient noch einmal die Präsidentenliste. Zusätzlich wurden ein Attribut und zwei Elemente zur Angabe der Haustiere eingeführt:

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tung  
Win-  
dows  
Forms

Controls

Propert-  
yGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

---

```
<President ID="42" NumberOfPets="2">
  <Name>William Jefferson Clinton</Name>
  <Birthday>1946-08-19</Birthday>
  <Birthplace>Hope, Arkansas</Birthplace>
  <YearsServed>1993-2001</YearsServed>
  <Party>Democratic</Party>
  <Terms>2</Terms>
  <Pet Name="Socks" Type="Cat"/>
  <Pet Name="Buddy" Type="Golden Retriever">xxx</Pet>
</President>
```

---

Vor der Deserialisierung werden zwei Ereignis-Handler (Listing 503) an die `XmlSerializer`-Instanz gebunden:

---

```
' XmlSerializer-Instanz für Typ Presidents anlegen
Dim xs As New XmlSerializer(GetType(Presidents))

' Handler für unbekannte Attribute und Elemente anbinden
AddHandler xs.UnknownAttribute, AddressOf UnknownAttribute
AddHandler xs.UnknownElement, AddressOf UnknownElement
```

---

Bei der Deserialisierung werden die Ereignis-Handler aufgerufen und erzeugen die folgende Ausgabe:

---

```
Unbekanntes Attribut: NumberOfPets (2)
Unbekanntes Element: Pet
Unbekanntes Element: Pet
((Beginn Listing))
Private Sub UnknownAttribute(ByVal sender As Object, _
    ByVal e As XmlAttributeEventArgs)

    Debug.WriteLine("Unbekanntes Attribut: " & e.Attr.Name & _
        " (" & e.Attr.Value & ")")

End Sub
Private Sub UnknownElement(ByVal sender As Object, _
    ByVal e As XmlElementEventArgs)

    Debug.WriteLine("Unbekanntes Element: " & e.Element.Name)

End Sub
```

---

*Listing 503: Reagieren auf unbekannte Attribute und Knoten beim Deserialisieren mit dem `XmlSerializer`*

## 281 Serialisierung mit Hilfe der Klasse `SoapFormatter`

`SoapFormatter` steht für Seifenformatierer, ein Werkzeug, mit dem man langweilige quaderförmige Seifen in niedliche kleine Pinguine, Enten usw. verwandeln kann (siehe Abbildung 252)... oh, falscher Film ;-)





Abbildung 252: Mit dem SoapFormatter erzeugte Seifenpinguine J

Also noch mal von vorn: *Soap* steht für Simple Object Access Protocol und ist, wie die Bezeichnung vermuten lässt, eher für die Datenübertragung als für die Speicherung in Dateien gedacht. Es dient als Grundlage für die Kommunikation mit Web-Diensten sowie für Kommunikationskanäle bei .NET Remoting. Grundsätzlich ist es ein XML-Format, das zum Serialisieren beliebiger Objekte verwendet werden kann.

Die Klasse, die die Serialisierung mit Soap unterstützt, heißt *SoapFormatter*. Ähnlich *XmlSerializer* implementiert sie die Methoden *Serialize* und *Deserialize*. Als Ziel bzw. Quelle werden hier allerdings keine *StreamReader*-oder *StreamWriter*-Objekte, sondern ein *Stream*-Objekt angegeben. Eine Serialisierung in eine Datei kann dann z. B. so aussehen:

```
Dim B As New Bank
...

' Instanz des SoapFormatters anlegen
Dim sf As New SoapFormatter

' FileStream für Dateiausgaben öffnen
Dim fs As New FileStream("Bank.xml", FileMode.Create)

' Serialisieren
sf.Serialize(fs, B)

' FileStream schließen
fs.Close()
```

Die Klasse *Bank* hat je eine Liste von Kunden und Konten. In Listing 504 sehen Sie einen Ausschnitt der Implementierung der verwendeten Klassen *Bank*, *Person* und *Account*. Abbildung 253 zeigt einen Ausschnitt der erzeugten XML-Datei, dargestellt im Internet Explorer.

Bei der Serialisierung der Objekte geht der *SoapFormatter* allerdings grundlegend anders vor als der *XmlSerializer*. Damit Objekte serialisiert werden können, müssen die betreffenden Klassen entsprechend vorbereitet werden. Grundsätzlich reicht es aus, sie mit dem Attribut *Serializable* zu kennzeichnen. Sie können aber auch selbst in die Serialisierung eingreifen und die Schnittstelle *ISerializable* implementieren oder die Klasse von speziellen Basisklassen (z. B. *MarshalByRefObject*) ableiten.

---

```

<Serializable(> _
Public Class Bank

    Public Customers As New ArrayList
    Public Accounts As New ArrayList

    Public Function AddCustomer ...
    Public Function AddAccount ...
End Class

<Serializable(> _
Public Class Person

    Public Accounts As New ArrayList
    Public Name As String

End Class

<Serializable(> _
Public Class Account

    Public Owner As Person
    Public ReadOnly Number As Integer

    Private accBalance As Double

    Public Property Balance() As Double ...

End Class

```

---

*Listing 504: Die Beispielklassen sind über Referenzen miteinander verknüpft*

Serialisiert werden nur Felder, also die Membervariablen eines Objektes. Eigenschaften werden nicht berücksichtigt. Im Gegensatz zum `XmlSerializer` werden nicht nur öffentliche, sondern auch private und geschützte Felder serialisiert. In der Abbildung sehen Sie z.B. einen Eintrag für das geschützte Feld `accBalance`.

Referenzen werden auch als solche behandelt. Ein Objektbaum kann beliebig tief geschachtelt und strukturiert sein. Auf der XML-Seite wird das durch Attribute wie `id="ref-7"` und `href="ref-14"` realisiert. Anders als beim `XmlSerializer` werden hier bei der Deserialisierung diese Referenzen wieder aufgelöst. Mehrfache Verweise auf ein Objekt führen daher nicht zur Objektvervielfältigung, wie es beim Aufruf von `XmlSerializer.Deserialize` der Fall ist.

Auch eine andere Einschränkung des `XmlSerializers` gibt es beim `SoapFormatter` nicht: den Zwang, einen Standardkonstruktor für jede serialisierbare Klasse bereitzustellen. `SoapFormatter.Deserialize` ruft die Methode `GetSafeUninitializedObject` der Klasse `FormatterServices` auf, um eine nicht initialisierte Instanz einer Klasse anzulegen. Hierbei wird kein Konstruktor aufgerufen. Die Initialisierung erfolgt sofort danach. Möglich ist das deswegen, weil ja auch alle privaten und geschützten Member-Variablen deserialisiert werden und ein Objekt somit nach der Deserialisierung den gleichen Zustand hat wie vor der Serialisierung.

Die Deserialisierung verläuft nach dem gleichen Muster wie die Serialisierung:

```
' Instanz des SoapFormatters anlegen
Dim sf As New SoapFormatter

' FileStream zum Lesen aus der Datei öffnen
Dim fs As New FileStream("Bank.xml", FileMode.Open)

' Deserialisieren
Dim B As Bank = DirectCast(sf.Deserialize(fs), Bank)

' FileStream schließen
fs.Close()
```

Wie das Beispiel zeigt, muss dem SoapFormatter nicht der zu behandelnde Typ bekannt gemacht werden. Bei der Serialisierung werden Informationen zu den verwendeten Typen abgespeichert, die bei der Deserialisierung wieder gelesen werden. Vermerkt werden hier insbesondere der Name und die Version der Assembly. So kann die Methode Deserialize eine Instanz vom Typ Bank anlegen, ohne die Klasse kennen zu müssen.

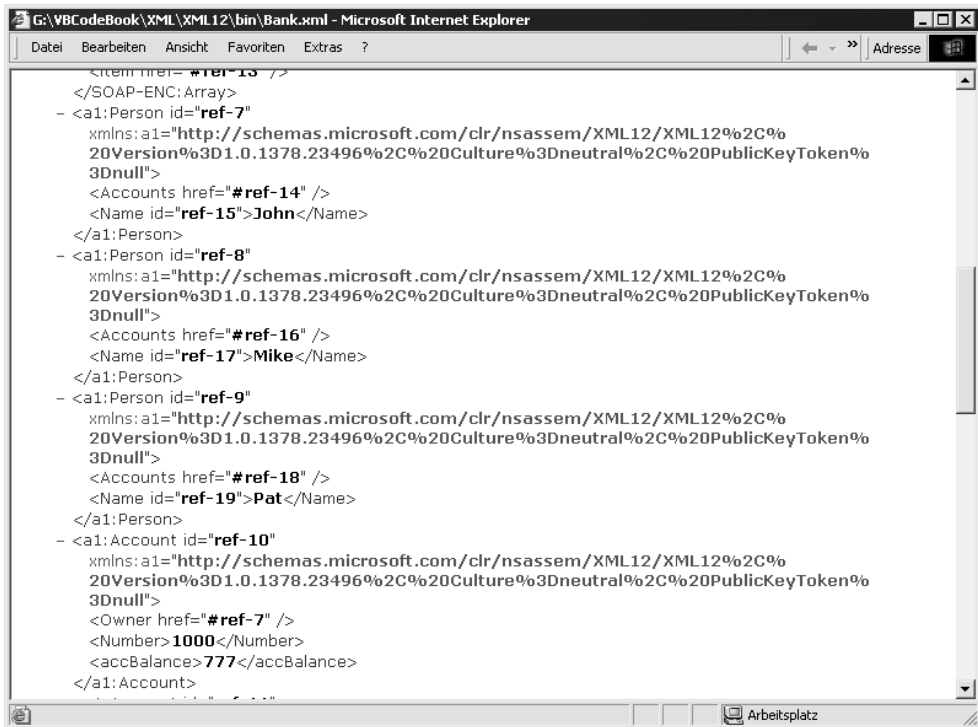


Abbildung 253: Eine mit dem SoapFormatter erzeugte XML-Datei im Internet Explorer

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Propor-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

**XML**

Wissen-  
schaft

Ver-  
schiede-  
nes



# Wissenschaftliche Berechnungen und Kurvendiagramme

Der Umgang mit physikalischen Größen in der Programmierung erfordert besondere Sorgfalt. Zu leicht werden Messwerte, Ergebnisse von Simulationen oder sonstige berechnete Daten in anonymen Wertetypen, meist Integer oder Double, gespeichert. Daraus resultieren oft vermeidbare Fehler, sei es bedingt durch den unsachgemäßen Umgang mit Gleitkommazahlen, durch Vermischung unterschiedlicher Einheiten oder die unzulässige Verknüpfung physikalischer Größen.

In diesem Kapitel werden einige Rezepte für den sicheren Umgang mit physikalisch technischen Größen vorgestellt. Abgerundet wird das Kapitel durch Rezepte für die Darstellung derartiger Größen in Kurvendiagrammen.

## 282 Gleitkommazahlen vergleichen

Nicht nur in wissenschaftlichen Anwendungen wird oft mit Gleitkommazahlen vom Typ Single oder Double gearbeitet. Die interne Zahlendarstellung besteht aus Mantisse und Exponent und ist so über viele Zehnerpotenzen hinweg einsetzbar. Bei Rechenoperationen ergeben sich aber zwangsweise Rundungsfehler, da die Mantisse auf drei bzw. sieben Byte begrenzt ist. Selbst wenn eine formatierte Ausgabe zweier Zahlenwerte gleich ist, heißt das nicht, dass der Vergleich auf Identität, also `Zahl1 == Zahl2`, `True` ergibt. Unabhängig von der Programmiersprache ist es deswegen ratsam, auf Vergleiche mit den Operatoren der Sprache zu verzichten.

Um zwei Gleitkommazahlen vergleichen zu können definiert man üblicherweise einen Toleranzbereich *Epsilon* für die Differenz der Zahlen. Wenn der Absolutwert der Differenz zweier Zahlen kleiner oder gleich diesem Epsilon ist, dann werden die Zahlen als gleich betrachtet:

---

wert1 ist gleich wert2, wenn  $| \text{wert1} - \text{wert2} | \leq \text{Epsilon}$

---

Werden derartige Vergleiche des Öfteren benötigt, bietet sich die Kapselung in einer Klasse an. Listing 505 zeigt die Implementierung einer solchen Klasse. `Epsilon` und die Vergleichsmethoden sind absichtlich nicht statisch (`Shared`) definiert, damit es möglich ist, mehrere Vergleiche mit unterschiedlichen Toleranzen zu instanzieren. Zahlenwerte um 10.000 wird man üblicherweise mit einer anderen Toleranz vergleichen wollen als Zahlenwerte um 0,001. Der Konstruktor der Klasse `DoubleComparer` nimmt den Toleranzwert `Epsilon` als Parameter entgegen.

---

```
Public Class DoubleComparer

    ' Erlaubter Differenzwert für Vergleich auf Identität
    Public Epsilon As Double

    ' Konstruktor mit vorgegebenem Epsilon
    Public Sub New(ByVal epsilon As Double)
        Me.Epsilon = epsilon
    End Sub
End Class
```

---

Listing 505: Klasse zum Vergleichen von Double-Zahlen

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Propor-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schie-  
denes

---

```
End Sub

' Vergleich auf Identität
Public Function IsEqual(ByVal value1 As Double, _
    ByVal value2 As Double) As Boolean

    Return Math.Abs(value1 - value2) <= Epsilon
End Function

' Vergleich auf Ungleichheit
Public Function IsNotEqual(ByVal value1 As Double, _
    ByVal value2 As Double) As Boolean

    Return Not IsEqual(value1, value2)
End Function

' Vergleich auf Null
Public Function IsZeroe(ByVal value As Double) As Boolean
    Return Math.Abs(value) <= Epsilon
End Function

' Vergleich auf ungleich Null
Public Function IsNotZeroe(ByVal value As Double) As Boolean
    Return Math.Abs(value) > Epsilon
End Function

' Größer gleich
Public Function IsGreaterOrEqual(ByVal value1 As Double, _
    ByVal value2 As Double) As Boolean

    Return IsEqual(value1, value2) OrElse (value1 > value2)
End Function

' Kleiner gleich
Public Function IsLessOrEqual(ByVal value1 As Double, _
    ByVal value2 As Double) As Boolean

    Return IsEqual(value1, value2) OrElse (value1 < value2)
End Function

' Größer
Public Function IsGreater(ByVal value1 As Double, _
    ByVal value2 As Double) As Boolean

    Return Not IsGreaterOrEqual(value1, value2)
End Function

' Kleiner
Public Function IsLess(ByVal value1 As Double, _
    ByVal value2 As Double) As Boolean

    Return Not IsLessOrEqual(value1, value2)
```

---

```
End Function

End Class
```

Listing 505: Klasse zum Vergleichen von Double-Zahlen (Forts.)

In der Klasse werden die Ersatzmethoden aller Vergleichsoperatoren zur Verfügung gestellt. Auch Operatoren wie >, >= etc. müssen umgesetzt werden.

Die Methoden lassen sich leicht anstelle der Vergleichsoperatoren nutzen:

```
Dim Comparer As New DoubleComparer(CDb1(NUDEpsilon.Value))
If (Comparer.IsEqual(Value1, 1.234)) Then ...
If (Comparer.IsZeroe(Value2)) Then ...
```

Auf der CD finden Sie das in Abbildung 254 dargestellte Beispiel zum Testen der Vergleichsfunktionen.

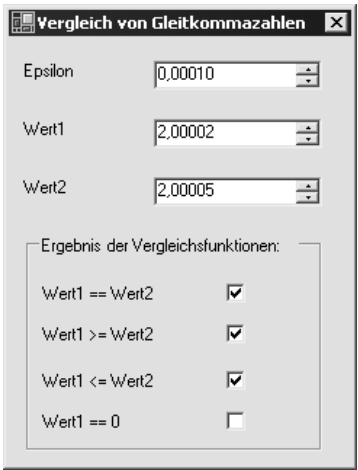


Abbildung 254: Testprogramm für die Klasse zum Vergleich von Gleitkommazahlen

Hinweis

Trotz der Hilfestellung durch die Klassenmethoden ist Vorsicht und Umsicht beim Vergleich von Gleitkommazahlen geboten. Wenn z.B. `IsEqual(A,B)` und `IsEqual(B,C)` jeweils `True` ergibt, folgt daraus nicht zwangsläufig, dass auch `IsEqual(A,C)` `True` zurückgibt. Es könnte sein, dass A geringfügig größer ist als B und B geringfügig größer als C, aber die Differenz von A und C außerhalb der Toleranz liegt.

## 283 Typsichere Maßeinheiten

Die Typsicherheit von Variablen wird in .NET besonders sorgfältig überprüft. Bei allen Operationen wird möglichst schon zur Compilezeit getestet, ob die verwendeten Datentypen auch wirklich miteinander verknüpft werden können. Zuweisungen, die z.B. zu Datenverlusten führen können (Double-Wert in Integer-Wert) werden zurückgewiesen wie Typumwandlungen, für die der Compiler keine Lösung kennt.

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Propor-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

In technisch-wissenschaftlichen Anwendungen werden oft physikalische Größen miteinander verknüpft. Diese Größen müssen auf die verfügbaren Datentypen abgebildet werden. Meist werden hierzu Ganzzahl- oder Gleitkomma-Variablen verwendet. Einmal als Zahlenwert abgelegt, kann der Compiler aber den eigentlichen Typ nicht mehr feststellen. Folgendes ist z.B. syntaktisch korrekt:

---

```
Dim Apples As Double = 5
Dim Pears As Double = 4
Dim Sum As Double = Apples + Pears
```

---

Aber was ist das Ergebnis? Multivitaminsaft? Oder Obstsalat? J Die Weisheit, dass man Äpfel und Birnen nicht addieren kann, ist noch nicht bis zum Compiler vorgedrungen. Beim Umgang mit Zahlenwerten geht für den Compiler der Typ verloren. Für ihn sind alle Größen vom Typ Double.

Bei physikalisch-technischen Größen kommt noch hinzu, dass ein und dieselbe Größe in unterschiedlichen Maßeinheiten definiert werden kann. Längenmaße können beispielsweise in Meter, Zentimeter, Zoll, Meilen usw. angegeben werden. Liegen die Maße in unterschiedlichen Einheiten vor, kommt es schnell zu Fehlern, wenn sie nicht sorgfältig umgerechnet werden.

Um dennoch sicher mit wissenschaftlichen Größen umgehen zu können, muss man selbst ein wenig Hand anlegen. Ausgehend von einer Basisklasse, die in diesem Rezept vorgestellt wird, muss für jede physikalische Größe eine eigene Klasse angelegt werden. In den folgenden Rezepten werden exemplarisch die Klassen für Längen- und Flächenmaße, für Zeit, Geschwindigkeit und für Temperaturen vorgestellt. Instanzen dieser Klassen können nicht miteinander vermischt werden. So wird es unmöglich, einer Funktion, die ein Längenmaß als Parameter erwartet, stattdessen eine Temperatur oder ein Flächenmaß zu übergeben. Die Typsicherheit kann vom Compiler gewährleistet werden.

Zusätzlich bieten die Klassen die Möglichkeit, die Werte in beliebigen Einheiten zu definieren bzw. abzufragen. Eine Variable vom Typ Flächenmaß kann z.B. mit einem Wert in Quadratmeter vorbelegt und ihr Inhalt später in Quadratzentimeter abgefragt werden. Intern rechnet jede Klasse mit einer Basiseinheit. Das ist sinnvollerweise eine *SI-Einheit*. Das Internationale Einheitensystem SI (Système International d'Unités) besteht aus sieben Basiseinheiten (Tabelle 41).

| Basisgröße              | Name      | Symbol |
|-------------------------|-----------|--------|
| Länge                   | Meter     | m      |
| Masse                   | Kilogramm | kg     |
| Zeit                    | Sekunde   | s      |
| elektrische Stromstärke | Ampere    | A      |
| Temperatur              | Kelvin    | K      |
| Stoffmenge              | Mol       | mol    |
| Lichtstärke             | Candela   | cd     |

Tabelle 41: Die sieben SI-Einheiten



Alle anderen Maßeinheiten können von diesen abgeleitet werden (z.B. Geschwindigkeit in m/s). In Deutschland wird zusätzlich noch unterschieden, ob es sich um gesetzliche Einheiten handelt oder nicht. Eine Übersicht über die Einheiten und ihre Definitionen stellt die Physikalisch-Technische Bundesanstalt (PTB) zur Verfügung:

[http://www.ptb.de/de/publikationen/\\_index.html](http://www.ptb.de/de/publikationen/_index.html)

Einmal angelegt können Instanzen einer Maßeinheitsklasse beliebig miteinander verrechnet werden, da sie die gleiche Basiseinheit benutzen. In C# können die Klassen auf nützliche Weise durch überladene Operatoren erweitert werden. So lassen sich beispielsweise die Addition und die Subtraktion für Längenmaße so definieren, dass der Compiler direkt eine Addition von Variablen dieses Typs zulässt. Eine Multiplikation von Längenmaßen kann derart programmiert werden, dass als Ergebnis ein Flächenmaß erzeugt wird. Leider bleibt in der VB.NET-Programmierung diese Möglichkeit versperrt. Für Rechenoperationen muss grundsätzlich auf den Zahlenwert zurückgegriffen werden.

### Basisklasse für Maßeinheiten

Aus verschiedenen Erwägungen erscheint es sinnvoll, eine Basisklasse für alle Klassen zu definieren, die Maßeinheiten darstellen sollen. So lassen sich z.B. leicht Funktionsparameter anlegen, die eine beliebige Maßeinheit annehmen. Eine Definition als Struktur kommt daher nicht in Betracht. Da später mit Referenzen auf Instanzen dieser Klassen gearbeitet wird, sollten die Objekte klonbar sein, also die Schnittstelle `ICloneable` implementieren. Listing 506 zeigt die grundlegende Definition der Basisklasse `MeasurementBase`.

```
Public MustInherit Class MeasurementBase
    Implements ICloneable

    ' Implementierung von ICloneable
    Public Function Clone() As Object Implements _
        System.ICloneable.Clone

        Return Me.MemberwiseClone()
    End Function

    ' Die Symbole der Einheiten
    Protected Symbols() As String

    ' Faktoren für Basiseinheit -> andere Einheit
    Protected Factors() As Double

    ' Interne Verrechnung in SI-Einheit
    Protected Value0 As Double
End Class
```

Listing 506: Die grundlegenden Elemente der Basisklasse `MeasurementBase`

Die Basisklasse selbst ist abstrakt, so dass sie nicht instanziiert werden kann. Die drei wesentlichen Variablen zeigt Tabelle 42.

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

| Membervariable | Bedeutung                                         |
|----------------|---------------------------------------------------|
| Symbols        | Array mit den Symbolen der verfügbaren Einheiten  |
| Factors        | Array mit Umrechnungsfaktoren für diese Einheiten |
| Value0         | Wert in der Basiseinheit                          |

Tabelle 42: Die wichtigen Membervariablen von *MeasurementBase*

Jede abgeleitete Klasse stellt ein statisches Array mit den Symbolen und eines mit den Umrechnungsfaktoren zur Verfügung. Das erste Element (Index 0) ist immer die Basiseinheit, der Faktor also 1. Der Basiswert wurde deswegen *Value0* genannt, um den Bezug auf Index 0 zu verdeutlichen.

Die Eigenschaft *Value* (s. Listing 507) greift auf den Basiswert zurück. So kann mit *EinWert.Value* gearbeitet werden, ohne dass man sich Gedanken über die ursprüngliche Einheit machen muss.

---

```

Public Overridable Property Value() As Double
    Get
        Return Value0
    End Get
    Set(ByVal Value As Double)
        Value0 = Value
    End Set
End Property

```

---

Listing 507: Die Eigenschaft *Value* erlaubt den Zugriff auf den Basiswert

Die *Value*-Eigenschaft wird überladen. Der Zugriff kann über einen Parameter gesteuert werden. Wahlweise kann die Einheit als Zahl oder als String übergeben werden. Für erstere Variante definieren die abgeleiteten Klassen Enumerationen, um die Zahlenwerte vor dem Anwender zu verbergen. Leider ist hier die Typsicherheit etwas eingeschränkt, da ein Enum hier nur als Integer-Wert angenommen werden kann. Alternativ können die Property-Funktionen in den abgeleiteten Klassen mit einem Enum als Parameter überschrieben werden.

---

```

' Zugriff auf den Basiswert und Umrechnung in angegebene Einheit
' Der Index wird in den abgeleiteten Klassen als Enum zur
' Verfügung gestellt
Public Overridable Property value(ByVal unitIndex As Integer) _
    As Double

    Get
        Return (Value0) * Factors(unitIndex)
    End Get
    Set(ByVal Value As Double)
        Value0 = Value / Factors(unitIndex)
    End Set
End Property

```

---

Listing 508: Überladungen der Eigenschaft *Value*

---

```

' Ermittlung des Index der als String übergebenen Einheit
Protected Function GetIndex(ByVal unit As String) As Integer
    Dim i As Integer
    For i = 0 To Symbols.GetUpperBound(0)
        If unit = Symbols(i) Then Return i
    Next
    Return 0
End Function

' Zugriff auf den Basiswert und Umrechnung in die als String
' übergebene Einheit
Public Overridable Property Value(ByVal unit As String) _
    As Double

    Get
        Dim i As Integer = GetIndex(unit)
        Return (Value0) * Factors(i)
    End Get
    Set(ByVal Value As Double)
        Dim i As Integer = GetIndex(unit)
        Value0 = Value / Factors(i)
    End Set
End Property

```

---

Listing 508: Überladungen der Eigenschaft Value (Forts.)

Die in Listing 507 und Listing 508 dargestellten Property-Funktionen sind geschützt und überschreibbar. So kann eine abgeleitete Klasse falls erforderlich die entsprechende Eigenschaft anders definieren. Nicht alle Umrechnungen lassen sich durch Multiplikation mit einem Faktor durchführen. Einige benötigen z.B. Offsets (Temperatur in °Celsius und °Fahrenheit). Um die Umrechnungen in der Basisklasse nicht zu überfrachten und die Rechenzeit gering zu halten, wird an dieser Stelle die spätere Überschreibung offen gehalten.

Ebenso überladen und überschreibbar sind die ToString-Funktionen. Die von Object überschriebene Methode ToString erzeugt einen Text, bestehend aus dem Zahlenwert und dem Symbol der Basiseinheit. Die überladene Variante erzeugt einen String aus dem in die angegebene Einheit umgerechneten Wert und dem zugehörigen Symbol. Listing 509 zeigt die beiden Methoden.

---

```

Public Overloads Overrides Function ToString() As String
    Return String.Format("{0} {1}", Value0, Symbols(0))
End Function

Public Overridable Overloads Function ToString _
    (ByVal unitIndex As Integer) As String

    Return String.Format("{0:f2} {1}", Value(unitIndex), _
        Symbols(unitIndex))

End Function

```

---

Listing 509: ToString-Überladung für die Textausgabe des Wertes

---

Basics

---

Datum/  
Zeit

---

Anwen-  
dungen

---

Zeich-  
nen

---

Bild-  
bearbei-  
tung  
Win-  
dows  
Forms

---

Controls

---

Propor-  
tyGrid

---

Datei-  
system

---

Netz-  
werk

---

System

---

Daten-  
banken

---

XML

---

Wissen-  
schaft

---

Ver-  
schiede-  
nes

Über die Funktion `GetUnitSymbols` lassen sich für jede Instanz einer Maßeinheits-Klasse die Symbole unterstützten Maßeinheiten ermitteln. Zur Sicherheit wird nicht die Referenz des statischen Arrays übergeben, sondern eine Kopie angelegt.

Um zur Laufzeit ermitteln zu können, welche Maßeinheit-Klassen insgesamt zur Verfügung stehen, wird in `MeasurementBase` die statische Funktion `GetAllMeasurementClasses` implementiert. Sie durchsucht die entsprechende Assembly nach Klassen, die mit dem Attribut `MeasurementAttribute` gekennzeichnet sind. Für jede gefundene Klasse wird eine Instanz von `MeasurementInfo` angelegt, die die Referenz des Typ- und des Attribut-Objektes festhält. Ein Array der gefundenen `MeasurementInfo`-Instanzen wird zurückgegeben. Listing 510 zeigt die beiden Hilfsfunktionen, Listing 511 die Klassen `MeasurementAttribute` und `MeasurementInfo`.

---

```

' Kopie des Symbol-Arrays bereitstellen
Public Function GetUnitSymbols() As String()
    Return DirectCast(Symbols.Clone(), String())
End Function

' Ermittlung aller Klassen, die für die Berechnung mit
' Maßeinheiten zur Verfügung stehen
Public Shared Function GetAllMeasurementClasses() _
    As MeasurementInfo()

    ' Assembly ermitteln
    Dim asmb As System.Reflection.Assembly = _
        System.Reflection.Assembly.GetAssembly( _
            GetType(MeasurementBase))

    ' Typen ermitteln
    Dim types() As Type = asmb.GetTypes()

    ' Liste der gesuchten Klassen
    Dim typelist As New ArrayList()

    ' Hilfsvariablen
    Dim t As Type
    Dim atts() As MeasurementAttribute

    ' Alle Typen dieses Assemblies untersuchen
    For Each t In types
        ' Feststellen, ob diesem Typ das MeasurementAttribute
        ' zugeordnet ist
        atts = DirectCast(t.GetCustomAttributes( _
            GetType(MeasurementAttribute), False), _
            MeasurementAttribute())

        If atts.Length > 0 Then
            ' Attribut gefunden, Typ in Liste aufnehmen
            typelist.Add(New MeasurementInfo(t, atts(0)))
        End If
    Next

    ' Rückgabe der Typliste als Array

```

---

---

```
Return DirectCast(typelist.ToArray( _
    GetType(MeasurementInfo)), MeasurementInfo())
```

---

```
End Function
```

---

*Listing 510: Abfrage von Detailinformationen zu den Klassen für Maßeinheiten (Forts.)*

---

```
' Information zu einer Maßeinheit-Klasse
Public Class MeasurementInfo
    ' Typ der Klasse
    Public Type As System.Type
    ' Beschreibungsattribut
    Public Attribute As MeasurementAttribute

    ' Konstruktor
    Public Sub New(ByVal type As System.Type, _
        ByVal attribute As MeasurementAttribute)

        Me.Type = type
        Me.Attribute = attribute

    End Sub

    ' ToString Überschreibung
    Public Overrides Function ToString() As String
        Return Me.Attribute.Name
    End Function
End Class

' Attribut zur Kennzeichnung von Maßeinheits-Klassen,
' die von MeasurementBase abgeleitet sind
<AttributeUsage(AttributeTargets.Class)> _
Public Class MeasurementAttribute

    Inherits Attribute

    ' Bezeichnung der physikalischen Größe
    Public Name As String

    ' Bezeichnung der Basiseinheit
    Public Baseunit As String

    Public Sub New(ByVal name As String, _
        ByVal baseunit As String)

        Me.Name = name
        Me.Baseunit = baseunit
    End Sub
End Class
```

---

*Listing 511: Hilfsklassen zur Bereitstellung näherer Informationen*

---

Basics

---

Datum/  
Zeit

---

Anwen-  
dungen

---

Zeich-  
nen

---

Bild-  
bearbei-  
tung

---

Win-  
dows  
Forms

---

Controls

---

Propert-  
yGrid

---

Datei-  
system

---

Netz-  
werk

---

System

---

Daten-  
banken

---

XML

---

Wissen-  
schaft

---

Ver-  
schiede-  
nes

---

Wie Sie in den folgenden Rezepten sehen werden, wird durch den aufwändigen Code der Basis-klasse die Implementierung der eigentlichen Maßeinheits-Klassen stark vereinfacht.

## 284 Definition von Längenmaßen

In wissenschaftlichen Berechnungen und Dokumentationen wird vorzugsweise mit dem metrischen System gearbeitet, das auch in Europa (außer auf den britischen Inseln) der Standard ist. In den USA sowie im Flug- und Schiffsverkehr kommen aber auch die alten britischen Maße zum Einsatz. Ferner gibt es noch Einheiten im Bereich der Drucktechnik (z.B. Punkt). Eine Klasse für Längenmaße sollte die gängigen Einheiten unterstützen. Listing 512 zeigt die Implementierung der Klasse `LinearMeasures`, die zehn der wichtigsten Einheiten berücksichtigt.

---

```

' Klasse für Längenmaße
' Basismaß ist Meter
<Measurement("Längen", "Meter")> _
Public Class LinearMeasures
    Inherits MeasurementBase

    ' Symbole der unterstützten Einheiten
    Protected Shared LengthSymbols() As String = _
        {"m", "cm", "mm", "km", "inch", "ft", "yard", _
         "mile", "nm", "pt"}

    ' Umrechnungsfaktoren
    Protected Shared LengthFactors() As Double = _
        {1, 100, 1000, 0.001, 100 / 2.54, 1 / 0.3048, 1 / 0.9144, _
         1 / 1609, 1 / 1852, 100 / 2.54 * 72}

    ' Indizes der unterstützten Einheiten als Enum
    Public Enum Units
        m
        cm
        mm
        km
        Inch
        Feet
        Yard
        Mile
        NauticalMile
        pt
    End Enum

    ' Standardkonstruktor, speichern der Array-Referenzen
    Public Sub New()
        Symbols = LengthSymbols
        Factors = LengthFactors
    End Sub

    ' Konstruktor mit Wert in Basiseinheit
    Public Sub New(ByVal value As Double)

```

---

Listing 512: `LinearMeasures`: Die Klasse für Längenmaße

```

        MyClass.New()
        Value0 = value
    End Sub

    ' Konstruktor mit Wert, dessen Einheitssymbol übergeben wird
    Public Sub New(ByVal value As Double, ByVal unit As String)
        MyClass.New()
        Me.Value(unit) = value
    End Sub

    ' Konstruktor mit Wert, dessen Einheits-Index übergeben wird
    Public Sub New(ByVal value As Double, ByVal unit As Units)
        MyClass.New()
        Me.Value(unit) = value
    End Sub

End Class

```

Listing 512: LinearMeasures: Die Klasse für Längenmaße (Forts.)

Exemplarisch für andere Ableitungen von `MeasurementBase` soll hier der Aufbau der Klasse `LinearMeasures` kurz beschrieben werden.

Der Klasse wird ein Attribut vom Typ `MeasurementAttribute` zugeordnet, das für den Zugriff via Reflection die Bezeichnung der physikalischen Größe und ihrer Basiseinheit bereithält.

Zwei statische Arrays mit den Symbolen und den Umrechnungsfaktoren werden für die Klasse definiert. Die Referenzen werden in den Konstruktoren den Membervariablen `Symbols` und `Factors` zugewiesen.

Eine Enumeration namens `Units` dient zur Definition der Indices. Die Konstruktoren werden überladen, um auch Werte unter Angabe der Einheit oder ihres Indexes annehmen und umrechnen zu können.

Nachfolgend sehen Sie einige Beispiele für die Anwendung der Klasse:

```

' Definitionen
Dim L1 As New LinearMeasures()           ' 0 m
Dim L2 As New LinearMeasures(5)         ' 5 m
Dim L3 As New LinearMeasures(10, "inch") ' 10 Zoll
' 3 Meilen
Dim L4 As New LinearMeasures(3, LinearMeasures.Units.Mile)
Dim L5 As New LinearMeasures()
Dim L6 As New LinearMeasures()

' Zuweisungen
L5.Value = 100                          ' 100 m
L6.Value(LinearMeasures.Units.Inch) = 10 ' 10 Zoll

' Ausgaben
' In m
Debug.WriteLine(L1)
' In cm
Debug.WriteLine(L2.Value("cm"))

```

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Proper-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

---

```
' nur der Zahlenwert
Debug.WriteLine(L3.Value)
' In km
Debug.WriteLine(L4.ToString(LinearMeasures.Units.km))
' In m
Debug.WriteLine(L5)
' In Punkt
Debug.WriteLine(L6.ToString(LinearMeasures.Units.pt))
```

---

Im Umgang mit den Werten können Sie jederzeit, d.h. beim Anlegen, bei Zuweisungen, beim Lesen und Formatieren die Einheit frei wählen. Die Umrechnung erfolgt automatisch, der in `Value0` gespeicherte Basiswert ist immer die Länge in Meter. Der Code-Ausschnitt erzeugt die folgende Ausgabe:

---

```
0 m
500
0,254
4,83 km
100 m
720,00 pt
```

---

Sie können diese Klasse bzw. die Basisklasse noch durch sinnvolle Funktionen erweitern. Da VB.NET keine überladbaren Operatoren unterstützt, können Sie stattdessen Methoden wie `Add`, `Subtract` oder `Multiply` implementieren, die als Ergebnis ein neues `LinearMeasures` bzw. `SurfaceMeasures`-Objekt zurückgeben.

## 285 Definition von Flächenmaßen

Analog zur Klasse `LinearMeasures` stellt die Klasse `SurfaceMeasures` alles bereit, um mit Flächenmaßen zu arbeiten. Listing 513 zeigt die Implementierung.

---

```
' Basismaß ist Quadratmeter
<Measurement("Fläche", "Quadratmeter")> _
Public Class SurfaceMeasures
    Inherits MeasurementBase

    ' Unterstützte Einheiten
    Protected Shared SurfaceSymbols() As String = _
        {"m²", "cm²", "mm²", "km²", "square inch", _
         "square foot", "square yard", "acre", "square mile"}

    ' Umrechnungsfaktoren
    Protected Shared SurfaceFactors() As Double = _
        {1, 10000, 1000000, 0.000001, 10000 / 6.452, 10000 / 929.029, _
         10000 / 8361.26, 1 / 4046.8, 1 / 2590000}

    Public Enum Units
        qm
        qcm
```

---

Listing 513: Klasse `SurfaceMeasures` für den Umgang mit Flächenmaßen



```

    qmm
    qkm
    SquareInch
    SquareFoot
    SquareYard
    Acre
    SquareMile
End Enum

Public Sub New()
    Symbols = SurfaceSymbols
    Factors = SurfaceFactors
End Sub

Public Sub New(ByVal value As Double)
    MyClass.New()
    Value0 = value
End Sub

Public Sub New(ByVal value As Double, ByVal unit As String)
    MyClass.New()
    Me.Value(unit) = value
End Sub

Public Sub New(ByVal value As Double, ByVal unit As Units)
    MyClass.New()
    Me.Value(unit) = value
End Sub

' Flächenmaß aus zwei Längenmaßen bilden
Public Sub New(ByVal length As LinearMeasures, _
    ByVal width As LinearMeasures)

    MyClass.New()
    value0 = length.Value * width.Value
End Sub

End Class

```

Listing 513: Klasse *SurfaceMeasures* für den Umgang mit Flächenmaßen (Forts.)

Der letzte Konstruktor im Listing erlaubt es, aus zwei Längenmaßen durch Multiplikation ein Flächenmaß zu bilden. Nachfolgend ein paar Code-Beispiele für die Anwendung der Klasse:

```

' Definitionen
Dim F1 As New SurfaceMeasures()
Dim F2 As New SurfaceMeasures(100)
Dim F3 As New SurfaceMeasures(1, _
    SurfaceMeasures.Units.SquareMile)
Dim F4 As New SurfaceMeasures(10, "cm²")
Dim L1 As New LinearMeasures(5) ' 5 m
Dim L2 As New LinearMeasures(10, "inch") ' 10 Zoll

```

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Proper-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

---

```
Dim F5 As New SurfaceMeasures(L1, L2)      ' 5 m x 10 Zoll

Debug.WriteLine(F1)
Debug.WriteLine(F2.ToString(SurfaceMeasures.Units.qcm))
Debug.WriteLine(F3.ToString(SurfaceMeasures.Units.qkm))
Debug.WriteLine(F4.ToString(SurfaceMeasures.Units.SquareInch))
Debug.WriteLine(F5)
```

---

Der Code erzeugt die folgende Ausgabe:

---

```
0m²
1000000,00 cm²
2,59 km²
1,55 square inch
1,27 m²
```

---

Aufbauend auf diesem Beispiel lässt sich leicht eine Klasse für Volumenmaße definieren. Auch hier könnten Sie wieder Konstruktoren überladen, die Längen- und Flächenmaße als Parameter annehmen.

## 286 Definition von Geschwindigkeiten

Den Umgang mit Geschwindigkeiten erlaubt die Klasse `SpeedMeasures`. Listing 514 zeigt die Implementierung.

---

```
' Geschwindigkeit in Meter / Sekunde
<Measurement("Geschwindigkeit", "Meter pro Sekunde")> _
Public Class SpeedMeasures
    Inherits MeasurementBase

    ' Unterstützte Einheiten
    Protected Shared SpeedSymbols() As String = _
        {"m/s", "km/h", "miles/h"}

    ' Umrechnungsfaktoren
    Protected Shared SpeedFactors() As Double = _
        {1, 3.6, 3.6 / 1.609}

    Public Enum Units
        MeterProSekunde
        KilometerProStunde
        MeilenProStunde
    End Enum

    Public Sub New()
        Symbols = SpeedSymbols
        Factors = SpeedFactors
    End Sub
```

---

Listing 514: `SpeedMeasures` erlaubt den Umgang mit Geschwindigkeiten

```

Public Sub New(ByVal value As Double)
    MyClass.New()
    Value0 = value
End Sub

Public Sub New(ByVal value As Double, ByVal unit As String)
    MyClass.New()
    Me.Value(unit) = value
End Sub

Public Sub New(ByVal value As Double, ByVal unit As Units)
    MyClass.New()
    Me.Value(unit) = value
End Sub

' Definition der Geschwindigkeit aus Längenmaß und Zeitangabe
Public Sub New(ByVal distance As LinearMeasures, _
    ByVal time As TimeMeasures)

    MyClass.New()
    value0 = distance.Value / time.Value
End Sub

End Class

```

*Listing 514: SpeedMeasures erlaubt den Umgang mit Geschwindigkeiten (Forts.)*

Auch hier sehen Sie wieder einen speziellen Konstruktor, der die Definition der Geschwindigkeit aus einem Längenmaß und einer Zeitangabe errechnet. Auch hierzu ein paar Beispiele:

```

Dim S1 As New SpeedMeasures(10)
Dim S2 As New SpeedMeasures(100, _
    SpeedMeasures.Units.KilometerProStunde)

Dim S3 As New SpeedMeasures( _
    New LinearMeasures(100), New TimeMeasures(5))

Debug.WriteLine(S1.ToString( _
    SpeedMeasures.Units.KilometerProStunde))
Debug.WriteLine(S2.ToString( _
    SpeedMeasures.Units.MeilenProStunde))
Debug.WriteLine(S3)

```

Erzeugte Ausgabe:

```

36,00 km/h
62,15 miles/h
20 m/s

```

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schie-  
des

## 287 Definition von Zeiten

Die `DateTime` und die `TimeSpan`-Strukturen sind für wissenschaftliche Berechnungen, z.B. zur Ermittlung von Beschleunigungen und Geschwindigkeit, nicht sonderlich geeignet. Benötigt werden meist relative Zeitangaben, die als Gleitkommazahl ausgedrückt werden können. In der Klasse `TimeMeasures` erfolgt die Definition der Zeit daher analog zu den bereits beschriebenen Klassen. Listing 515 zeigt die Implementierung.

---

```

' Zeit in Sekunden
<Measurement("Zeit", "Sekunde")> _
Public Class TimeMeasures
    Inherits MeasurementBase

    ' Symbole der unterstützten Einheiten
    Protected Shared TimeSymbols() As String = _
        {"s", "ms", "min", "h"}

    ' Umrechnungsfaktoren
    Protected Shared TimeFactors() As Double = _
        {1, 1000, 1 / 60, 1 / 3600}

    ' Statische Variable für Basiszeit
    Public Shared BaseTime As DateTime = DateTime.Now

    Public Enum Units
        s
        ms
        min
        h
    End Enum

    ' Zeit ist Differenz zwischen "Now" und BaseTime
    Public Sub New()
        Symbols = TimeSymbols
        Factors = TimeFactors
        value0 = DateTime.Now.Subtract(BaseTime).TotalSeconds
    End Sub

    Public Sub New(ByVal value As Double)
        MyClass.New()
        Value0 = value
    End Sub

    Public Sub New(ByVal value As Double, ByVal unit As String)
        MyClass.New()
        Me.Value(unit) = value
    End Sub

    Public Sub New(ByVal value As Double, ByVal unit As Units)
        MyClass.New()
        Me.Value(unit) = value

```

---

Listing 515: `TimeMeasures` kapselt Zeitangaben

```

End Sub

' Wert aus DateTime-Struktur auf BaseTime beziehen
Public Sub New(ByVal time As DateTime)
    MyClass.New()
    value0 = time.Subtract(BaseTime).TotalSeconds
End Sub

End Class

```

Listing 515: TimeMeasures kapselt Zeitangaben (Forts.)

Damit zur Laufzeit dynamisch Zeitangaben aus `DateTime`-Strukturen verwendet werden können, bietet die Klasse einen Konstruktor an, der einen `DateTime`-Parameter auf eine Basiszeit bezieht und in Sekunden umrechnet. Die Basiszeit ist als statische Variable `BaseTime` definiert und wird automatisch bei der ersten Benutzung der Klasse gesetzt. Sie ist allerdings öffentlich, so dass sie auch per Programm auf einen gewünschten Wert gesetzt werden kann. Es ist allerdings zu berücksichtigen, dass die Basiszeit für alle `TimeMeasures`-Objekte gleichermaßen gilt.

Der folgende Beispiel-Code

```

Dim T1 As New TimeMeasures()
Dim T2 As New TimeMeasures(10)
Dim T3 As New TimeMeasures(2, TimeMeasures.Units.h)
Dim T4 As New TimeMeasures(New DateTime(2004, 1, 1))

Debug.WriteLine(T1)
Debug.WriteLine(T2.ToString(TimeMeasures.Units.min))
Debug.WriteLine(T3.ToString())
Debug.WriteLine(T4)

```

erzeugt eine Ausgabe wie diese:

```

10,9457392 s
0,17 min
7200 s
14521269,4288928 s

```

Der Standardkonstruktor setzt die Zeit nicht zu Null, sondern bildet die Differenz der aktuellen Zeit zur Basiszeit. Dieses Verhalten können Sie natürlich ändern und z. B. grundsätzlich 0 s vorsehen.

## 288 Definition von Temperaturen

Das letzte Beispiel beschreibt die Klasse `TemperatureMeasures` zur Definition von Temperaturwerten. Sie ist etwas aufwändiger, da die Umrechnung zwischen den Einheiten nicht allein durch Multiplikation mit einem Faktor erfolgen kann, sondern auch Offsets berücksichtigt werden müssen. Aus diesem Grund werden die Property-Funktionen überschrieben. Listing 516 zeigt den vollständigen Code.

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Propor-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

```

' Temperatur in Kelvin
<Measurement("Temperatur", "Kelvin")> _
Public Class TemperatureMeasures
    Inherits MeasurementBase

    ' Die drei Einheiten
    Protected Shared TemperaturSymbols() As String = _
        {"K", "°C", "°F"}

    ' Faktoren und Offsets für die Umrechnung
    Protected Shared TemperaturFactors() As Double = {1, 1, 9 / 5}
    Protected Shared TemperaturOffsets() As Double = {0, -273.16, 32}

    Public Enum Units
        Kelvin
        Celsius
        Fahrenheit
    End Enum

    Public Sub New()
        Symbols = TemperaturSymbols
        Factors = TemperaturFactors
    End Sub

    Public Sub New(ByVal value As Double)
        MyClass.New()
        Value0 = value
    End Sub

    ' Zuweisung über die überschriebene Property
    Public Sub New(ByVal value As Double, ByVal unit As String)
        MyClass.New()
        Me.Value(unit) = value
    End Sub

    ' Zuweisung über die überschriebene Property
    Public Sub New(ByVal value As Double, ByVal unit As Units)
        MyClass.New()
        Me.Value(unit) = value
    End Sub

    ' Überschreibung für spezielle Berechnungen
    Public Overloads Overrides Property Value(ByVal unit As String) _
        As Double

        Get
            Dim i As Integer = GetIndex(unit)
            Return Me.Value(i)
        End Get
        Set(ByVal Value As Double)
            Dim i As Integer = GetIndex(unit)

```

---

```

        Me.Value(i) = Value
    End Set
End Property

' Überschreibung für spezielle Berechnungen
Public Overloads Overrides Property value(ByVal unitIndex _
    As Integer) As Double

    Get
        Select Case unitIndex
            Case Units.Kelvin : Return value0
            Case Units.Celsius : Return value0 + _
                TemperaturOffsets(1)
            Case Units.Fahrenheit : Return (value0 + _
                TemperaturOffsets(1)) * factors(2) + _
                TemperaturOffsets(2)
        End Select
    End Get
    Set(ByVal Value As Double)
        Select Case unitIndex
            Case Units.Kelvin : value0 = Value
            Case Units.Celsius : value0 = Value - _
                TemperaturOffsets(1)
            Case Units.Fahrenheit : value0 = (Value - _
                TemperaturOffsets(2)) _
                / factors(2) - TemperaturOffsets(1)
        End Select
    End Set
End Property

End Class

```

Listing 516: Die Klasse *TemperatureMeasures* für die Rechnung mit Temperaturen (Forts.)

Auch die Konstruktoren verwenden die überschriebenen Property-Methoden für die Umrechnung. Nachfolgend noch ein paar Beispielaufufe:

```

Dim T1 As New TemperatureMeasures(300)
Dim T2 As New TemperatureMeasures(100, "°C")
Dim T3 As New TemperatureMeasures(100, _
    TemperatureMeasures.Units.Fahrenheit)

Debug.WriteLine(T1.ToString( _
    TemperatureMeasures.Units.Celsius))
Debug.WriteLine(T2)
Debug.WriteLine(T3.ToString( _
    TemperatureMeasures.Units.Celsius))

```

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Propor-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

Und die erzeugte Ausgabe:

26,84 °C  
373,16 K  
37,78 °C

## 289 Universeller Umrechner für Maßeinheiten

Neben dem eigentlichen Ziel der vorangegangenen Rezepte, nämlich der typsicheren Definition physikalischer Größen, lässt sich mit den vorgestellten Klassen ganz einfach ein universeller Umrechner für Einheiten programmieren. Dieser braucht keinerlei Kenntnisse von Längen und Temperaturen, auch die entsprechenden Klassen muss er nicht kennen. Die Basisklasse `MeasurementBase` bietet alles, was für die Umsetzung erforderlich ist.

Folgende Anforderungen sollen erfüllt werden:

- ▶ Der Anwender kann die physikalische Größe aus einer Liste der verfügbaren auswählen.
- ▶ Für alle unterstützten Einheiten dieser Größe werden Eingabefelder mit den zugehörigen Symbolen angezeigt.
- ▶ Wird der Zahlenwert eines (beliebigen) Eingabefeldes geändert, werden alle anderen Felder mit den neuen umgerechneten Werten aufgefrischt.
- ▶ Der Code enthält keine spezifischen Zugriffe auf die von `MeasurementBase` abgeleiteten Klassen. Das Hinzufügen weiterer Klassen erfordert daher keine Änderung des Codes des Umrechners.

Abbildung 255 zeigt das Fenster des Umrechners für die Auswahl »Längenmaße«.

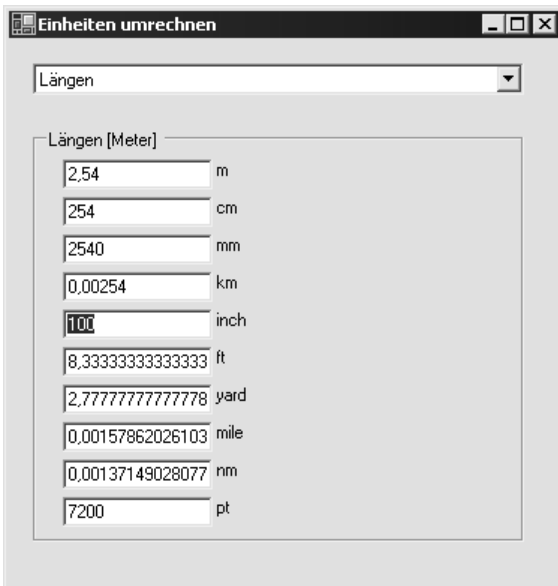


Abbildung 255: Universeller Umrechner für Maßeinheiten



In einer ComboBox (CBOMTypes) werden die Bezeichnungen der physikalischen Größen angezeigt, die zur Verfügung stehen. Ermittelt werden diese mit Hilfe der Methode `MeasurementBase.GetAllMeasurementClasses`, die oben beschrieben wurde. Sie gibt ein Array mit Referenzen auf `MeasurementInfo`-Objekte zurück, die direkt an die Items-Auflistung der ComboBox angehängt werden können. Listing 517 zeigt die Realisierung im `FormLoad`-Ereignis des Fensters.

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

    Dim types() As MeasurementInfo = _
        MeasurementBase.GetAllMeasurementClasses()

    CBOMTypes.Items.AddRange(types)

End Sub
```

Listing 517: Füllen einer ComboBox mit den Namen der verfügbaren physikalischen Größen

Auf einer GroupBox (PNLCalculate) werden nach Auswahl der physikalischen Größe für alle verfügbaren Maßeinheiten jeweils ein Label und eine TextBox angelegt. Die Referenzen der Controls werden in zwei Membervariablen der Form festgehalten.

```
Private Labels() As Label
Private TBs() As TextBox
```

Vor Anlegen der neuen Steuerelemente werden alle Steuerelemente, die bereits auf der GroupBox liegen, entfernt. Anschließend wird das `MeasurementInfo`-Objekt der ausgewählten Größe ermittelt. Das Objekt enthält einen Verweis auf das Typ-Objekt der zugehörigen Klasse. Mit `Activator.CreateInstance` wird dann eine Instanz dieser Klasse erzeugt und die Referenz in der Membervariablen `Measures` gespeichert.

In einer Schleife werden für jede Einheit ein Label für die Beschriftung und eine TextBox für den Zahlenwert angelegt. Listing 518 zeigt die Implementierung im Ereignis-Handler der ComboBox.

```
Private Sub CBOMTypes_SelectedIndexChanged( _
    ByVal sender As Object, ByVal e As System.EventArgs) _
    Handles CBOMTypes.SelectedIndexChanged

    ' Löschen der vorhandenen Steuerelemente
    Dim c As Control
    Do While PNLCalculate.Controls.Count > 0
        c = PNLCalculate.Controls(0)
        c.Dispose()
        PNLCalculate.Controls.Remove(c)
    Loop

    ' Ermitteln der Infos zur ausgewählten Einheit
    Dim mi As MeasurementInfo = DirectCast(DirectCast(sender, _
        ComboBox).SelectedItem, MeasurementInfo)
```

Listing 518: Anlegen von Labels und TextBoxen für die verfügbaren Einheiten

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

```

' Instanziierung der betreffenden Measurement-Klasse
Measures = DirectCast(Activator.CreateInstance(mi.Type), _
    MeasurementBase)

' Titel setzen
PNLCalculate.Text = mi.Attribute.Name & " [" & _
    mi.Attribute.Baseunit & "]"

' Lesen der verfügbaren Einheiten
Dim sb As String() = Measures.GetUnitSymbols()

Dim i As Integer

' Anlegen der Control-Arrays
Labels = New Label(sb.GetUpperBound(0)) {}
TBs = New TextBox(sb.GetUpperBound(0)) {}

' Einstellen der GroupBox-Höhe
PNLCalculate.Height = sb.Length * 25 + 25

' Für alle verfügbaren Einheiten
For i = 0 To sb.GetUpperBound(0)
    ' Label anlegen, Symbol der Einheit eintragen
    Labels(i) = New Label()
    Labels(i).Text = sb(i)
    ' Label positionieren und der Controls-Auflistung
    ' hinzufügen
    Labels(i).Location = New Point(120, 20 + i * 25)
    PNLCalculate.Controls.Add(Labels(i))

    ' TextBox anlegen und Position setzen
    TBs(i) = New TextBox()
    TBs(i).Location = New Point(20, 20 + i * 25)
    TBs(i).Width = 100
    ' Zuordnung des Symbols der Einheit
    TBs(i).Tag = sb(i)

    ' TextBox hinzufügen
    PNLCalculate.Controls.Add(TBs(i))

    ' Ereignis-Handler an TextBox binden
    AddHandler TBs(i).TextChanged, AddressOf ValueChanged
    AddHandler TBs(i).MouseUp, AddressOf TextBox_Enter
Next

' Initialisieren aller TextBoxen durch Setzen der ersten
TBs(0).Text = "0"

```

End Sub

Dem Tag-Feld jeder TextBox wird das Symbol der Einheit zugewiesen, um eine einfache Umrechnung zu erlauben. Alle TextBoxen werden an die Ereignis-Handler `TextChanged` und `MouseUp` gebunden.

Es gibt nur ein zentrales `Measurement`-Objekt, das für die Umrechnung zuständig ist. Eine Änderung in einer TextBox löst das `TextChanged`-Ereignis aus. Im gemeinsamen Handler `ValueChanged` wird der eingegebene Wert ermittelt. Die Prüfung mit `TryParse` ignoriert dabei nicht numerische Eingaben. Mit Hilfe des im Tag-Feld angegebenen Symbols wird der Zahlenwert umgerechnet und gespeichert.

In der nachfolgenden Schleife werden dann für alle TextBoxen die Zahlenwerte entsprechend ihrer zugewiesenen Einheit errechnet und ausgegeben. In Listing 519 sehen Sie die Realisierung.

Da die Zuweisung an die Text-Eigenschaft einer TextBox wiederum das `TextChanged`-Ereignis auslöst, wird über die boolesche Member-Variable `Changing` der Ablauf gesteuert und die Rekursion unterdrückt.

---

```

Private Sub ValueChanged(ByVal sender As Object, _
    ByVal e As System.EventArgs)

    ' Wenn das Event nicht unmittelbar durch eine Benutzeraktion
    ' ausgelöst wurde, nichts tun
    If Changing Then Exit Sub

    ' Welche TextBox hat das Ereignis ausgelöst?
    Dim tb As TextBox = DirectCast(sender, TextBox)
    Dim d As Double

    ' Zahlenwert einlesen
    ' Rücksprung, wenn die Eingabe nicht in Double gewandelt
    ' werden kann
    If Not Double.TryParse(tb.Text, _
        Globalization.NumberStyles.Float, _
        System.Globalization.NumberFormatInfo.CurrentInfo, d) _
        Then Exit Sub

    Dim i As Integer

    ' Wert umrechnen und speichern
    Measures.Value(DirectCast(tb.Tag, String)) = d

    ' Merker setzen, um rekursive Ereignisse abzublocken
    Changing = True

    ' Schleife über alle TextBoxen
    For i = 0 To TBs.GetUpperBound(0)

        ' Wert entsprechend der Einheit umrechnen und eintragen
        ' aber die auslösende TextBox nicht ändern
        If Not TBs(i) Is sender Then
            TBs(i).Text = Measures.Value(TBs(i).Tag.ToString()). _

```

---

Listing 519: Die Wertänderung einer beliebigen TextBox führt zur Neuberechnung der Werte der anderen TextBoxen

---

Basics

---

Datum/  
Zeit

---

Anwen-  
dungen

---

Zeich-  
nen

---

Bild-  
bearbei-  
tung

---

Win-  
dows  
Forms

---

Controls

---

Propert-  
yGrid

---

Datei-  
system

---

Netz-  
werk

---

System

---

Daten-  
banken

---

XML

---

Wissen-  
schaft

---

Ver-  
schiede-  
nes

---

---

```

        ToString()
    End If
Next

' Änderungen beendet
Changing = False

```

```
End Sub
```

---

*Listing 519: Die Wertänderung einer beliebigen TextBox führt zur Neuberechnung der Werte der anderen TextBoxen (Forts.)*

Der in Listing 520 dargestellte Ereignis-Handler dient lediglich dazu, bei Anwahl einer TextBox mit der Maus den gesamten Text zu markieren, damit der Anwender den Inhalt schnell überschreiben kann.

---

```

Private Sub TextBox_Enter(ByVal sender As System.Object, _
    ByVal e As MouseEventArgs)

    ' Welche TextBox hat das Ereignis ausgelöst?
    Dim tb As TextBox = DirectCast(sender, TextBox)

    ' Gesamten Text markieren
    tb.SelectionStart = 0
    tb.SelectionLength = tb.Text.Length

```

```
End Sub
```

---

*Listing 520: Markieren des gesamten Textes, wenn die TextBox den Fokus erhält*

| Einheit     | Wert             |
|-------------|------------------|
| m²          | 4046,8           |
| cm²         | 40468000         |
| mm²         | 4046800000       |
| km²         | 0,0040468        |
| square inch | 6272163,67017979 |
| square foot | 43559,4583161559 |
| square yard | 4839,94039175914 |
| acre        | 1                |
| square mile | 0,00156247104247 |

Abbildung 256: Umrechnen von Flächenmaßen

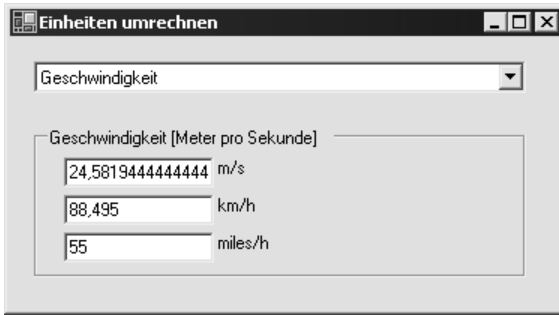


Abbildung 257: Umrechnen von Geschwindigkeiten

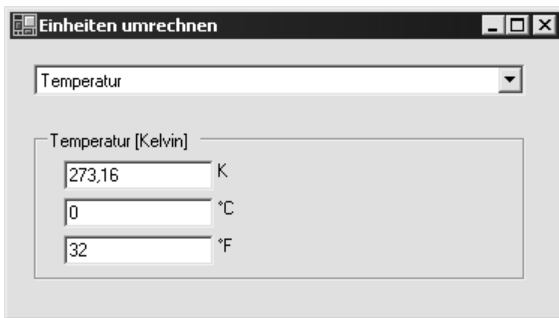


Abbildung 258: Umrechnen von Temperaturen

Die Abbildungen 256, 257 und 258 zeigen den Umrechner jeweils für Flächenmaße, Geschwindigkeiten und Temperaturen.

Der Umrechner arbeitet völlig unabhängig von den tatsächlichen Implementierungen der `Measurement`-Klassen. Sie können somit beliebige Klassen für andere technisch-physikalische Größen nach dem Muster der bereits beschriebenen hinzufügen und können den Umrechner ohne jede Anpassung sofort für diese Klassen einsetzen.

## 290 Schnittstelle für Datenquellen mit typsicheren physikalischen Werten

Oft werden physikalische Werte in Tabellen- oder Diagrammform dargestellt. Um Diagramme und Tabellen leichter und unabhängig von den tatsächlichen Daten erstellen zu können bietet es sich an, eine virtuelle Datenquelle zu definieren. Eine Schnittstelle legt die benötigten Funktionen fest, die implementiert werden müssen. Die in Listing 521 gezeigte Schnittstelle `IDataProvider` definiert die benötigten Funktionen, um z.B. Diagramme zeichnen zu können, die für einen vorgegebenen Wert  $x$  einen eindeutigen Funktionswert  $y$  ( $y = f(x)$ ) in Kurvenform darstellen.

```
Public Interface IDataProvider
```

```
    ' Lesen eines Wertes  $y=f(x)$ 
```

```
    Function GetValue(ByVal x As MeasurementBase) As MeasurementBase
```

Listing 521: Schnittstellendefinition für eine typsichere Datenquelle

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Proper-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

---

```

' Hilfsobjekt zur Typermittlung des (unabhängigen) x-Wertes
Function TypeX() As MeasurementBase

' Hilfsobjekt zur Typermittlung des (abhängigen) y-Wertes
Function TypeY() As MeasurementBase

' Bezeichnung der Datenquelle
Function GetName() As String

```

End Interface

---

*Listing 521: Schnittstellendefinition für eine typsichere Datenquelle (Forts.)*

Die Methoden `TypeX` und `TypeY` geben jeweils ein `Measurement`-Objekt zurück, das den Datentyp der unabhängigen Größe (x) bzw. der abhängigen Größe (y) widerspiegelt. Der Wert der Objekte spielt keine Rolle.

`GetValue` liefert für einen vorgegebenen x-Wert den zugehörigen y-Wert, `GetName` die Bezeichnung der Datenquelle.

Als Beispiel für eine Klasse, die diese Schnittstelle implementiert, dient die in Listing 522 dargestellte Klasse `SinusSample`. Sie definiert eine einfache Sinus-Funktion. Die unabhängige Größe ist mit dem Datentyp `TimeMeasures` auf Zeitangaben fixiert, die abhängige Größe hat einen frei wählbaren Typ, der durch die Zuweisung an die Eigenschaft `Amplitude` vorgegeben wird.

---

```

Public Class SinusSample
    Implements IDataProvider

    '  $y(t) = \text{Amplitude} * \sin(2 * \pi * \text{Frequency} * t) + \text{Offset}$ 
    ' Parameter zur Berechnung der Sinusfunktion
    Public Frequency As Double = 1
    Public Amplitude As MeasurementBase = New LinearMeasures(1)
    Public Offset As MeasurementBase = New LinearMeasures(0)
    Public Name As String

    Public Function GetValue(ByVal x As _
        MeasurementBase) As MeasurementBase _
        Implements Measurement.IDataProvider.GetValue

        '
        Debug.Assert(TypeOf x Is TimeMeasures)
        Dim v As MeasurementBase = DirectCast(Amplitude.Clone(), _
            MeasurementBase)

        v.Value = Math.Sin(x.Value * 2 * Math.PI * Frequency) * _
            Amplitude.Value + Offset.Value

        Return v

    End Function

```

---

*Listing 522: Beispiel-Implementierung von `IDataProvider`*

```

Public Function TypeX() As Measurement.MeasurementBase _
    Implements Measurement.IDataProvider.TypeX
    ' Typ für x ist TimeMeasures
    Return New TimeMeasures(0)
End Function

Public Function TypeY() As Measurement.MeasurementBase _
    Implements Measurement.IDataProvider.TypeY
    ' Typ für y ist wie für Amplitude festgelegt
    Return DirectCast(Amplitude.Clone(), MeasurementBase)
End Function

Public Function GetName() As String Implements _
    Measurement.IDataProvider.GetName
    Return Name
End Function

End Class

```

Listing 522: Beispiel-Implementierung von *IDataProvider* (Forts.)

Eine Instanzierung nach folgendem Muster wird in den nachfolgenden Tipps als Datenquelle für Diagramme verwendet:

```

' Sinusfunktion vom Typ Geschwindigkeit
ssmp = New SinusSample()
ssmp.Amplitude = New SpeedMeasures(50, _
    SpeedMeasures.Units.KilometerProStunde)
ssmp.Frequency = 0.33
ssmp.Name = "Geschwindigkeit(t)"
ssmp.Offset = New SpeedMeasures(50, _
    SpeedMeasures.Units.KilometerProStunde)

```

Diese Datenquelle dient nur zu Demonstrationszwecken. In der Realität werden Sie eine Klasse konstruieren, die z. B. Werte aus einer Datenbank liest oder über Treiberaufrufe Geräteschnittstellen abfragt. Auch der gemischte Betrieb, also historische Daten aus einer Datenbank und aktuelle Daten von Messgeräten, ist oft von großem Nutzen.

## 291 Skalierung für Diagramme berechnen

Sollen Messwerte oder errechnete Größen in Diagrammen dargestellt werden, müssen die Achsen des Koordinatensystems mit Zahlenwerten und ggf. Einheiten beschriftet werden. Selten können dabei die Einteilungen vorab festgelegt werden, da die benötigten Informationen für die Skalierung oft erst zur Laufzeit zur Verfügung stehen und sich dynamisch ändern können.

Als Anwendungsbeispiel sei das im nächsten Rezept vorgestellte  $f(t)$ -Diagramm genannt. Für eine auswählbare Datenquelle wird ermittelt, mit welcher physikalischen Einheit die Daten dargestellt werden sollen, und an Hand dieser Daten die Skalierung der Y-Achse vorgenommen.

Bei der Einteilung des Bereiches verwendet man üblicherweise glatte Zahlen. Aufgabe der Berechnungsmethode ist es daher, einen Bereich zu ermitteln, der in eine überschaubare Menge glatter

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Proper-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

Zahlen unterteilt werden kann. Zusätzlich muss das Zahlenformat bestimmt werden, damit nicht unnötig Nachkommastellen dargestellt werden.

Die wesentlichen Rückgabeinformationen einer solchen Berechnung bestehen aus dem Intervall und dem untersten Skalenwert. Es bietet sich daher an, Werte und Berechnung in einer Klasse zu verpacken. Listing 523 zeigt die Implementierung der Klasse `ScaleInfo`, die die Berechnungen ausführt.

`ScaleInfo` selbst verwendet nicht die `Measurement`-Klassen, so dass die Klasse auch in anderen Applikationen, die nur mit Zahlenwerten arbeiten, eingesetzt werden kann. Im Konstruktor werden alle benötigten Informationen übergeben. Da intern eine Instanz von `DoubleComparer` (siehe 505, Gleitkommazahlen vergleichen) für Zahlenvergleiche benötigt wird, ist der Konstruktor überladen worden, so dass wahlweise ein vorhandener Vergleich übergeben werden kann oder automatisch ein neuer angelegt wird.

Die Berechnung erfolgt in mehreren Schritten:

1. Nach Ermittlung des Bereiches wird die Dekade mit Hilfe des Zehnerlogarithmus bestimmt. Sie ist maßgeblich für die Anzahl der darzustellenden Stellen.
2. Bei glatten Zehnerpotenzen liefert der Logarithmus bereits die nächste Dekade. Da aber beispielsweise 100 in der gleichen Dekade liegen soll wie 99 und 11, wird dieser Fall gesondert berücksichtigt.
3. Die Aufteilung in Unterbereiche ist abhängig davon, wie groß der belegte Wertebereich innerhalb der ermittelten Dekade ist. Anhand der Abweichung von der Dekade werden die folgenden Bereiche festgelegt:
  1.  $0 < \text{Abweichung} \leq 2$   
Die Aufteilung erfolgt in Schritten von 20% der Dekade, also z.B.  $-4 \cdot -2 \cdot 0 \cdot 2 \cdot 4$
  2.  $2 < \text{Abweichung} \leq 5$   
Die Aufteilung erfolgt in Schritten von 50% der Dekade, also z.B.  $-150 \cdot -100 \cdot -50 \cdot 0 \cdot 50 \cdot 100 \cdot 150$
  3. Ansonsten in Schritten der Dekade, also z.B.  $-4 \cdot -3 \cdot -2 \cdot -1 \cdot 0 \cdot 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5$
4. Nach Festlegung des Intervalls wird der Wert der kleinsten Skalierung berechnet. Ausgehend von diesem Wert muss in einem Diagramm die Skala aufgebaut werden.
5. Zuletzt wird das Zahlenformat ermittelt. Hier ist insbesondere die Bestimmung der Nachkommastellen von Bedeutung.

Hinweis

Der unterste Skalenwert muss nicht zwangsläufig das Minimum des Wertebereiches und somit die kleinste Koordinate sein. Wenn der Wertebereich mit einer »krummen« Zahl beginnt, wird der unterste Skalenwert größer sein.

```
Public Class ScaleInfo
    ' Unterster Skalenwert
    Public ScaleStart As Double
```



```

' Intervall der Skalierung
Public ScaleInterval As Double

' Zahlenformat für Skalenwerte
Public FormatString As String

' Vergleicher für Double
Public Comparer As DoubleComparer

' Wertebereich
Public Range As Double

' Konstruktor, der selbst eine Instanz von DoubleComparer anlegt
Public Sub New(ByVal min As Double, ByVal max As Double, _
    ByVal unit As String)

    ' Comparer anlegen und anderen Konstruktor aufrufen
    MyClass.New(min, max, unit, _
        New DoubleComparer((max - min) / 10000000000.0))

End Sub

' Hier erfolgt die Berechnung
Public Sub New(ByVal min As Double, ByVal max As Double, _
    ByVal unit As String, ByVal comparer As DoubleComparer)

    Me.Comparer = comparer

    ' Der gesamte Bereich
    Me.Range = max - min
    Debug.Assert(Me.Range > 0)

    ' Dekade und Teilbereich innerhalb der Dekade feststellen
    Dim z As Double = Math.Log10(Range)
    Dim iDecade As Double = Math.Floor(z)

    ' Sonderfall: glatte Zehnerpotenz berücksichtigen
    ' Z.B. 100 wird behandelt wie 99
    If (comparer.IsEqual(iDecade, z)) Then iDecade -= 1

    ' Dekade als glatte Zehnerpotenz
    Dim Decade As Double = 10 ^ iDecade

    ' Abweichung von der Dekade
    z = 10 ^ (z - iDecade)

    ' Teilung bestimmen
    If z > 0 And z <= 2 Then
        ScaleInterval = 0.2 * Decade
    ElseIf z > 2 And z <= 5 Then
        ScaleInterval = 0.5 * Decade

```

---

```

Else
    ScaleInterval = 1 * Decade
End If

' Unterste Skalierung berechnen
If comparer.IsEqual(Math.Floor(min / ScaleInterval) * _
    ScaleInterval, min) Then

    ScaleStart = min
Else
    ScaleStart = (Math.Floor(min / ScaleInterval) + 1) * _
        ScaleInterval
End If

' Zahlenformat bestimmen (insb. für Nachkommastellen)
Dim frmt As String
Dim offs As Double
If z >= 5 Then
    offs = 0
Else
    offs = 1
End If

If iDecade < 0 Then
    frmt = "0." & New String("0"c, CInt(-iDecade + offs))
ElseIf comparer.IsZero(iDecade) Then
    If z >= 5 Then
        frmt = "0"
    Else
        frmt = "0.0"
    End If
Else
    frmt = "0"
End If

' Einheit anhängen
FormatString = "{0:" & frmt & "} " & unit

End Sub
End Class

```

---

*Listing 523: Berechnung der Skalenwerte für Diagramme (Forts.)*

Die Anwendung der vorgestellten Klasse erfolgt nach folgendem Schema:

---

```

Dim Max As Double = ...
Dim SI As New ScaleInfo(...)
' Kleinster Skalenwert
Dim y As Double = SI.ScaleStart

' Schleife zur Ausgabe der Skala
Do While SI.Comparer.IsLessOrEqual(y, Max)

```

---

```

Position (y) berechnen
String mit Format SI.FormatString a

' Nächster Wert
y += SI.ScaleInterval
Loop

```

Auf der CD finden Sie ein kleines Testprogramm, mit dessen Hilfe Sie die Ergebnisse der Berechnungen der Skalenwerte betrachten können (s. Abbildung 259). Im nächsten Rezept wird eine praktische Anwendung vorgestellt.



Abbildung 259: Beispiel für berechnete Skalenwerte

## 292 Einfaches T-Y-Diagramm mit statischen Werten

Unter Einbeziehung der vorangegangenen Rezepte wird nun ein Diagramm-Steuererelement vorgestellt, das für einen festgelegten Bereich von Zeitwerten die zugehörigen Y-Werte der ausgewählten Datenquellen in Linienform darstellt. Das Diagramm besteht aus drei Bereichen. Oben wird die Skalierung der Zeitachse angezeigt, links die der Y-Achse. Der restliche Bereich dient der Kurvendarstellung.

Dem Steuererelement können beliebige Datenquellen hinzugefügt werden, sofern sie die Schnittstelle `IDataProvider` implementieren. Über ein Kontextmenü lässt sich einstellen, für welche Datenquelle die Skalierung der Y-Achse vorgenommen werden soll und in welcher Maßeinheit die Skalenwerte darzustellen sind.

Um Aufbau und Funktion demonstrieren zu können, werden als Datenquellen zwei Instanzen der in Kapitel 14.9 vorgestellten `SinusSample`-Klasse verwendet:

1. Distanz mit einem Wertebereich von +/- 4 m, Farbe Rot
2. Geschwindigkeit mit einem Wertebereich von 0 bis 100 km/h, Farbe Blau

Abbildung 260 zeigt den Ausgangszustand des Diagramms. Leider geht in der Schwarz/Weiß-Darstellung naturgemäß die Farbinformation verloren. Die aktive Kurve, für die die Skalierung der Y-Achse angezeigt wird, wird daher mit einer dicken Linie gezeichnet.

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Propert-  
yGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

Im Beispielprogramm wird das Diagramm-Steuerelement auf einem Fenster platziert und die Dock-Eigenschaft auf `Fill` gesetzt, so dass das Steuerelement die gesamte Fensterfläche einnimmt. Die Initialisierung besteht lediglich aus der Bereitstellung der beiden Datenquellen, dem Hinzufügen dieser Datenquellen und der Einstellung einiger Parameter:

---

```
Dim ssmp As SinusSample

' Sinusfunktion vom Typ Längenmaß
ssmp = New SinusSample()
ssmp.Frequency = 0.1
ssmp.Amplitude = New LinearMeasures(4)
ssmp.Name = "Distanz(t)"
' Datenquelle hinzufügen (Typ Längenmaß)
TyPlot1.AddDataProvider(ssmp, New LinearMeasures(-5), _
    New LinearMeasures(5), Color.Red)

' Sinusfunktion vom Typ Geschwindigkeit
ssmp = New SinusSample()
ssmp.Amplitude = New SpeedMeasures(50, _
    SpeedMeasures.Units.KilometerProStunde)
ssmp.Frequency = 0.33
ssmp.Name = "Geschwindigkeit(t)"
ssmp.Offset = New SpeedMeasures(50, _
    SpeedMeasures.Units.KilometerProStunde)

' Datenquelle hinzufügen (Typ Geschwindigkeit)
TyPlot1.AddDataProvider(ssmp, New SpeedMeasures(0), _
    New SpeedMeasures(100, _
    SpeedMeasures.Units.KilometerProStunde), Color.Blue)

' Anfangs- und Endzeit für Kurvendarstellung
TyPlot1.StartTime = New TimeMeasures(0)
TyPlot1.EndTime = New TimeMeasures(27)

' Anzahl darzustellender Bildpunkte
TyPlot1.NumberOfPoints = 200

' Diagramm zeichnen
TyPlot1.Plot()
```

---

Mehr ist hier nicht zu tun. Das Steuerelement untersucht selbst, welchen Datentyp die Datenquellen liefern, und passt die Skalierung der Y-Achse automatisch an.

### Aufbau des Diagramm-Steuerelementes TYPlot

Alle Member-Variablen und Methoden der Klasse `TYPlot` sind geschützt, damit die Klasse durch Ableitung erweitert werden kann. Diese Variablen werden für die Kurvendarstellung benötigt:

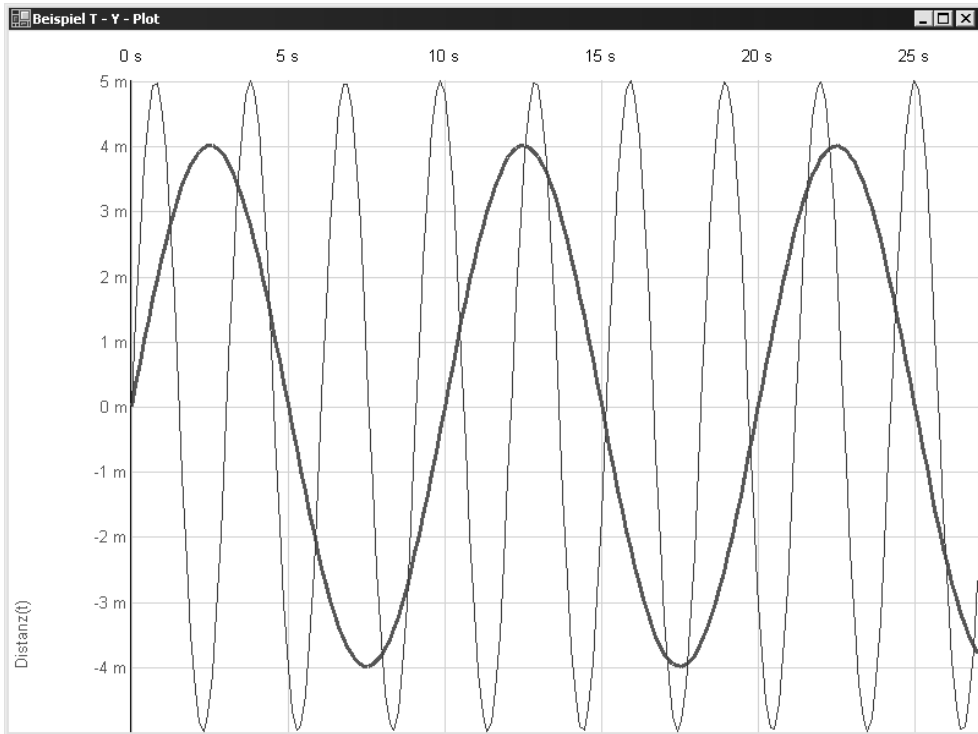


Abbildung 260: Ausgangszustand des Beispieldiagramms

```

' Anfangs- und Endzeit
Protected T0, Tn As TimeMeasures

' Anzahl der darzustellenden Punktepaare
Protected nPoints As Integer

' Liste der Datenquellen
Protected DataProviderList As New ArrayList()

' Array mit den zuletzt gezeichneten Werten
Protected LastValues() As MeasurementBase

' Array mit den neu zu zeichnenden Werten
Protected NewValues() As MeasurementBase

' Bildpuffer
Protected PicBuf As Bitmap

' Offsets für den Zeichenbereich
Protected OffsetX As Integer = 100
Protected OffsetY As Integer = 40

' Größe des Zeichenbereichs
Protected DrawWidth As Integer

```

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Propor-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

---

Protected DrawHeight As Integer

*' Stringformate für die Achsenbeschriftungen*

Protected StringFormatTAxis As New StringFormat()

Protected StringFormatYAxis As New StringFormat()

*' Font für die Achsenbeschriftungen*

Protected ScaleFont As New Font("Arial", 10)

*' Index der ausgewählten Datenquelle*

Public DIdxSkala As Integer = 0

*' Index der ausgewählten Einheit für die Y-Achse*

Public DPUntIdxSkala As Integer = 0

---

Sie dienen größtenteils zur Speicherung der Einstellungen, die das Erscheinungsbild festlegen. Lediglich drei Variablen stehen über öffentliche Property-Methoden nach außen zur Verfügung:

---

Public Property StartTime() As TimeMeasures *' T0*

Public Property EndTime() As TimeMeasures *' Tn*

Public Property NumberOfPoints() As Integer *' nPoints*

---

Die Grafikausgabe erfolgt nicht direkt in das Graphics-Objekt des Steuerelementes, sondern in eine Bitmap gleicher Größe, deren Referenz in PicBuf gespeichert wird. Um das Flackern der Ausgabe zu vermeiden, werden im Konstruktor von TYPlot einige ControlStyles gesetzt:

---

InitializeComponent()

SetStyle(ControlStyles.AllPaintingInWmPaint, True)

SetStyle(ControlStyles.UserPaint, True)

SetStyle(ControlStyles.DoubleBuffer, True)

StringFormatYAxis.Alignment = StringAlignment.Far

StringFormatYAxis.LineAlignment = StringAlignment.Center

StringFormatTAxis.Alignment = StringAlignment.Center

StringFormatTAxis.LineAlignment = StringAlignment.Center

---

StringFormatYAxis und StringFormatTAxis werden für die Ausrichtung der Skalentexte benötigt.

Die ebenfalls öffentliche Methode AddDataProvider (s. Listing 524) ermöglicht das Hinzufügen der Datenquellen. Sie legt eine Instanz der Hilfsklasse TYPlotInfo (s. Listing 525) an, die als interne Klasse von TYPlot definiert ist. Die zum Zeichnen benötigten Objekte (Pen, Brush, Font) werden angelegt und die Referenzen in der Info-Klasse gespeichert.

---

Public Sub AddDataProvider(ByVal dataProvider As IDataProvider, \_

ByVal minValue As MeasurementBase, ByVal maxValue As \_  
MeasurementBase, ByVal LineColor As Color)

*' Info-Objekt für diese Datenquelle anlegen*

---

Listing 524: Hinzufügen von Datenquellen mit AddDataProvider

```

Dim pinfo As New TYPlotInfo()
pinfo.Max = maxValue
pinfo.Min = minValue
pinfo.LineColor = LineColor

' Erste Datenquelle hat eine dicke Linie
If DataProviderList.Count = 0 Then
    pinfo.LinePen = New Pen(LineColor, 3)
Else
    pinfo.LinePen = New Pen(LineColor)
End If

pinfo.FontBrush = New SolidBrush(LineColor)

' Referenz der Datenquelle
pinfo.DP = dataProvider

' Info an Liste anhängen
DataProviderList.Add(pinfo)

' Array-Größen anpassen
ReDim LastValues(DataProviderList.Count)
ReDim NewValues(DataProviderList.Count)

End Sub

```

Listing 524: Hinzufügen von Datenquellen mit AddDataProvider (Forts.)

```

Protected Class TYPlotInfo
    ' Die Datenquelle
    Public DP As IDataProvider

    ' Verwendete Farbe für Darstellung
    Public LineColor As Color

    ' Größter und kleinster Wert für Y-Achse
    Public Min, Max As MeasurementBase

    ' Stift zum Zeichnen der Linie
    Public LinePen As Pen

    ' Brush zum Zeichnen der Skalenbeschriftung
    Public FontBrush As Brush

End Class

```

Listing 525: Hilfsklasse TYPlotInfo mit Informationen zur Datenquelle

Aufgebaut wird das Diagramm in der geschützten Methode Plot. Hier werden alle grafischen Ausgaben vorgenommen und auf dem Graphics-Objekt des Bildpuffers (PicBuf) ausgeführt. Zunächst wird der gesamte Bildbereich gelöscht. Das Zeichnen der Achsen-Skalierungen und des Skalierungsgitters übernehmen die Funktionen DrawYAxis und DrawTAxis.

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Proper-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

In einer Schleife wird das Array `LastValues` mit den Werten der Datenquellen für den Zeitpunkt `T0` gefüllt. Sie dienen als Startpunkt für die jeweils erste Linie der Kurvendarstellung.

Letztlich erfolgt das Zeichnen der Liniensegmente. Für die angegebene Auflösung (`nPoints`) werden in einer Schleife alle Datenpunkte durchlaufen. Zu jedem Punkt werden die zugehörige Zeit (`t`) sowie die Position im Diagramm (`x2`) ermittelt. Danach wird für jede Datenquelle der Wert für diesen Zeitpunkt abgefragt und die `y`-Koordinaten für diesen und den vorhergehenden Punkt (Array `LastValues`) errechnet. `x1`, `x2`, `y1` und `y2` sind dann die Koordinaten des zu zeichnenden Linienelementes, das mit dem im `TYPlotInfo`-Objekt gespeicherten Stift gezeichnet wird.

Nach Abschluss der Schleifen sind alle Liniensegmente für alle Datenquellen gezeichnet. Mit `Refresh` wird abschließend das Neuzeichnen des Fensters in der überschriebenen `OnPaint`-Methode erzwungen. Listing 526 zeigt die Implementierung der Methode `Plot`.

---

```

' Diagramm zeichnen
Protected Sub Plot()

    ' Hilfsvariablen
    Dim i As Integer
    Dim pinfo As TYPlotInfo
    Dim dpidx As Integer
    Dim t As TimeMeasures
    Dim x1, x2 As Integer

    ' Graphics-Objekt des Puffers holen
    Dim g As Graphics = Graphics.FromImage(PicBuf)

    ' Gesamten Bereich löschen
    g.Clear(Color.White)

    ' Info-Objekt für ausgewählte Datenquelle ermitteln
    pinfo = DirectCast(DataProviderList(DPIDxSkala), TYPlotInfo)

    ' Y-Achse zeichnen
    DrawYAxis(g, pinfo)

    ' X-Achse zeichnen
    DrawTAxis(g)

    ' Für alle Datenquellen
    For dpidx = 0 To DataProviderList.Count - 1
        ' Info ermitteln
        pinfo = DirectCast(DataProviderList(dpidx), TYPlotInfo)

        ' Anfangswerte der Kurven ermitteln
        LastValues(dpidx) = pinfo.DP.GetValue(T0)
    Next

    ' Kurven zeichnen
    ' (Zeichnen von x1 nach x2)

```

---



```

' Ausgangsposition
x1 = OffsetX

' Für alle darzustellenden Punktepaare
For i = 1 To nPoints

    ' Zeitwert für Position ermitteln
    t = New TimeMeasures(T0.Value + (Tn.Value - T0.Value) / _
        nPoints * i)

    ' Position für zweiten Punkt
    x2 = OffsetX + CInt(DrawWidth * (t.Value - T0.Value) / _
        (Tn.Value - T0.Value))

    ' Für alle Datenquellen
    For dpidx = 0 To DataProviderList.Count - 1

        ' Info holen
        pinfo = DirectCast(DataProviderList(dpidx), TYPlotInfo)

        ' Wert ermitteln
        NewValues(dpidx) = pinfo.DP.GetValue(t)

        ' y-Koordinaten berechnen
        Dim h As Double = pinfo.Max.Value - pinfo.Min.Value
        Dim y1 As Integer = DrawHeight + OffsetY - _
            CInt((-pinfo.Min.Value + LastValues(dpidx).Value) _
                / h * DrawHeight)

        Dim y2 As Integer = DrawHeight + OffsetY - _
            CInt((-pinfo.Min.Value + NewValues(dpidx).Value) _
                / h * DrawHeight)

        ' Liniensegment zeichnen
        g.DrawLine(pinfo.LinePen, x1, y1, x2, y2)

        ' Array der letzten Werte auffrischen
        LastValues(dpidx) = NewValues(dpidx)
    Next

    x1 = x2
Next

' Neu zeichnen
Refresh()

End Sub

```

---

End Sub

Listing 526: Zeichnen des Diagramms in der Methode Plot (Forts.)

Listing 527 zeigt die überschriebene `OnPaint`-Methode, in der lediglich das im Puffer gespeicherte Abbild gezeichnet wird. So kann das Neuzeichnen der Steuerelemente erfolgen, ohne dass das Diagramm neu berechnet werden muss.

---

Basics

---

Datum/  
Zeit

---

Anwen-  
dungen

---

Zeich-  
nen

---

Bild-  
bearbei-  
tung

---

Win-  
dows  
Forms

---

Controls

---

Proper-  
tyGrid

---

Datei-  
system

---

Netz-  
werk

---

System

---

Daten-  
banken

---

XML

---

Wissen-  
schaft

---

Ver-  
schiede-  
nes

---

```
Protected Overrides Sub OnPaint(ByVal e As _
    System.Windows.Forms.PaintEventArgs)

    ' Gespeichertes Bild zeichnen
    e.Graphics.DrawImage(PicBuf, 0, 0, Width, Height)
End Sub
```

---

*Listing 527: Zeichnen des Bildpuffers in OnPaint*

Wird die Größe des Steuerelementes geändert (im Beispiel durch Änderung der Fenstergröße), dann muss das Diagramm neu aufgebaut werden. In der Überschrift der Methode `OnResize` (s. Listing 528) wird daher der Bildpuffer neu eingerichtet, die Größe des Zeichenbereiches neu bestimmt und, sofern Datenquellen existieren, durch den Aufruf von `Plot` das Diagramm neu gezeichnet.

---

```
Protected Overrides Sub OnResize(ByVal e As System.EventArgs)

    ' Wenn der Puffer schon existiert, entsorgen
    If Not PicBuf Is Nothing Then PicBuf.Dispose()

    ' Neuen Puffer mit passender Größe anlegen
    PicBuf = New Bitmap(Width, Height)

    ' Zeichenbereich ermitteln
    DrawWidth = ClientSize.Width - OffsetX - 1
    DrawHeight = ClientSize.Height - OffsetY - 1

    ' Neu zeichnen
    If DataProviderList.Count <> 0 Then Plot()

End Sub
```

---

*Listing 528: Bei Größenänderung Ausgabe neu aufbauen in OnResize*

Zum Zeichnen der Skalierungen wird auf die Klasse `ScaleInfo` zurückgegriffen. Listing 529 zeigt die Implementierung der Methode `DrawTAxis`, die die Skalierung der Zeitachse vornimmt, Listing 530 die Implementierung der Methode `DrawYAxis`, die für die Skalierung des Wertebereiches zuständig ist.

Der Bereich der Zeitachse ergibt sich aus dem Wert der Member-Variablen `T0` und `Tn`. Mit Instanziierung der `ScaleInfo`-Klasse werden die benötigten Informationen zum Zeichnen der Skalierung bereitgestellt. In einer Schleife wird dann beginnend mit der kleinsten Skalenzeit für jede Skaleneinteilung eine senkrechte Linie und, sofern der Platz ausreicht, die Zeit in Sekunden gezeichnet. Im Listing befindet sich auch Code für eine kontinuierliche Variante des Diagramms, die im nächsten Rezept beschrieben wird.

---

```
Protected Sub DrawTAxis(ByVal g As Graphics)

    ' Bereich für Achsenbeschriftung löschen
```

---

*Listing 529: Zeitachse zeichnen in DrawTAxis*

```

g.FillRectangle(Brushes.White, OffsetX, 0, DrawWidth, OffsetY)

' Skalierungsinformation ermitteln
Dim SI As New ScaleInfo(T0.Value, Tn.Value, "s")

' Erste Zeit für Zeitachsenbeschriftung
Dim t As Double = SI.ScaleStart

' Für alle Skalenbeschriftungen
Do While SI.Comparer.IsLessOrEqual(t, Tn.Value)

    ' x-Position berechnen
    Dim x As Integer = CInt(OffsetX + (t - T0.Value) / _
        (Tn.Value - T0.Value) * DrawWidth)

    ' Umschließendes Rechteck für Beschriftung festlegen
    Dim r As New RectangleF(x - 40, 10, 80, 20)

    ' Ausgabe nur, wenn das Rechteck vollständig sichtbar ist
    If (x + 40 < OffsetX + DrawWidth) Then

        ' Wert formatieren
        Dim s As String = String.Format(SI.FormatString, t)

        ' Text formatiert ausgeben
        g.DrawString(s, ScaleFont, Brushes.Black, r, _
            StringFormatTAxis)

        ' Für kontinuierlichen Betrieb Skalierungszeit merken
        LastScaleTime = t

    End If

    ' Senkrechte Linie zeichnen
    g.DrawLine(Pens.LightGray, x, OffsetY, x, Height)

    ' Nächstes Zeitintervall
    t += SI.ScaleInterval

Loop

' Für kontinuierlichen Betrieb
TScaleInfo = SI

End Sub

```

Listing 529: Zeitachse zeichnen in DrawTAxis (Forts.)

```

Protected Sub DrawYAxis(ByVal g As Graphics, _
    ByVal pinfo As TYPlotInfo)

```

Listing 530: Skalierung des Wertebereiches in DrawYAxis

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Propert-  
yGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schie-  
denes

---

```

' Bereich löschen
g.FillRectangle(Brushes.White, 0, 0, OffsetX - 1, Height)

' Senkrechte Linie für die Abgrenzung zeichnen
g.DrawLine(Pens.Black, OffsetX - 1, OffsetY, OffsetX - 1, _
    Height)

' Temporäre Koordinatentransformation zur gedrehten Ausgabe
' der Datenquellenbezeichnung
Dim gc As Drawing2D.GraphicsContainer = g.BeginContainer()
g.TranslateTransform(2, Height)
g.RotateTransform(- 90)
g.DrawString(pinfo.DP.GetName(), ScaleFont, pinfo.FontBrush, _
    50, 0)

g.EndContainer(gc)

' Größe des Wertebereichs
Dim Min As Double = pinfo.Min.Value(DPUnitIdxSkala)
Dim Max As Double = pinfo.Max.Value(DPUnitIdxSkala)
Dim Range As Double = Max - Min

' Skalierungsinformationen berechnen
Dim SI As New ScaleInfo(Min, Max, _
    pinfo.Max.GetUnitSymbols()(DPUnitIdxSkala))

' Für kontinuierlichen Betrieb merken
YScaleInfo = SI

' Erster Skalenwert
Dim y As Double = SI.ScaleStart

' Skalenwerte ausgeben und horizontale Linien zeichnen
Do While SI.Comparer.IsLessOrEqual(y, Max)
    ' Wert formatieren
    Dim s As String = String.Format(SI.FormatString, y)

    ' Position für Textausgabe errechnen
    Dim ys As Integer = CInt(DrawHeight * (y - Min) / Range)
    Dim r As New RectangleF(20, Height - ys - 10, _
        OffsetX - 20, 20)

    ' Ausgabe nur, wenn Text vollständig gezeichnet werden kann
    If ys >= 10 Then
        g.DrawString(s, ScaleFont, pinfo.FontBrush, r, _
            StringFormatYAxis)
    End If

    ' Waagerechte Linien zeichnen
    g.DrawLine(Pens.LightGray, OffsetX, Height - ys, Width, _

```

---

```

        Height - ys)

        ' Nächstes Skalenintervall
        y += SI.ScaleInterval

    Loop
End Sub

```

Listing 530: Skalierung des Wertebereiches in DrawYAxis (Forts.)

Bevor die Skalierung der Y-Achse vorgenommen werden kann, muss ermittelt werden, für welche Datenquelle und in welcher Einheit die Werte angezeigt werden sollen. Die Bezeichnung der Datenquelle wird, um 90° gedreht, links neben den Skalenwerten ausgegeben. Für die Drehung der Schrift wird das Koordinatensystem mit `TranslateTransform` und `RotateTransform` vorübergehend verschoben und gedreht. Durch den Aufruf von `g.BeginContainer` wird der vorherige Zustand des Koordinatensystems gespeichert und durch den Aufruf von `g.EndContainer` wieder hergestellt.

Minimum und Maximum des Wertebereichs werden auf Basis der ausgewählten Einheit berechnet. Dadurch ergeben sich unterschiedliche Zahlenwerte, wenn z.B. Meter oder Meilen ausgewählt werden. Beide Werte sowie das Symbol der ausgewählten Einheit werden dem Konstruktor von `ScaleInfo` übergeben, der die Skalierung berechnet.

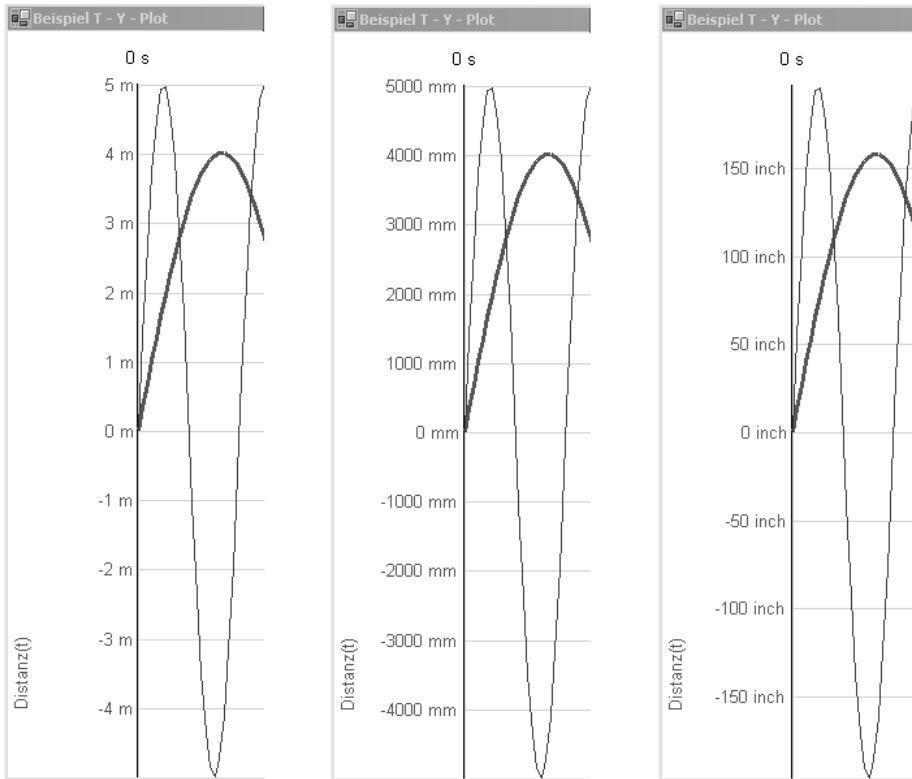


Abbildung 261: Skalierung der Distanz in Meter, Millimeter und Inch

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Propor-  
tivityGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

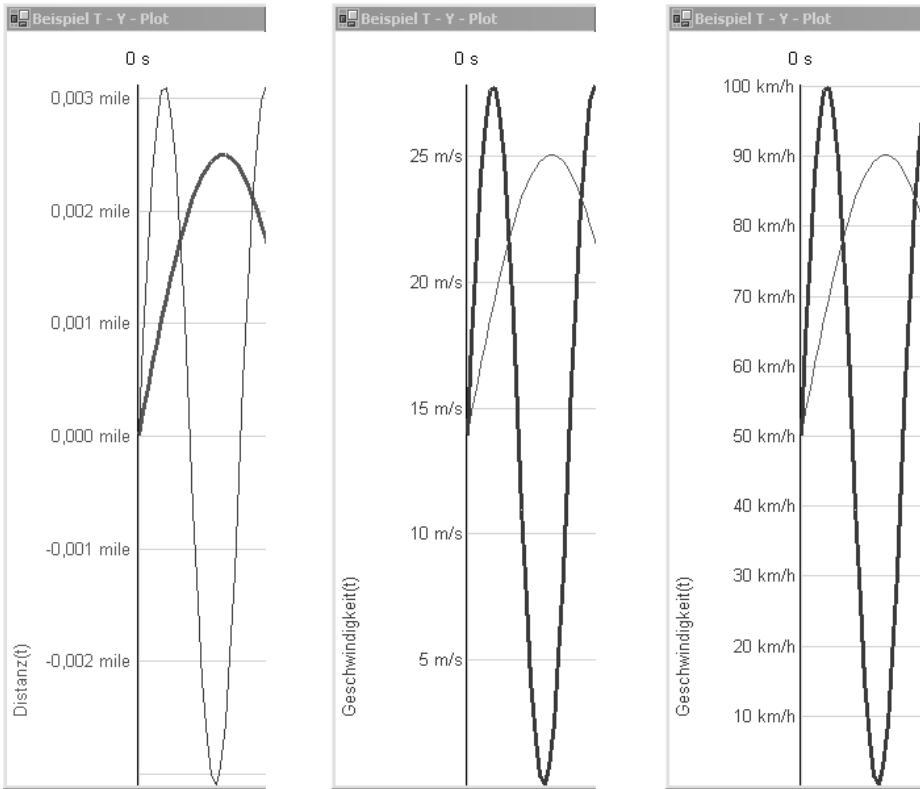


Abbildung 262: Skalierungen in Meilen, Geschwindigkeit in Meter pro Sekunde und Kilometer pro Stunde

Analog zur Implementierung von `DrawTAxis` werden dann die Skalenwerte und Gitternetzlinien für die Y-Achse gezeichnet. Abbildung 261 und Abbildung 262 zeigen Ausschnitte des Diagramms mit verschiedenen Einstellungen.

Zwar stellt das Diagramm gleichzeitig die Kurven aller Datenquellen dar, doch die Skalierung der Y-Achse erfolgt immer nur für eine ausgewählte Datenquelle. Die Auswahl läuft über ein Kontextmenü. Für die ausgewählte Datenquelle kann dann eine beliebige Einheit ausgewählt werden, sofern sie die physikalische Größe der Datenquelle repräsentiert (s. Abbildung 263).

Dem Anwender wird eine öffentliche Methode zum Zeichnen des Diagramms zur Verfügung gestellt: `ShowDiagram`.

In dieser Methode (Listing 531) wird zunächst für jede hinzugefügte Datenquelle ein Eintrag in einem Kontextmenü angelegt. Die Reihenfolge ist identisch mit der Ablage im `ArrayList DataProviderList`, damit die Indizierung 1:1 erfolgen kann. Anschließend wird mit

---

```
MISource.MenuItems(0).PerformClick()
```

---

die Auswahl der ersten Datenquelle simuliert und dadurch das Diagramm gezeichnet.

```

Public Sub ShowDiagram()

    ' Kontextmenü neu aufbauen
    Dim pinfo As TYPlotInfo
    Dim dpidx As Integer

    ' Kontextmenü löschen
    MISource.MenuItems.Clear()

    ' Für alle Datenquellen
    For dpidx = 0 To DataProviderList.Count - 1
        ' Info ermitteln
        pinfo = DirectCast(DataProviderList(dpidx), TYPlotInfo)

        ' Neuen Menüeintrag hinzufügen und an Handler binden
        Dim mnu As MenuItem = New MenuItem()
        mnu.Text = pinfo.DP.GetName()
        AddHandler mnu.Click, AddressOf MenuSourceClick
        MISource.MenuItems.Add(mnu)

    Next

    ' Erste Datenquelle auswählen
    ' Dadurch wird das Diagramm gezeichnet
    MISource.MenuItems(0).PerformClick()

```

End Sub

*Listing 531: ShowDiagram legt das Kontextmenü an und initiiert den Zeichenvorgang*

Den Handler für die Auswahl der Datenquelle zeigt Listing 532. Hier werden die Stiftbreiten für aktive und inaktive Kurven eingestellt und das Menü für die Auswahl der verfügbaren Einheiten aufgebaut. Anschließend wird das Diagramm neu gezeichnet.

```

Private Sub MenuSourceClick(ByVal sender As Object, _
    ByVal e As System.EventArgs)

    ' Linienbreite der zuvor ausgewählten Datenquelle zurücksetzen
    Dim pinfo As TYPlotInfo = DirectCast( _
        DataProviderList(DPIdxSkala), TYPlotInfo)

    pinfo.LinePen.Dispose()
    pinfo.LinePen = New Pen(pinfo.LineColor)

    ' Index feststellen
    Dim mnu As MenuItem = DirectCast(sender, MenuItem)
    DPIdxSkala = mnu.Index

    ' Informationen hierzu ermitteln
    pinfo = DirectCast(DataProviderList(DPIdxSkala), TYPlotInfo)

```

*Listing 532: Auswahl einer Datenquelle über Kontextmenü bearbeiten*

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

```

pinfo.LinePen.Dispose()
pinfo.LinePen = New Pen(pinfo.LineColor, 3)

' Einheit ist zunächst die Standard-Einheit
DPUntIdxSkala = 0

' Für alle verfügbaren Symbole neue Kontext-Submenüs anlegen
'Dim sb As String() = mnu.Pinfo.DP.TypeY.GetUnitSymbols()
Dim sb As String() = pinfo.DP.TypeY.GetUnitSymbols()

Dim s As String
MIUnit.MenuItems.Clear()
For Each s In sb
    MIUnit.MenuItems.Add(s, AddressOf MenuUnitClick)
Next

' Diagramm neu zeichnen
Plot()

```

End Sub

---

Listing 532: Auswahl einer Datenquelle über Kontextmenü bearbeiten (Forts.)

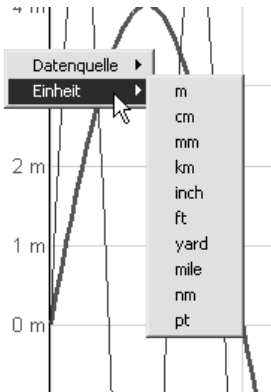


Abbildung 263: Auswahl von Datenquelle und Einheit für die Skalierung der Y-Achse

Auch nach Auswahl einer Einheit muss das Diagramm neu gezeichnet werden, da sich auch die Gitternetzlinien ändern können. Der Aufruf erfolgt im Ereignis-Handler `MenuUnitClick` (Listing 533). Nach Speicherung des Auswahlindex wird erneut `Plot` aufgerufen.

---

```

Private Sub MenuUnitClick(ByVal sender As Object, _
    ByVal e As System.EventArgs)

    ' Index der Einheit ermitteln
    Dim mnu As System.Windows.Forms.MenuItem = _
        DirectCast(sender, System.Windows.Forms.MenuItem)
    DPUntIdxSkala = mnu.Index

```

---

Listing 533: Auf Auswahl einer Einheit reagieren



```
' Diagramm neu zeichnen
Plot()
End Sub
```

Listing 533: Auf Auswahl einer Einheit reagieren (Forts.)

**Tipp**

Aus Gründen der Performance kann es sinnvoll sein, die Typsicherheit der Daten für die Kurvendarstellung etwas einzuschränken. Schließlich muss die Datenquelle für jeden abgefragten Wert eine Instanz der entsprechenden `Measurement`-Klasse erzeugen, die vom Garbage-Collector später wieder entsorgt werden muss. Für das Zeichnen der Kurven sind die zusätzlichen Informationen aber nicht notwendig.

Sie können daher das Interface `IDataProvider` erweitern und eine Methode vorsehen, die die Werte als bloßen `Double`-Wert zurückgibt, den Sie auch mit `MeasurementBase.Value` erhalten würden. Im Diagramm müssen Sie die Typ-Informationen der Datenquellen nur einmal einlesen und darauf basierend die Skalierung vornehmen.

## 293 Kontinuierliches T-Y-Diagramm mit dynamischen Werten

Klassische Linienschreiber zeichnen Kurven auf kontinuierlich transportiertem Endlospapier. Übersetzt auf eine Fensterdarstellung bedeutet dies, dass die Zeitachse schrittweise verschoben wird. Das T/Y-Diagramm aus dem vorherigen Beispiel lässt sich für eine solche Darstellung erweitern.

Im kontinuierlichen Betrieb muss der Zeichenbereich zyklisch aktualisiert werden. Die Zeitachse verschiebt sich dadurch nach links, so dass die zuvor gezeichneten Kurven auch allesamt nach links verschoben werden müssen. Abbildung 264 zeigt das Diagramm im kontinuierlichen Betrieb. Als zusätzliche Datenquelle wurde eine leicht modifizierte Sinusfunktion (`SinusSample2`) definiert, die durch Multiplikation zweier Sinuskurven unterschiedlicher Frequenz eine Modulationskurve erzeugt. Diese `IDataProvider`-Implementierung ist hier nicht als Listing abgebildet, sie finden Sie aber auf der Buch-CD.

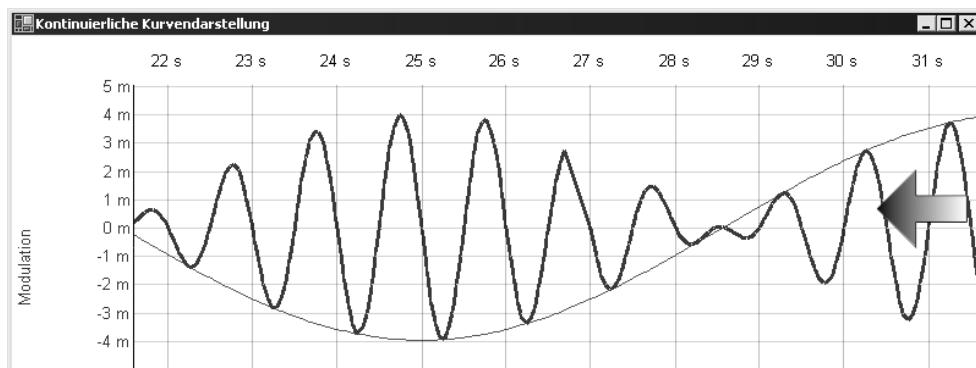


Abbildung 264: Kontinuierliche Kurvendarstellung

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

Betrachten wir zunächst die Initialisierung des Diagramms. Für den kontinuierlichen Betrieb werden einige Member-Variablen dem Steuerelement hinzugefügt:

---

```
' Zeitpunkt, ab dem neu gezeichnet werden muss
Protected TS As TimeMeasures

' Zeitauflösung pro Datenpunkt
Protected DeltaT As Double

' Abweichung durch Rundungsfehler
Protected ShiftingError As Double = 0

' Skalierung Zeitachse
Protected TScaleInfo As ScaleInfo

' Skalierung y-Achse
Protected YScaleInfo As ScaleInfo

' Zeit der letzten Zeitskalierung
Protected LastScaleTime As Double
```

---

Die öffentliche Methode `ShowContinuousDiagram` ist der Einstiegspunkt für den Start der Darstellung. Sie zeigt den bislang verborgenen Menüpunkt `MISartStop` des Kontextmenüs an, über den die kontinuierliche Darstellung angehalten und fortgeführt werden kann. Anschließend wird die geschützte Methode `PlotConinuously` aufgerufen. Listing 534 zeigt beide Methoden.

In `PlotConinuously` wird je nach Parameterwert der für den zyklischen Betrieb verwendete `Time TMCont` gestartet oder gestoppt. Die Zeitdifferenz aus  $T_0$  und  $T_n$  ergibt die Gesamtzeit `DeltaT`, die die Breite der Zeitachse ausmacht. Alle Zeiten werden neu berechnet. Hinzu kommt die Zeit `TS`, die den Zeitpunkt markiert, ab dem die Kurven neu gezeichnet werden müssen (siehe auch Abbildung 265). Zunächst ist `TS` jedoch gleich  $T_n$ .

`ShowDiagram` wird aufgerufen, um den Anfangszustand des Diagramms herzustellen. Die Datenquelle muss hierfür so beschaffen sein, dass sie auch historische Werte (z. B. aus einer Datenbank) bereitstellen kann.

Zuletzt wird das erforderliche Timer-Intervall eingestellt und der Timer gestartet. Alle weiteren Zeichenoperationen finden dann im Event-Handler des Timers (`TMCont_Tick`) statt.

---

```
Public Sub ShowContinuousDiagram()

    ' Kontextmenü "Start/Stop" sichtbar machen
    MISartStop.Visible = True

    ' Diagrammdarstellung starten
    PlotContinuously(True)

End Sub

Protected Sub PlotContinuously(ByVal start As Boolean)
```

---

Listing 534: Öffentliche und geschützte Methoden zur Einleitung der kontinuierlichen Diagrammdarstellung

```

' Bei "Stop" Timer anhalten
If Not start Then
    TMCont.Enabled = False
    Exit Sub
End If

' Zeitdifferenz zu Beginn bleibt konstant für alle
' folgenden Darstellungen
DeltaT = Tn.Value - T0.Value

' Anfangszeiten für Ausgangssituation
' Rechts (Tn) ist die aktuelle Zeit
' Links (T0) ist die aktuelle Zeit - DeltaT
Tn = New TimeMeasures()
TS = Tn
T0 = New TimeMeasures(Tn.Value - DeltaT)

' Grafik löschen
Graphics.FromImage(PicBuf).Clear(Color.White)

' Diagramm mit historischen Daten anzeigen
ShowDiagram()

' Timer-Intervall berechnen und Timer starten
TMCont.Interval = CInt(DeltaT / nPoints * 1000)
TMCont.Enabled = True

End Sub

```

*Listing 534: Öffentliche und geschützte Methoden zur Einleitung der kontinuierlichen Diagrammdarstellung (Forts.)*

Im ersten Schritt im Timer-Event müssen die Zeiten und Koordinaten berechnet werden. Listing 535 zeigt die Implementierung, in Abbildung 265 werden die wichtigsten Variablen erläutert. Bekanntermaßen kann man sich nicht auf die Einhaltung des Timer-Intervalls verlassen, so dass die neuen Zeiten mit Hilfe der Systemzeit ermittelt werden müssen. Nach Berechnung der Zeitpunkte werden die zugehörigen x-Koordinaten ermittelt. Da die Zeit, um die das Diagramm verschoben wird, auf die Bildpunkte des Graphics-Objektes abgebildet werden muss, ergeben sich zwangsläufig Rundungsfehler. Diese Fehler werden in der Variablen `ShiftingError` aufaddiert und laufend bei der Berechnung von  $x_2$  berücksichtigt und ausgeglichen.

Engpass in Hinblick auf Performance ist bei einer kontinuierlichen Kurvendarstellung die Art und Weise, wie in jedem Zyklus das Diagramm neu gezeichnet wird. Eine einfach zu realisierende Lösung besteht darin, die Zeiten `T0` und `Tn` neu zu setzen und `ShowDiagram` aufzurufen. Dann würde das gesamte Diagramm neu gezeichnet. U.U. ist das auch die schnellste Lösung, vorausgesetzt die Datenquellen liefern schnell die zu den jeweiligen Zeitpunkten angefragten Daten. Optimieren lässt sich die Darstellung dadurch, dass man die einmal ermittelten Daten in einer Liste oder einem Array ablegt und auf diese gespeicherten Daten zurückgreift. In jedem Zyklus müssen dann die alten, nicht mehr benötigten Daten aus der Liste entfernt und neu am Ende angehängt werden. Es bleibt jedoch der Aufwand zum Zeichnen der einzelnen Liniensegmente.

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

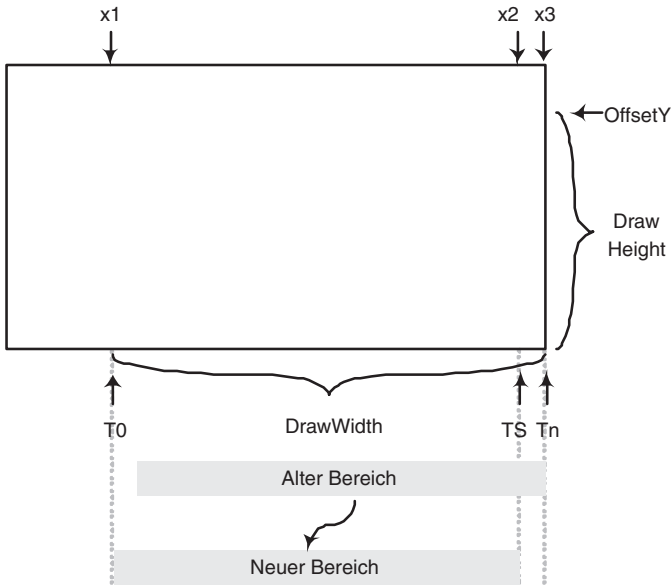
Proper-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

Abbildung 265: Variablen in `TMCont_Tick`

Hier soll eine andere Lösung vorgestellt werden, nämlich das Verschieben des bereits gezeichneten Bildes. Die zu einem Zeitpunkt  $t$  erstellten Kurven sind ja bereits in einem Bitmap (`PicBuf`) gespeichert. Es muss lediglich der Ausschnitt, der bestehen bleiben soll, nach links verschoben werden. Hierfür gibt es wiederum verschiedene Möglichkeiten. Benutzt man API-Funktionen, dann lässt sich eine sehr schnelle Variante mit Hilfe der GDI-Funktion `ScrollDC` programmieren. Leider verlässt man damit den sicheren Bereich des verwalteten .NET-Codes. Insbesondere die Vermischung von GDI- und GDI+-Code birgt viele Gefahren und sollte nach Möglichkeit vermieden werden.

Gefragt ist daher eine reine GDI+-Lösung. GDI+ bietet leider keine direkte Möglichkeit zum Verschieben eines Bitmap-Bereiches an. Stattdessen muss ein zweites temporäres Bitmap angelegt werden, das die Kopie des zu verschiebenden Bereiches aufnimmt (`bmp`). Das Übertragen der Bild-daten mit `DrawImage` ist leider recht zeitintensiv und kostet bei üblichen Fenstergrößen leicht einige zehn Millisekunden. Es kann etwas beschleunigt werden, indem die Eigenschaft `CompositingMode` des `Graphics`-Objektes auf `SourceCopy` gestellt wird. In der Voreinstellung (`SourceOver`) wird mit Alpha-Blending gearbeitet, was die Ausführungszeit unnötig verlängert. Zukünftig werden aber wahrscheinlich die GDI+-Befehle durch Hardware-Beschleunigung wesentlich schneller abgearbeitet, so dass der Zeitaufwand entsprechend geringer ausfallen wird.

---

```
Private Sub TMCont_Tick(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles TMCont.Tick

    ' Zeiten berechnen
    Tn = TS
    TS = New TimeMeasures()
```

---

Listing 535: Zeichnen der kontinuierlichen Darstellung im Timer-Ereignis

```

T0 = New TimeMeasures(-DeltaT + TS.Value)

' Graphics-Objekt ermitteln
Dim g As Graphics = Graphics.FromImage(PicBuf)

' Neue Zeitskalierung zeichnen, falls erforderlich
DrawTAxisNewOnly(g)

' Positionen berechnen
Dim x1 As Integer = OffsetX
Dim dx As Double = DrawWidth * (Tn.Value - T0.Value) / DeltaT

' Rundungsfehler berechnen
ShiftingError += dx - CInt(dx)

' x2 berechnen
' Rundungsfehler berücksichtigen und korrigieren
Dim sherr As Integer
Dim x2 As Integer = OffsetX + CInt(dx)
If Math.Abs(ShiftingError) >= 1 Then
    sherr = CInt(ShiftingError)
    ShiftingError -= sherr
    x2 += sherr
End If

' x3 ist ganz rechts
Dim x3 As Integer = x1 + DrawWidth

' Hilfsrechtecke für Zeichenmethoden
Dim r1 As New Rectangle(x1, 0, x2 - x1 + 1, Height)
Dim r2 As New Rectangle(x1 + x3 - x2, 0, x2 - x1 + 1, Height)

' Temporäres Bitmap
Dim bmp As New Bitmap(x2 - x1 + 1, Height)

' Und das Graphics-Objekt dazu
Dim g2 As Graphics = Graphics.FromImage(bmp)

' Zur Beschleunigung der DrawImage-Methode
g.CompositingMode = _
    Drawing.Drawing2D.CompositingMode.SourceCopy
g2.CompositingMode = _
    Drawing.Drawing2D.CompositingMode.SourceCopy

' Bereich löschen
g2.FillRectangle(Brushes.White, New Rectangle(0, 0, _
    bmp.Width, bmp.Height))

' Temporäres Bitmap mit zu verschiebendem Bereich füllen
g2.DrawImage(PicBuf, New Rectangle(0, 0, bmp.Width, _
    bmp.Height), r2, GraphicsUnit.Pixel)

```

Listing 535: Zeichnen der kontinuierlichen Darstellung im Timer-Ereignis (Forts.)

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Propert-  
yGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

---

```

' Hinteren Zeichenbereich löschen
g.FillRectangle(Brushes.White, x2, 0, x3 - x2 + 1, Height)

' Vershobenen Bereich neu zeichnen
g.DrawImage(bmp, x1, 0)

' Erster Skalenwert
Dim y As Double = YScaleInfo.ScaleStart

' Min/Max bestimmen
Dim max As Double = DirectCast(DataProviderList(DPIDxSkala), _
    TYPlotInfo).Max.Value(DPUnitIdxSkala)
Dim min As Double = DirectCast(DataProviderList(DPIDxSkala), _
    TYPlotInfo).Min.Value(DPUnitIdxSkala)

' Horizontale Linien zeichnen
Do While YScaleInfo.Comparer.IsLessOrEqual(y, max)
    Dim ys As Integer = CInt(DrawHeight * (y - min) / _
        YScaleInfo.Range)
    ' Waagerechte Linien zeichnen
    g.DrawLine(Pens.LightGray, x2, Height - ys, x3, _
        Height - ys)

    ' Nächstes Skalenintervall
    y += YScaleInfo.ScaleInterval
Loop

Dim dpidx As Integer
Dim pinfo As TYPlotInfo

' Für alle Datenquellen
For dpidx = 0 To DataProviderList.Count - 1
    ' Info holen
    pinfo = DirectCast(DataProviderList(dpidx), TYPlotInfo)

    ' Neue Werte ermitteln
    NewValues(dpidx) = pinfo.DP.GetValue(TS)

    ' Koordinaten für neues Liniensegment ermitteln
    Dim h As Double = pinfo.Max.Value - pinfo.Min.Value
    Dim y1 As Integer = CInt(DrawHeight + OffsetY - _
        CSng((-pinfo.Min.Value + LastValues(dpidx).Value) _
            / h * DrawHeight))

    Dim y2 As Integer = CInt(DrawHeight + OffsetY - _
        CSng((-pinfo.Min.Value + NewValues(dpidx).Value) _
            / h * DrawHeight))

    ' Linie zeichnen
    g.DrawLine(pinfo.LinePen, x2, y1, x3, y2)

```

---

```

        ' Letzte Werte für nächsten Zyklus speichern
        LastValues(dpidx) = NewValues(dpidx)
    Next

    ' Wichtig! Ressourcen freigeben
    g2.Dispose()
    bmp.Dispose()

    ' Steuerelement neu zeichnen
    Invalidate()

End Sub

```

Listing 535: Zeichnen der kontinuierlichen Darstellung im Timer-Ereignis (Forts.)

Nachdem die notwendigen Koordinaten errechnet worden sind, wird der Teil des bestehenden Bildes, der erhalten werden soll, in das temporäre Bitmap kopiert. Der zu erneuernde Bereich wird im alten Bild gelöscht und das temporäre Bitmap an Position  $x1$  ( $T0$ ) kopiert. Die Verschiebung ist damit abgeschlossen. Jetzt werden in einer Schleife die Skalierungslinien für den neuen Bereich gezeichnet. Anschließend werden die neuen Kurvensegmente ergänzt.

Das Diagramm ist nun fertig gestellt und steht im Bildpuffer (`PicBuf`) bereit. `Invalidate` leitet das Zeichnen des Steuerelementes ein. Wichtig ist an dieser Stelle noch, dass die Ressourcen, die für das temporäre Bitmap und dessen Graphics-Objekt belegt wurden, mit `Dispose` wieder freigegeben werden.

Abweichend von `DrawTAxis` wird in `DrawTAxisNewOnly` (Listing 536) die Zeitskalierung nur ergänzt, wenn sie bislang fehlt. Da zyklisch das existierende Bild nach links geschoben wird, wandern bereits vorhandene Skalierungen mit. Nur wenn die Lücke des neuen Bereiches breit genug für einen neuen Zeitskalenwert ist, wird dieser einmalig gezeichnet. Da im Gegensatz zu `DrawTAxis` hier die senkrechten Zeitlinien über die bestehenden Kurven gezeichnet werden müssen, werden sie mit einer halbtransparenten Farbe aufgetragen.

```

Protected Sub DrawTAxisNewOnly(ByVal g As Graphics)

    ' Linken Bereich löschen
    g.FillRectangle(Brushes.White, OffsetX - 40, 0, 40, 30)

    ' Erste Zeit für Zeitachsenbeschriftung
    Dim t As Double = LastScaleTime + TScaleInfo.ScaleInterval

    ' Für alle Skalenbeschriftungen
    Do While TScaleInfo.Comparer.IsLessOrEqual(t, Tn.Value)

        ' x-Position berechnen
        Dim x As Integer = Cint(OffsetX + (t - T0.Value) / _
            (Tn.Value - T0.Value) * DrawWidth)

        ' Umschließendes Rechteck für Beschriftung festlegen
        Dim r As New RectangleF(x - 40, 10, 80, 20)
    
```

Listing 536: Ergänzen der Skalierung der Zeitachse

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tung  
Win-  
dows  
Forms

Controls

Proper-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

```

' Wert formatieren
Dim s As String = String.Format(TScaleInfo.FormatString, t)

' Ausgabe nur, wenn das Rechteck vollständig sichtbar ist
If (x + g.MeasureString(s, ScaleFont).Width / 2 < _
    OffsetX + DrawWidth) Then

    ' Text formatiert ausgeben
    g.DrawString(s, ScaleFont, Brushes.Black, r, _
        StringFormat.TAxis)
    ' Senkrechte Linie zeichnen
    g.DrawLine(New Pen(Color.FromArgb(&H40000000)), 0), x, _
        OffsetY, x, Height)

    LastScaleTime = t
End If

' Nächstes Zeitintervall
t += TScaleInfo.ScaleInterval

Loop

End Sub

```

*Listing 536: Ergänzen der Skalierung der Zeitachse (Forts.)*

## Erweiterungen der Diagramme

Sicher erfüllen die beiden vorgestellten Diagramm-Steuerelemente nicht alle Wünsche und Bedürfnisse. Sie sollen als Beispiel dienen und einige Techniken der Diagrammerstellung aufzeigen. Ein Diagrammpaket, das wie z.B. Microsoft Excel eine Vielzahl von Diagrammtypen beinhaltet, würde ganz sicher den Rahmen dieses Buches sprengen.

Viele Erweiterungen der beiden T-Y-Diagramme sind aber relativ leicht möglich. Beispielsweise eine logarithmische Skalierung der Y-Achse oder eine vertikale Trennung der verschiedenen Kurven lassen sich mit wenig Aufwand realisieren.

## 294 Die Zahl $\pi$

Eine der geheimnisvollsten Zahlen in der Mathematik kommt indirekt jeden Tag in unserem Leben vor, die Zahl  $\pi$  (griechischer Buchstabe, gesprochen »pi«). Immer wenn es um Kreise, Kugeln, Zylinder, Kegel und Ähnliches geht, spielt diese Zahl eine Rolle, auch wenn wir es nicht merken. Die Zahl  $\pi$  kann als das Verhältnis von Umfang zu Durchmesser eines Kreises definiert werden. Wenn Sie den Durchmesser eines Baumes bestimmen wollen, brauchen Sie diesen Baum wegen der Zahl  $\pi$  nicht zu zersägen, es reicht, wenn Sie den Umfang mit einem Zentimetermaß bestimmen. Teilen Sie den Umfang durch die Zahl  $\pi$  und Sie haben den Durchmesser des Baumes. Nun dauert die Division unendlich lange, da die Zahl unendlich lang ist. Eine gewisse Ungenauigkeit in der millionsten Stelle nach dem Komma müssen Sie schon hinnehmen ☺.

VB.Net bringt zwei Möglichkeiten mit, für solche Berechnungen  $\pi$  in ausreichender Genauigkeit zu bestimmen. Im Namensraum `Math` ist die Konstante als Eigenschaft `PI` hinterlegt. Eine andere Möglichkeit ist die Berechnung mit der trigonometrischen Funktion `Arcus Tangens (Atan)`. Diese Möglichkeiten sind in Abbildung 266 und in Listing 537 zu sehen.



Eine sehr schnelle, aber nicht so genaue Berechnung mit Integer-Werten liefert die Division  $355/113 = 3,1415929\dots$  Der »richtige« Wert lautet  $3,1415926\dots$

```
Private Sub btnStart_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnStart.Click

    txtATAN.Text = 4 * Atan(1)
    txtPINet.Text = Math.PI.ToString
    txtPI.Text = CreatePI()

End Sub
```

Listing 537: Die Zahl  $\pi$  mit VB-Mitteln

Eine Methode,  $\pi$  auf mehr Stellen hinter dem Komma zu berechnen, findet sich in Listing 538. Die dort vorgestellte Funktion `CreatePI` berechnet diese Zahl auf 2400 Stellen hinter dem Komma genau. Das Originalprogramm ist in C geschrieben und findet sich unter anderem auf <http://www.mathematik.uni-bielefeld.de/infoboerse/wwwboard/messages/115.html>. Eine Suche bei Google liefert aber etwas mehr als 700 Verweise auf dieses C-Programm.

```
Public Function CreatePI() As String
    Dim Pi As String = ""
    Dim TmpString As String
    Dim Tmp1 As String
    Dim Tmp2 As String
    Dim a As Long = 10000
    Dim b As Long = 0
    Dim c As Long = 8400
    Dim d As Long
    Dim e As Long
    Dim f(8401) As Long
    Dim g As Long
    Dim mLen As Integer

    While (b <> c)
        f(b) = a \ 5
        b += 1
    End While

    While (c > 0)
        g = 2 * c
        d = 0
        b = c
        While (b > 0)
            d += (f(b) * a)
            g -= 1
            f(b) = d Mod g
            d \= g
            g -= 1
            b -= 1
        End While
    End While

    Return Pi
```

Listing 538: Berechnung von  $\pi$

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

```

If b <> 0 Then
    d *= b
End If
End While
c -= 14
TmpString = Convert.ToString(e + d \ a)
mLen = TmpString.Length
If mLen < 4 Then
    TmpString += ("0000" + TmpString)
End If
Tmp1 = VisualBasic.Left(TmpString, 5 - mLen)
Tmp2 = VisualBasic.Right(TmpString, mLen - 1)
Pi += (Tmp1 + Tmp2)
e = d Mod a
End While

Pi = VisualBasic.Left(Pi, 1) + "." + VisualBasic.Mid(Pi, 2)
Return Pi
End Function

```

Listing 538: Berechnung von Pi (Forts.)

Das Ergebnis dieser Berechnung ist ebenfalls in Abbildung 266 zu sehen.

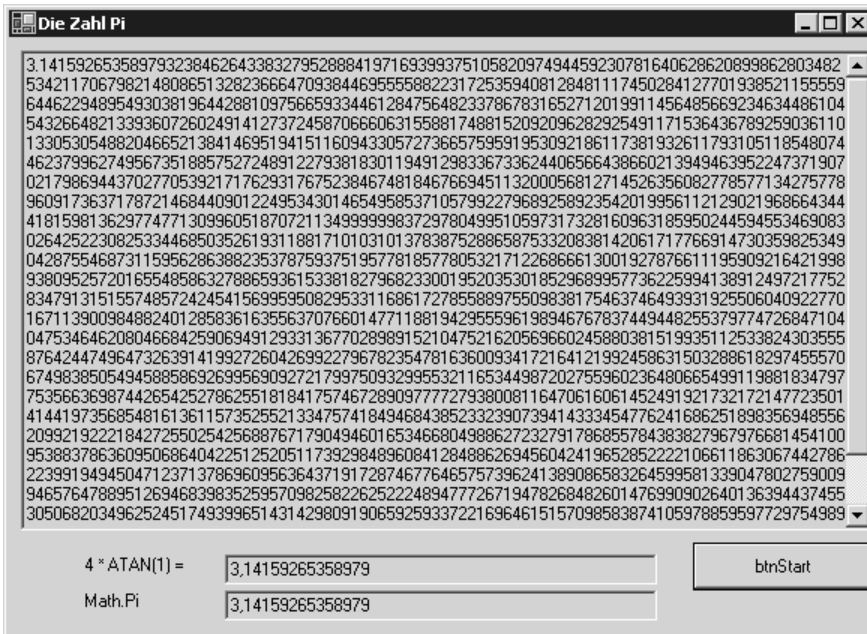


Abbildung 266: Die Zahl Pi aus drei Versionen

# Verschiedenes

Hier finden Sie all die Rezepte, die uns für die Kategorie Basics zu komplex erscheinen, sich nicht den anderen Kategorien zuordnen lassen und für eine eigene Kategorie nicht umfangreich genug sind. Dazu gehören beispielsweise die Rezepte zu Sound-Ausgaben und zu diversen Debugging-Vorgehensweisen sowie Fehlerbehandlungen.

## 295 Sound abspielen über API-Funktionen

Nahezu jeder moderne PC besitzt heute eine Sound-Karte, um Geräusche beliebiger Art über die angeschlossenen Lautsprecher von sich geben zu können. Oft wird daher nach einer Möglichkeit gefragt, gezielt vom Programm aus Sound-Dateien abspielen zu können. Im Framework wurde hierfür bislang nichts vorgesehen. Die beiden heute üblichen Varianten für die Ausgabe von Geräuschen sind:

- ▶ API-Funktionen `PlaySound` und `sndPlaySound`
- ▶ DirectX / DirectSound

Dieses Rezept beschäftigt sich mit der ersten Variante. Windows stellt die beiden genannten Funktionen zur Verfügung, um Sound-Dateien und -Ressourcen abzuspielen. Während `PlaySound` eine Reihe selten benötigter Einstellungen zulässt, beschränkt sich `sndPlaySound` auf die üblicherweise genutzten Steuerungen. Wir wollen uns daher auf die Anwendung dieser Methode beschränken. Listing 539 zeigt die API-Funktion mit ihren möglichen Flags.

```
Public Declare Auto Function sndPlaySound Lib "Winmm.dll" _
    (ByVal path As String, ByVal fuSound As SNDPlaySoundConstants) _
    As Boolean

<Flags(>> Public Enum SNDPlaySoundConstants
    Async = 1
    [Loop] = 8
    Memory = 4
    NoDefault = 2
    NoStop = &H10
    Sync = 0
End Enum
```

Listing 539: API-Funktion zur Sound-Ausgabe und Enumeration zur Steuerung derselben

Das `Async`-Flag steuert, ob der Aufruf der Methode sofort beendet wird oder erst nachdem die Datei vollständig abgespielt worden ist. Für die kontinuierliche Ausgabe in einer Schleife kann das `Loop`-Flag gesetzt werden. In diesem Fall muss auch das `Async`-Flag gesetzt werden, da die Methode sonst endlos lange warten würde (und weil es so in der Doku steht ;-)).

Klänge müssen nicht zwangsläufig in Dateien gespeichert sein, sondern können auch aus dem Speicher geladen werden. Hierfür dient das `Memory`-Flag, das jedoch hier nicht näher betrachtet wird. Mit `NoDefault` lässt sich steuern, ob im Fehlerfall ein Standard-Sound ausgegeben werden soll oder nicht.

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung  
Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schie-  
denes

Wird bereits ein Sound abgespielt, steuert das `NoStop`-Flag, was bei einem weiteren Aufruf von `sndPlaySound` passieren soll. Ist das Flag gesetzt, wird der zweite Aufruf ignoriert, ist es nicht gesetzt, wird die laufende Sound-Ausgabe abgebrochen.

Eine laufende Geräuschkulisse lässt sich abbrechen, indem `sndPlaySound` ohne Angabe eines Pfades (`Nothing`) aufgerufen wird.

Für den einfacheren Umgang mit den API-Funktionen haben wir für Sie Wrapper-Funktionen bereitgestellt. Listing 541 zeigt zwei Methoden, um Sound-Dateien abzuspielen (`PlaySound`) und eine, um die laufende Ausgabe abzubrechen (`StopSound`). Die erste Überladung von `PlaySound` spielt die angegebene Datei ein einziges Mal asynchron ab, die zweite erlaubt die Steuerung über mehrere boolesche Parameter.

---

```
Public Class ApiVbNet

    ...
    Public Shared Sub StopSound()
        ' Soundausgabe sofort stoppen
        API.sndPlaySound(Nothing, API.SNDPlaySoundConstants.Sync)
    End Sub

    Public Shared Sub PlaySound(ByVal soundFile As String)
        ' Sounddatei einmal synchron abspielen
        API.sndPlaySound(soundFile, API.SNDPlaySoundConstants.Sync)
    End Sub

    Public Shared Sub PlaySound(ByVal soundFile As String, _
        ByVal async As Boolean, ByVal [Loop] As Boolean, _
        ByVal noStop As Boolean, ByVal noDefault As Boolean)

        ' Flags zusammensetzen
        Dim flags As API.SNDPlaySoundConstants
        If async Or [Loop] Then flags = API.SNDPlaySoundConstants.Async
        If [Loop] Then flags = flags Or API.SNDPlaySoundConstants.Loop
        If noStop Then flags = flags Or API.SNDPlaySoundConstants.NoStop
        If noDefault Then flags = flags Or
            API.SNDPlaySoundConstants.NoDefault

        ' Sound abspielen
        API.sndPlaySound(soundFile, flags)

    End Sub

End Class
```

---

Listing 540: Wrapper-Funktionen zum Abspielen von Wave-Dateien mit `sndPlaySound`

Ein kleines Beispielprogramm (Abbildung 267) verdeutlicht die Anwendung der Methoden. Das Abspielen wird durch Aufruf von `PlaySound` gestartet. Hierbei werden die Zustände der Check-Boxen als Parameter übergeben:

```
ApiVBNNet.PlaySound(TBSoundFile.Text, CBAsync.Checked, _
    CBLoop.Checked, Not CBStop.Checked, Not CBDefault.Checked)
```

Im Click-Handler der Schaltfläche SOUND STOPPEN wird einzig und allein `StopSound` aufgerufen:

```
ApiVBNNet.StopSound()
```

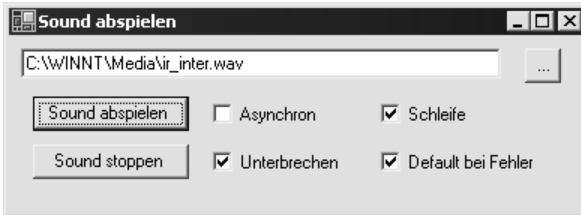


Abbildung 267: Abspielen und Stoppen von Wave-Dateien mit den Wrapper-Funktionen

## 296 Sound abspielen über DirectX

Mit der Einführung von DirectX Version 9 wurden Klassenbibliotheken bereitgestellt, die die Verwendung von DirectX mit managed Code erlauben. Um diese verwenden zu können, müssen Sie das DirectX SDK auf dem Entwicklungsrechner installiert haben. In der aktuellen Version (Sommer 2003) wird nun auch die Hilfe automatisch richtig installiert, so dass sie über die Visual Studio Online-Hilfe verfügbar ist. Die notwendigen Downloads finden Sie im Microsoft Download Center. Einen Überblick über DirectX Downloads bietet Microsoft unter folgender Adresse an:

<http://msdn.microsoft.com/library/default.asp?url=/downloads/list/directx.asp>

Die Leistungsfähigkeiten von DirectX gehen weit über die Geräuscherzeugung mit DirectSound hinaus und erstrecken sich über viele Multi-Media-Bereiche, von Audio- und Video-Ausgaben über komplexe 3D-Graphiken bis in die Welt moderner Computer-Spiele. Die Themenliste reicht wahrscheinlich für mehrere Bücher. Wir wollen daher nur ein kleines Beispiel geben, das Sie zu weiteren Experimenten anregen soll. Mit dem DirectX-SDK wird die Applikation *DirectX Sample Browser* installiert, die sehr übersichtlich nach Themenbereichen gegliedert für ausgewählte Programmiersprachen Tutorials, Code-Beispiele und Demo-Programme verfügbar macht.

Für die Sound-Ausgabe via DirectX benötigen Sie Verweise auf die Klassenbibliotheken *Microsoft.DirectX.dll* und *Microsoft.DirectX.DirectSound.dll*. Die erforderlichen Methoden gehören zum Namensraum *Microsoft.DirectX.DirectSound*.

In Listing 543 sehen Sie die wichtigsten Code-Fragmente für das Abspielen von Wave-Dateien, Abbildung 268 zeigt die Oberfläche des Demo-Programms. Ein `Buffer` vom Typ `SecondaryBuffer` ist das zentrale Objekt, das zur Steuerung der Ausgabe dient. Bei der Instanziierung wird dem Konstruktor der Pfad der zu ladenden Wave-Datei mitgegeben. Über Flags (`BufferPlayFlags`) lässt sich, ähnlich wie bei der API-Funktion `sndPlaySound`, die Ausgabe parametrieren. Im Beispiel wird nur das Flag `Looping` benutzt, das angibt, ob die Ausgabe zyklisch wiederholt werden soll oder nicht.

Durch Aufruf der Methode `Play` des `SecondaryBuffer`-Objektes wird die Sound-Ausgabe gestartet, mit der Methode `Stop` wieder beendet. Sie können mehrere Instanzen von `SecondaryBuffer` anlegen und so mehrere Wave-Dateien gleichzeitig abspielen und steuern. Im Beispielprogramm

können Sie das nachvollziehen, indem Sie mehrmals hintereinander auf die Start-Schaltfläche klicken. Bei jedem Klick wird eine neue Instanz angelegt und der zugehörige Wave-Stream abgespielt. Das Beispielprogramm hat allerdings nur Zugriff auf die zuletzt angelegte Instanz. Einmal gestartete Endlos-Ausgaben können daher nach Start einer weiteren Sound-Ausgabe nicht mehr gestoppt werden. Um das zu erreichen, müssen Sie die Referenzen aller existierenden Buffer in einem Array oder einer Liste nachhalten.

---

```
Imports Microsoft.DirectX.DirectSound
...
Private ApplicationBuffer As SecondaryBuffer = Nothing
Private ApplicationDevice As Device = Nothing

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

    ' DirectX initialisieren
    ApplicationDevice = New Device
    ApplicationDevice.SetCooperativeLevel(Me, _
        CooperativeLevel.Priority)

End Sub

Private Sub BTNSoundAPI_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles BTNSoundAPI.Click

    Try
        ' Wave-Datei laden
        ApplicationBuffer = New SecondaryBuffer(TBSoundFile.Text, _
            ApplicationDevice)

        ' Flag setzen
        Dim bpf As BufferPlayFlags
        If CBLoop.Checked Then
            bpf = BufferPlayFlags.Looping
        Else
            bpf = BufferPlayFlags.Default
        End If

        ' Sound abspielen
        ApplicationBuffer.Play(0, bpf)

    Catch ex As SoundException
        Debug.WriteLine(ex.Message)
    End Try

End Sub

Private Sub BTNStopSound_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles BTNStopSound.Click

    ' Geräuscherzeugung beenden
```

---

```
If (Not Nothing Is ApplicationBuffer) Then
    ApplicationBuffer.Stop()
End If

End Sub
```

Listing 541: Die wesentlichen Code-Schnipsel für die Sound-Ausgabe mit DirectX (Forts.)

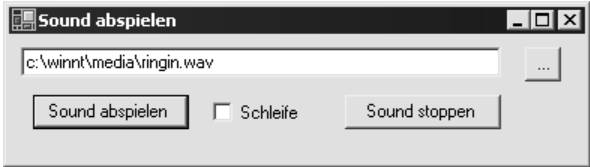


Abbildung 268: Demo-Programm für Sound-Ausgabe mit DirectX

## 297 Trace- und Debug-Ausgaben über Config-Datei steuern

Kontrollausgaben mit `Trace.WriteLine` oder `Debug.WriteLine` erleichtern oft immens die Fehlersuche, gestatten sie doch nachzuverfolgen, welche Programmzweige in welcher Reihenfolge durchlaufen worden sind und welche Werte dabei die wichtigsten Variablen gehabt haben. Spätestens jedoch wenn die erste Version beim Kunden installiert wird und nicht mehr in der Entwicklungsumgebung läuft und deshalb die Trace-Ausgaben umgeleitet werden müssen, entsteht der Wunsch, diese je nach Wichtigkeit an- oder abschalten zu können. .NET bietet hierfür einen Ansatz, über die Konfigurationsdatei der Anwendung einen Level vorzugeben und diesen bei der Ausgabe von Trace-Meldungen zu berücksichtigen.

In einer `.config`-Datei wie in Listing 545 können Sie innerhalb des `<switches>`-Knotens Schalter definieren und diesen einen Level zuordnen. Bedeutung und Enumerationskonstanten hierfür zeigt Tabelle 43.

| TraceLevel-Konstante | Wert | Bedeutung                 |
|----------------------|------|---------------------------|
| Off                  | 0    | keine Ausgaben            |
| Error                | 1    | nur Fehlermeldungen       |
| Warning              | 2    | zusätzlich Warnungen      |
| Info                 | 3    | zusätzliche Informationen |
| Verbose              | 4    | alle Meldungen            |

Tabelle 43: Level zur Steuerung von Trace-Ausgaben

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>

    <system.diagnostics>
        <switches>
```

Listing 542: Konfigurationsdatei mit Schalter für die Trace-Ausgaben

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

```

        <add name="MainSwitch" value="4" />
    </switches>
</system.diagnostics>

</configuration>

```

---

*Listing 542: Konfigurationsdatei mit Schalter für die Trace-Ausgaben (Forts.)*

Um die Einstellung des Schalters aus der Konfigurationsdatei berücksichtigen zu können, benötigen Sie eine Instanz der Klasse `TraceSwitch`. Der erste Parameter, der dem Konstruktor übergeben wird, ist der Name des Schalters. Für die angegebene Konfigurationsdatei wird die Instanz mit der folgenden Anweisung angelegt:

---

```
Public TSW As New TraceSwitch("MainSwitch", "Wichtige Meldungen")
```

---

Den vorgegebenen Trace-Level können Sie dann über die Eigenschaft `Level` abfragen. Zur Vereinfachung stellt `TraceSwitch` die schreibgeschützten Eigenschaften `TraceError`, `TraceWarning`, `TraceInfo` und `TraceVerbose` bereit, so dass Sie ohne Zahlenvergleiche direkt entscheiden können, ob eine Trace-Ausgabe erfolgen soll oder nicht. Auch eine If-Abfrage ist nicht notwendig, da Sie statt `Trace.WriteLine` die Methode `Trace.WriteLineIf` verwenden können, die diese Abfrage bereits vorsieht. Folgender Beispiel-Code soll die Zusammenhänge erläutern:

---

```

Trace.WriteLine("TraceLevel: " & TSW.Level.ToString())
Trace.WriteLineIf(TSW.TraceError, "Fehler")
Trace.WriteLineIf(TSW.TraceInfo, "Info")
Trace.WriteLineIf(TSW.TraceVerbose, "Ausführlich")
Trace.WriteLineIf(TSW.TraceWarning, "Warnung")

```

---

Die erste Zeile gibt den eingestellten Level aus und die nachfolgenden Zeilen jeweils eine Meldung, sofern der entsprechende Level eingeschaltet ist. Das Ergebnis für den Level 4, wie er in Listing 545 definiert ist, sieht dann so aus:

---

```

TraceLevel: Verbose
Fehler
Info
Ausführlich
Warnung

```

---

Wird der Level in der Konfigurationsdatei vor Programmstart auf 1 gestellt, ergibt sich diese Ausgabe:

---

```

TraceLevel: Error
Fehler

```

---

So kann ein Anwender steuern, ob und wenn ja welche Informationen ausgegeben werden sollen. Im Normalfall würde z.B. der Wert 0 alle Fehlermeldungen unterdrücken. Kommt es zu abnormalem Programmverhalten, kann zur Fehlersuche der Level entsprechend hochgesetzt werden. Zur Laufzeit können Sie den Trace-Level nur ändern, indem Sie der Eigenschaft `Level` einen anderen Wert zuweisen. Änderungen in der Konfigurationsdatei werden erst beim nächsten Start berücksichtigt.



## 298 Debug- und Trace-Ausgaben an eine TextBox weiterleiten

Steht die Entwicklungsumgebung nicht zur Verfügung oder läuft das Programm nicht im Debug-Modus, dann entfällt die Möglichkeit, Meldungen in das Ausgabe-Fenster der IDE auszugeben. Ausgaben z.B. mit `Trace.WriteLine` laufen dann ins Leere, es sei denn, man leitet sie an einen anderen Zuhörer weiter. Das Framework sieht hierfür das Konzept des `TraceListener`s vor. Eine `TraceListener`-Instanz kann alle Debug- und Trace-Ausgaben empfangen, wenn sie der `Listeners`-Auflistung der `Trace`-Klasse zugeordnet wird. Die Klassen `Debug` und `Trace` teilen sich eine gemeinsame `Listeners`-Auflistung. `TraceListener` ist eine abstrakte (`MustInherit`) Basisklasse, von der Sie eine neue Klasse ableiten müssen oder eine der existierenden Ableitungen (`EventLogTraceListener` oder `TextWriterTraceListener`, siehe nachfolgende Rezepte) verwenden können.

Um die Ausgaben in eine `TextBox` umleiten zu können, wird die Klasse `TraceToTextBox` definiert (Listing 547). Die Ableitung von `TraceListener` erfordert das Überschreiben der Methoden `Write` und `WriteLine`. Diese Methoden nehmen als Parameter den auszugebenden Text entgegen und fügen ihn an den Inhalt der `TextBox` an.

Im Falle von `WriteLine` werden zusätzliche Leerzeichen zu Beginn der Zeile eingefügt, falls der Ausgabertext eingerückt werden muss. `Me.IndentLevel` gibt in diesem Fall an, um wie viele Stufen eingerückt werden soll, während `Me.IndentSize` die Anzahl der Leerzeichen pro Stufe vorgibt.

Nach jeder `Write`- oder `WriteLine`-Operation wird die Einfügemarke an das Ende des Textes gesetzt und mit `ScrollToCaret` der gesamte Inhalt so verschoben, dass die letzte Zeile sichtbar ist. Abbildung 269 zeigt ein Beispiel für eine Trace-Ausgabe in eine `TextBox`, die mehrere Einrückungsstufen verwendet.

```
Public Class TraceToTextBox
    Inherits TraceListener

    ' Verknüpfte TextBox
    Protected TraceTextBox As TextBox

    ' Konstruktor übernimmt Referenz der Ausgabe-TextBox
    Public Sub New(ByVal traceTextBox As TextBox)
        Me.TraceTextBox = traceTextBox
        traceTextBox.Text = ""
    End Sub

    ' Ausgabe mit Debug.Write bzw. Trace.Write
    Public Overloads Overrides Sub Write(ByVal message As String)

        ' Text anhängen
        Me.TraceTextBox.Text &= message

        ' Cursor hinter Textende setzen
        Me.TraceTextBox.Select(Me.TraceTextBox.Text.Length, 0)

        ' Letzte Zeile sichtbar machen
        Me.TraceTextBox.ScrollToCaret()
```

Listing 543: `TraceToTextBox` erlaubt die Umleitung der Trace-Ausgaben in eine `TextBox`

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Proper-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

End Sub

```
' Ausgabe mit Debug.WriteLine bzw. Trace.WriteLine
Public Overloads Overrides Sub WriteLine(ByVal message As String)

    ' Leerzeichen für Einrückung berechnen
    Dim spaces As New String(" ", Me.IndentLevel * Me.IndentSize)

    ' Text mit Einrückung anhängen
    Me.TraceTextBox.Text &= spaces & message & Environment.NewLine

    ' Cursor hinter Textende setzen
    Me.TraceTextBox.Select(Me.TraceTextBox.Text.Length, 0)

    ' Letzte Zeile sichtbar machen
    Me.TraceTextBox.ScrollToCaret()
```

End Sub

End Class

Listing 543: *TraceToTextBox* erlaubt die Umleitung der Trace-Ausgaben in eine TextBox (Forts.)

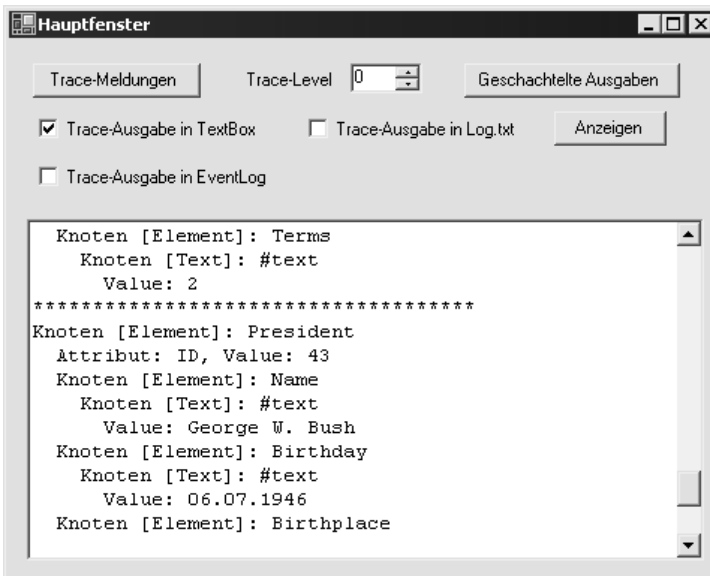


Abbildung 269: *Eingerückte Trace-Ausgabe in einer TextBox*

Die Anzahl der Leerzeichen für eine Einrückungsstufe lässt sich übrigens auch über die Konfigurationsdatei einstellen. Hierfür wird innerhalb des `<system.diagnostics>`-Knoten ein `<trace>`-Knoten definiert und die Anzahl der Leerzeichen im Attribut `indentsize` festgelegt:

```
<trace indentsize="2"></trace>
```

Für die TextBox selbst muss die `MultiLine`-Eigenschaft auf `True` gesetzt und zumindest die vertikale Bildlaufleiste eingeschaltet werden. Als Schriftart empfiehlt sich eine nicht proportionale Schrift, damit die Einrückungen richtig dargestellt werden. In der abgebildeten TextBox wurde die Schriftart `Courier` eingestellt.

Im Beispielprogramm wird eine Member-Variable der Fensterklasse für die Referenz der `TraceListener`-Instanz definiert:

---

```
Public TBLISTENER As TraceToTextBox
```

---

Im Load-Ereignis wird die Instanz angelegt und an die TextBox gebunden:

---

```
TBLISTENER = New TraceToTextBox(TBTraceOutput)
```

---

Ein- und ausgeschaltet wird die Ausgabe in die TextBox im `CheckedChanged`-Ereignis der `CheckBox CBTraceInTB`:

---

```
If CBTraceInTB.Checked Then
    Trace.Listeners.Add(TBLISTENER)
Else
    Trace.Listeners.Remove(TBLISTENER)
End If
```

---

Mit dieser Vorgehensweise können Sie die Trace-Ausgaben auch auf andere Steuerelemente umleiten (z.B. `ListBoxes`, `ListView`s etc.). Bei einer dauerhaften Verwendung für Trace-Ausgaben sollten Sie jedoch berücksichtigen, dass die Steuerelemente nicht beliebig viele Textzeilen verarbeiten können. Insbesondere die `TextBox` hat eine Beschränkung auf ca. 32.000 Zeichen. Die Methoden der `Listener`-Klasse müssen ggf. derart erweitert werden, dass alte Textzeilen wieder entfernt werden.

## 299 Debug- und Trace-Ausgaben in einer Datei speichern

Möchten Sie die Trace-Ausgaben für den Anwender unsichtbar in einer Log-Datei speichern, können Sie eine `TraceListener`-Ableitung verwenden, die alle `Trace.Write`- und `Trace.WriteLine`-Ausgaben in eine Textdatei schreibt. Hierzu müssen Sie selbst keine neue Klasse anlegen, sondern können sich der vom Framework bereitgestellten Klasse `TextWriterTraceListener` bedienen.

Ein Beispiel hierzu: Wieder wird eine Member-Variable definiert, die während der Laufzeit des Programms die Referenz des `Listener`-Objektes hält:

---

```
Public LogfileListener As TextWriterTraceListener
```

---

Im `CheckedChanged`-Ereignis der `CheckBox TRACE-AUSGABE IN LOG.TXT` wird eine Instanz der Klasse angelegt und an die `Listeners`-Auflistung angehängt bzw. wieder entfernt.

---

```
If CBLogListener.Checked Then
    File.Delete("log.txt")
    LogfileListener = New TextWriterTraceListener("Log.txt")
    Trace.Listeners.Add(LogfileListener)
```

---

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Propor-  
tityGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

```
Else  
    Trace.Listeners.Remove(LogfileListener)  
    LogfileListener.Dispose()  
End If
```

Die Ausgabe in die Datei kann parallel zu anderen Ausgaben erfolgen. In Abbildung 270 sehen Sie das Fenster des Beispielprogramms mit den Trace-Ausgaben in der TextBox und in Abbildung 271 die Ansicht der Textdatei, die gleichzeitig geschrieben wurde.

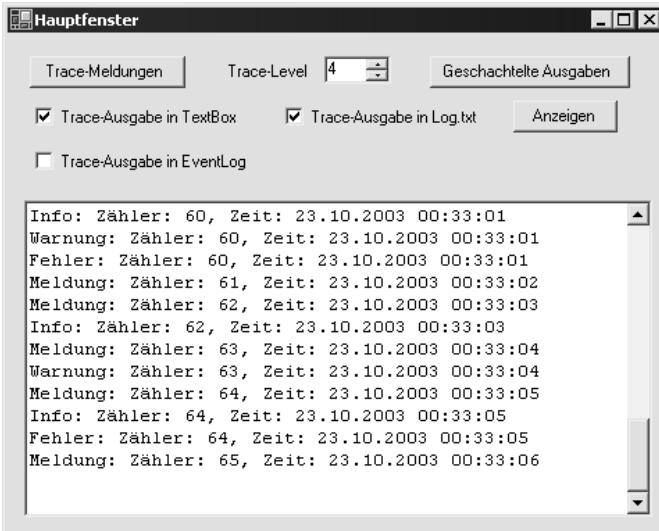


Abbildung 270: Laufende Trace-Ausgaben in der TextBox



Abbildung 271: Laufende Trace-Ausgaben in einer Textdatei

Die Trace-Ausgaben in eine Datei lassen sich auch ganz ohne Code erzeugen. Ein Eintrag in der .config-Datei genügt:

```
<trace autoflush="false" indentsize="2">
  <listeners>
    <add name="myListener2"
        type="System.Diagnostics.TextWriterTraceListener"
        initializeData="c:\temp\Log2.txt" />
  </listeners>
</trace>
```

Im <add>-Knoten innerhalb des <listeners>-Knoten wird ein neuer Listener definiert. Dazu wird im type-Attribut der vollqualifizierte Typname der Klasse TextWriterTraceListener angegeben. Das Attribut initializeData legt den Dateipfad fest. Hier muss offenbar ein absoluter Pfad angegeben werden. Bei Angabe eines relativen Pfades wurde bei unseren Tests die Datei zwar angelegt, blieb jedoch leer.

Innerhalb des <listeners>-Knotens können Sie auch bestehende Listener wieder entfernen. Mit

```
<remove name="Default" />
```

wird z.B. die Ausgabe in das Debug-Fenster unterdrückt.

## 300 Debug- und Trace-Ausgaben an das Eventlog weiterleiten

Mit einem einzigen Handgriff lassen sich die Trace-Ausgaben auch an ein Eventlog weiterleiten. Das Framework bietet hierfür die von TraceListener abgeleitete Klasse EventLogTraceListener an. Sie können eine Instanz dieser Klasse genauso der Listeners-Auflistung hinzufügen, wie in den beiden vorangegangenen Beispielen beschrieben:

```
Public EvLogTL As New EventLogTraceListener(Application.ProductName)
...
If CBEventlog.Checked Then
  Trace.Listeners.Add(EvLogTL)
Else
  Trace.Listeners.Remove(EvLogTL)
End If
```

CBEventlog ist dabei die in Abbildung 270 sichtbare CheckBox mit der Beschriftung TRACE-AUSGABE IN EVENTLOG. Im Konstruktor von EventLogTraceListener können Sie wahlweise die Referenz eines EventLog-Objektes angeben, das genau festlegt, in welches EventLog die Ausgaben umgeleitet werden sollen, oder Sie geben lediglich einen eindeutigen Namen (z.B. den Namen der Anwendung) an. Dann erfolgt die Ausgabe unter diesem Namen im EventLog für Anwendungen (siehe Abbildung 272).

Wie bei der Umleitung in eine Datei lässt sich auch die Trace-Ausgabe in ein EventLog über die Konfigurationsdatei der Anwendung steuern, so dass sie ohne zusätzliche Programmierung ein- oder ausgeschaltet werden kann:

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Proper-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schie-  
denes

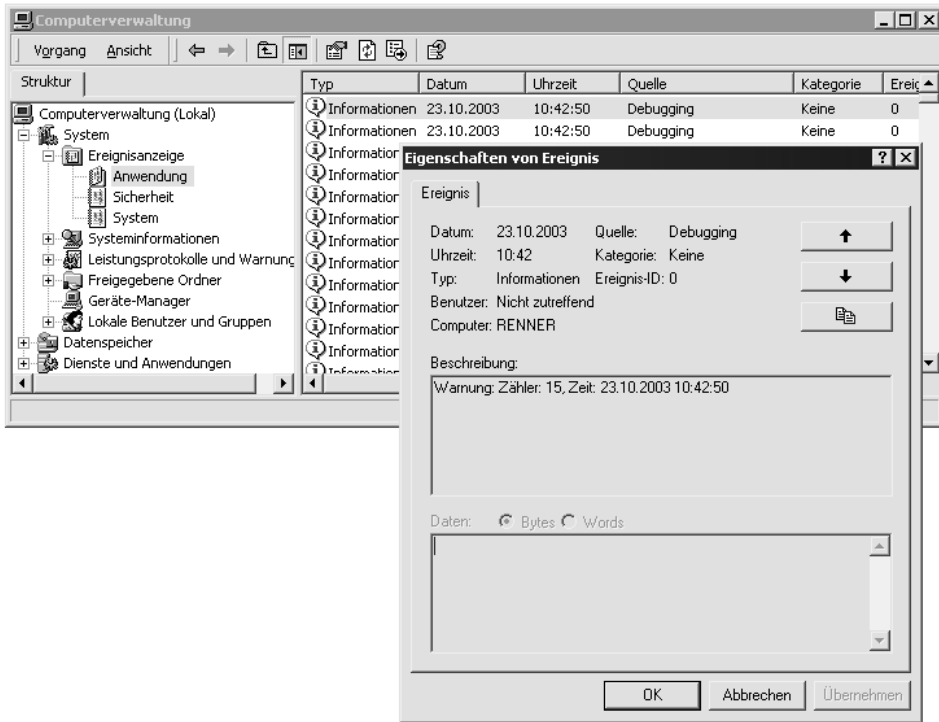


Abbildung 272: Trace-Ausgabe im EventLog des Betriebssystems

```
<trace autoflush="false" indentsize="2">
  <listeners>
    <add name="myListener3"
      type="System.Diagnostics.EventLogTraceListener"
      initializeData="Debuggin-Project" />
  </listeners>
</trace>
```

Im Attribut `initializeData` legen Sie den Namen fest, der im EventLog angezeigt werden soll.

Wenn Sie die Ausgaben, wie in unserem Beispiel, in das Standard-EventLog für Anwendungen schreiben, berücksichtigen Sie bitte, dass sich alle Anwendungen dieses EventLog teilen. Umfangreiche Meldungen Ihrer Anwendung kaschieren vielleicht wichtige Fehlermeldungen anderer Programme. Eventuell ist es daher besser, für die Umleitung der Trace-Ausgaben ein neues EventLog anzulegen.

## 301 Eigene EventLogs für die Ereignisanzeige anlegen und beschreiben

Wenn Sie sich dafür entschieden haben, Informationen und Fehlermeldungen in ein EventLog auszugeben, das Sie mit der Ereignisanzeige der Computerverwaltung analysieren möchten, dann können Sie zur besseren Übersicht ein separates EventLog verwenden. So vermeiden Sie die Überfrachtung des Standard-EventLogs für Anwendungen.

Mit folgendem Code wird ein neues EventLog angelegt, sofern es noch nicht existiert:

```
If Not EventLog.Exists("DebSmp1g") Then
    EventLog.CreateEventSource("DebSmp", "DebSmp1g")
End If
```

Um dieses EventLog zu nutzen, definieren Sie eine Instanz der Klasse EventLog und legen die Log- und die Source-Eigenschaften fest:

```
Public SampleEventlog As EventLog
...
SampleEventlog = New EventLog
SampleEventlog.Log = "DebSmp1g"
SampleEventlog.Source = "DebSmp"
```

Ausgaben in dieses EventLog nehmen Sie am besten mit Hilfe einer der zehn Überladungen der Methode WriteEntry vor. Beachten Sie hierbei jedoch, dass Sie keine Überladung verwenden, die im ersten Parameter die Quelle (Source) annimmt, da die Ausgaben sonst bei abweichender Angabe in einem anderen EventLog landen können.

Listing 549 zeigt exemplarisch, wie innerhalb eines Timer-Events Meldungen in das EventLog DebSmp1g geschrieben werden. Es wurde bewusst die WriteEntry-Methode mit den meisten Überladungen gewählt, um zu zeigen, welche Informationen übergeben werden können und wie sie in der Ereignisanzeige dargestellt werden. Die Meldungen werden mit verschiedenen Kategorietypen versehen (EventLogEntryType.Information, -Warning und -Error), denen in der Ereignisanzeige unterschiedliche Icons zugeordnet werden (siehe Abbildung 273).

```
Public Counter As Integer = 0

Private Sub Timer1_Tick(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles Timer1.Tick

    Counter += 1

    Dim t As String = String.Format("Zähler: {0}, Zeit: {1}", _
        Counter, DateTime.Now)

    ' Jede zweite Sekunde
    If (Counter Mod 2) = 0 Then SampleEventlog.WriteEntry( _
        "Info: " & t, EventLogEntryType.Information, 4711, 55, _
        BitConverter.GetBytes(Counter))

    ' Jede dritte Sekunde
    If (Counter Mod 3) = 0 Then SampleEventlog.WriteEntry( _
        "Warnung: " & t, EventLogEntryType.Warning, 4712, 73, _
        BitConverter.GetBytes(Counter))

    ' Jede vierte Sekunde
```

Listing 544: Gezielte Ausgabe von Meldungen in eine separate Log-Datei

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

```

If (Counter Mod 4) = 0 Then SampleEventlog.WriteEntry( _
    "Error: " & t, EventLogEntryType.Error, 4713, 33, _
    BitConverter.GetBytes(Counter))

```

End Sub

Listing 544: Gezielte Ausgabe von Meldungen in eine separate Log-Datei (Forts.)

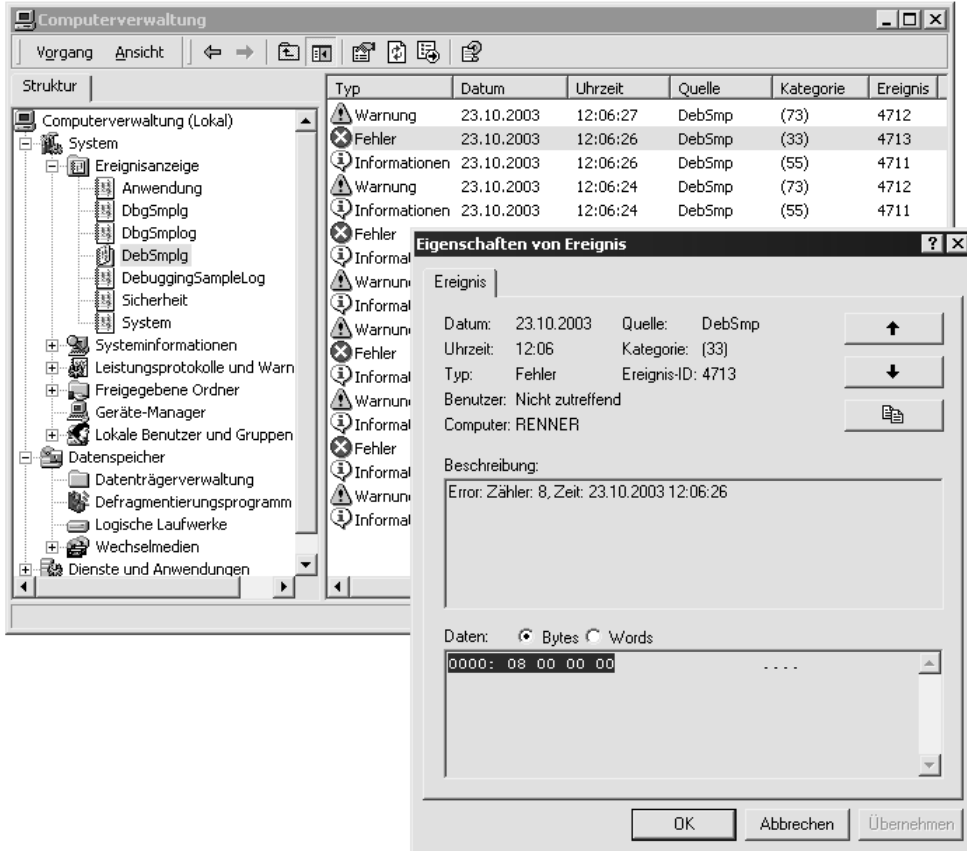


Abbildung 273: Ereignisanzeige für separat erstellte Logdatei

In der abgebildeten Eigenschaftsseite eines Ereignisses finden Sie die verschiedenen Informationen, die beim Aufruf von `WriteEntry` übergeben wurden, wieder. Neben den Zahlenwerten für Kategorie und Ereignis-ID sehen Sie im Datenbereich den Wert des als Byte-Array übergebenen Zählers.

Durch gezieltes Setzen dieser Informationen können Sie Ihre Einträge in ein Eventlog aussagekräftig gestalten und ggf. die Fehlersuche erleichtern. Einige der Daten können Sie in der Ereignisanzeige zum Filtern der Nachrichten nutzen. Abbildung 274 zeigt für das genannte Beispiel, wie nur Fehlermeldungen mit der ID 4713 ausgewählt und alle anderen unterdrückt werden können.



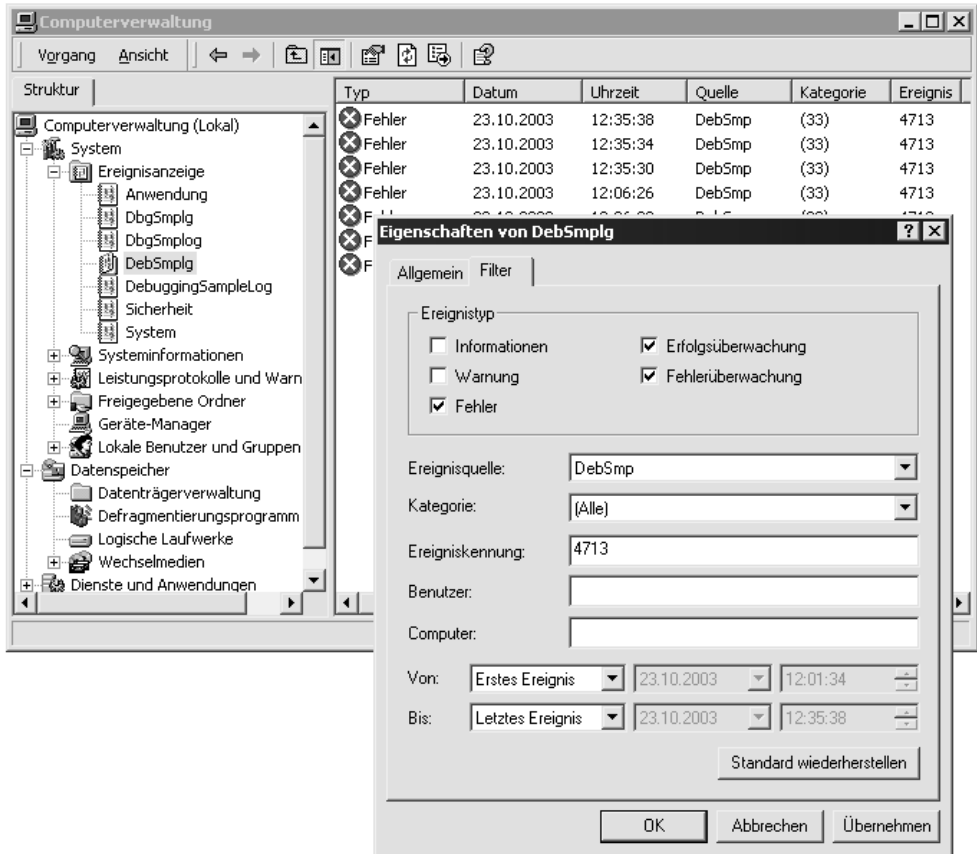


Abbildung 274: Filtern der erzeugten EventLog-Meldungen

## 302 EventLog überwachen und lesen

Die vordefinierten wie auch die selbst erzeugten EventLogs lassen sich auch per Programm abfragen und überwachen. Alles, was Sie benötigen, um auf ein EventLog zugreifen zu können, ist eine Instanz der Klasse `EventLog`. Diese können Sie wie im vorigen Beispiel selbst erzeugen, oder Sie nutzen den Designer und verwenden die in der Toolbox unter **KOMPONENTEN** aufgeführte Komponente `EventLog`. Im Eigenschaftsfenster können Sie dann bequem alle benötigten Einstellungen vornehmen. Dort können Sie z.B. für die Eigenschaft `Log` einen Wert aus der vorgegebenen Liste der verfügbaren EventLogs wählen (s. Abbildung 275).

Programmatisch können Sie die Liste der EventLogs abfragen über die statische Methode `EventLog.GetEventLogs`. Mit dem erhaltenen Array können Sie dem Anwender die verfügbaren Protokolle zur Auswahl anbieten.

Wenn Sie informiert werden möchten, sobald neue Meldungen in das EventLog eingetragen worden sind, dann müssen Sie die Eigenschaft `EnableRaisingEvents` des `EventLog`-Objektes auf `True` setzen. Bei jeder eintreffenden Meldung wird dann das Ereignis `EntryWritten` ausgelöst, auf das Sie reagieren können. Das Feld **LETZTE MELDUNG** im oberen Teil der Abbildung 275 wird in diesem Ereignis beschriftet:

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Propor-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

```
Private Sub EventLog1_EntryWritten(...)
    LBLMessage.Text = e.Entry.Message
End Sub
```

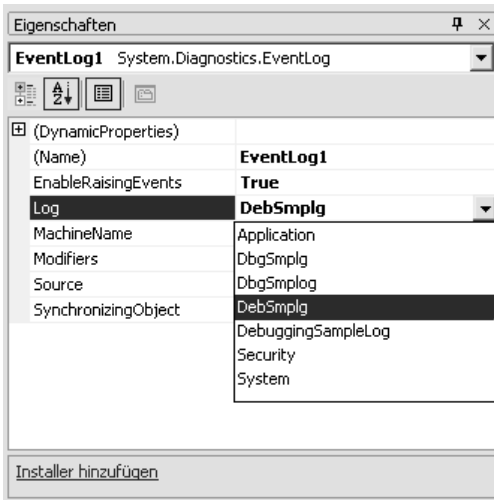


Abbildung 275: Einstellen einer EventLog-Komponente

Über die Eigenschaft `Entries` eines `EventLog`-Objektes erhalten Sie Zugriff auf alle Einträge, die bislang archiviert wurden. In einer Schleife über diese Liste können Sie für jeden Eintrag auf alle Informationen zugreifen. Listing 551 zeigt den Code, mit dessen Hilfe die `ListView` in Abbildung 275 mit den Informationen aus dem `EventLog` gefüllt wird.

```
' Alle Einträge des EventLogs durchlaufen
For Each entry As EventLogEntry In EventLog1.Entries
    ' Eine ListView-Zeile für jeden Eintrag anlegen
    Dim lvi As ListViewItem = LVEventlog.Items.Add( _
        entry.TimeGenerated.ToShortDateString())
    lvi.SubItems.Add(entry.TimeGenerated.ToLongTimeString())
    lvi.SubItems.Add(entry.EntryType.ToString())
    lvi.SubItems.Add(entry.Message)
Next
```

Listing 545: Ausgabe der EventLog-Einträge in eine ListView

## 303 Leistungsindikatoren anlegen und mit Daten versorgen

In Rezept [11.28](#) ((Leistungsüberwachung/Performance Counter)) wurde bereits vorgestellt, wie Sie Leistungsindikatoren (PerformanceCounter) in Ihren Programmen abfragen können. Nicht nur zu Debug-Zwecken ist es aber oft sinnvoll, auch eigene Leistungsindikatoren anzulegen und kontinuierlich mit Informationen zu versorgen. Auch hierfür hält das Framework die notwendige Funktionalität bereit.

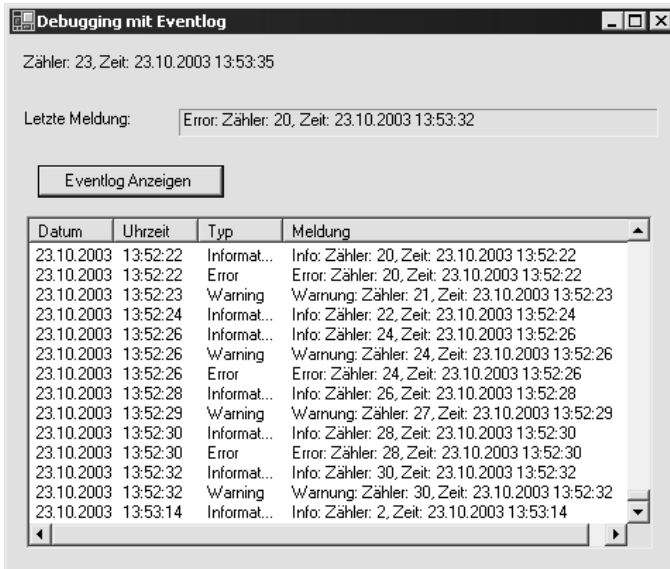


Abbildung 276: Anzeigen aktueller und archivierter Informationen eines EventLogs

Bevor Sie eine neue PerformanceCounter-Kategorie anlegen, sollten Sie mit PerformanceCounter-Category.Exists prüfen, ob diese bereits existiert. Gibt es sie noch nicht, können Sie sie anlegen und beliebige Counter hinzufügen. Für das Anlegen einer neuen PerformanceCounter-Liste wird eine Instanz der Klasse CounterCreationDataCollection benötigt, für jeden Counter eine Instanz von CounterCreationData. Jeder Counter erhält seine individuellen Einstellungen und wird in die CounterCreationDataCollection-Auflistung aufgenommen. Abschließend wird durch den Aufruf von PerformanceCounterCategory.Create eine neue Kategorie angelegt und die CounterCreationDataCollection-Auflistung übergeben.

Im Beispielcode in Listing 551 werden vier PerformanceCounter angelegt. Der erste (ClickFrequency) soll die Anzahl der Klicks pro Sekunde auf eine Schaltfläche angeben, die anderen drei (Zero, One, Two) sollen absolute Zählerstände wiedergeben. Mit der Eigenschaft CounterType wird festgelegt, wie der Counter verwaltet werden soll. Das Betriebssystem stellt hierzu eine Reihe von Möglichkeiten zur Verfügung (Average..., CounterDelta..., CounterTimer... und viele mehr), deren ausführliche Beschreibung Sie in der MSDN-Dokumentation finden. Alle vier Counter werden der Kategorie NeueBeispielkategorie hinzugefügt.

Der Member-Variable FreqCounter und dem Array Counters werden die Referenzen der jeweiligen PerformanceCounter-Objekte zugewiesen, um sie später verwenden zu können. Beachten Sie hierbei, dass dem Konstruktor als dritter Parameter der Wert False übergeben wird. Er legt fest, dass die PerformanceCounter nicht schreibgeschützt sind. Nur dann können Sie auch Werte zuweisen.

Die absoluten Zähler werden zum Abschluss zurückgesetzt. Das geschieht hier durch Zuweisung eines Wertes an die RawValue-Eigenschaft.

```
Public Counters(2) As PerformanceCounter
Public FreqCounter As PerformanceCounter
```

Listing 546: Anlegen einer neuen PerformanceCounter-Kategorie mit vier Countern

---

```

Private Sub MainWindow_Load(ByVal sender ...

If Not PerformanceCounterCategory.Exists( _
    "NeueBeispielkategorie") Then

    ' Counter-Liste vorbereiten
    Dim CCDCol As New CounterCreationDataCollection
    Dim CCData As CounterCreationData

    ' Neuen Counter definieren und parametrieren
    CCData = New CounterCreationData
    CCData.CounterName = "ClickFrequency"
    CCData.CounterHelp = "Anzahl der Klicks pro Sekunde"
    CCData.CounterType = _
        PerformanceCounterType.RateOfCountsPerSecond32

    ' Counter der Liste hinzufügen
    CCDCol.Add(CCData)

    ' Neuen Counter definieren und parametrieren
    CCData = New CounterCreationData
    CCData.CounterName = "Zero"
    CCData.CounterHelp = "Anzahl der Klicks bei Wert 0"
    CCData.CounterType = PerformanceCounterType.NumberOfItems32

    ' Counter der Liste hinzufügen
    CCDCol.Add(CCData)

    ' Neuen Counter definieren und parametrieren
    CCData = New CounterCreationData
    CCData.CounterName = "One"
    CCData.CounterHelp = "Anzahl der Klicks bei Wert 1"
    CCData.CounterType = PerformanceCounterType.NumberOfItems32

    ' Counter der Liste hinzufügen
    CCDCol.Add(CCData)

    ' Neuen Counter definieren und parametrieren
    CCData = New CounterCreationData
    CCData.CounterName = "Two"
    CCData.CounterHelp = "Anzahl der Klicks bei Wert 2"
    CCData.CounterType = PerformanceCounterType.NumberOfItems32

    ' Counter der Liste hinzufügen
    CCDCol.Add(CCData)

    ' Kategorie mit neuen Countern anlegen
    PerformanceCounterCategory.Create("NeueBeispielkategorie", _
        "Beispiel-Counter für VB.NET Codebook", CCDCol)

End If

```

---

```

' Counter-Objekte anlegen
Counters(0) = New PerformanceCounter("NeueBeispielkategorie", _
    "Zero", False)
Counters(1) = New PerformanceCounter("NeueBeispielkategorie", _
    "One", False)
Counters(2) = New PerformanceCounter("NeueBeispielkategorie", _
    "Two", False)
FreqCounter = New PerformanceCounter("NeueBeispielkategorie", _
    "ClickFrequency", False)

' Absolute Zähler zurücksetzen
Counters(0).RawValue = 0
Counters(1).RawValue = 0
Counters(2).RawValue = 0

```

End Sub

Listing 546: Anlegen einer neuen PerformanceCounter-Kategorie mit vier Countern (Forts.)

Das kleine Beispielprogramm (Abbildung 277) zeigt auf einem Label Zufallszahlen zwischen Null und Zwei an, die von einem Timer kontinuierlich erzeugt werden. Bei jedem Klick auf die Schaltfläche PUSH ME werden der Zähler ClickFrequency und der zur aktuell gezogenen Zufallszahl zugeordnete Zähler (Zero, One oder Two) inkrementiert (Listing 551).

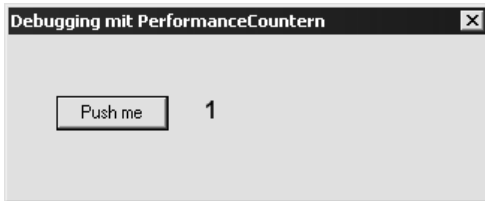


Abbildung 277: Testen Sie Ihre Reaktion und die Beweglichkeit Ihrer Finger

```

Private Sub BTNPushMe_Click(ByVal sender As System.Object, ...
    FreqCounter.Increment()
    Counters(Number).Increment()
End Sub

```

Listing 547: Inkrementieren der PerformanceCounter im Programm

Die im Beispielprogramm erzeugten Daten können dann mit externen Programmen ausgewertet werden. Unter START / PROGRAMME / VERWALTUNG / SYSTEMMONITOR (Windows 2000) bzw. in der Verwaltungskonsole unter LEISTUNG (Windows XP) finden Sie das zu Windows gehörende Anzeige- und Auswerte-Programm. Auch die mit dem Beispielprogramm erzeugten PerformanceCounter können damit angezeigt werden (Abbildung 278 und Abbildung 279).

Eigene PerformanceCounter können Sie in vielen Fällen einsetzen, um z.B. Datenraten oder Zykluszeiten zu überwachen. Sie können einerseits ein hilfreiches Mittel beim Debuggen sein und andererseits dem Endanwender wertvolle Informationen über Ablauf und Auslastung der Prozesse und der Maschine geben.

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schie-  
denes

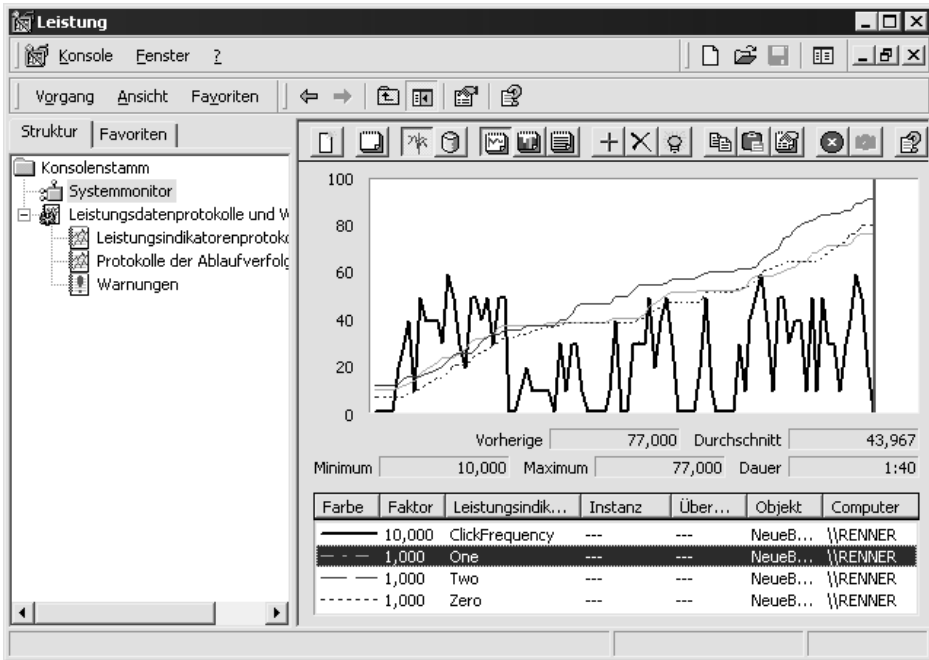


Abbildung 278: Kurvendiagramm der vier erstellten PerformanceCounter

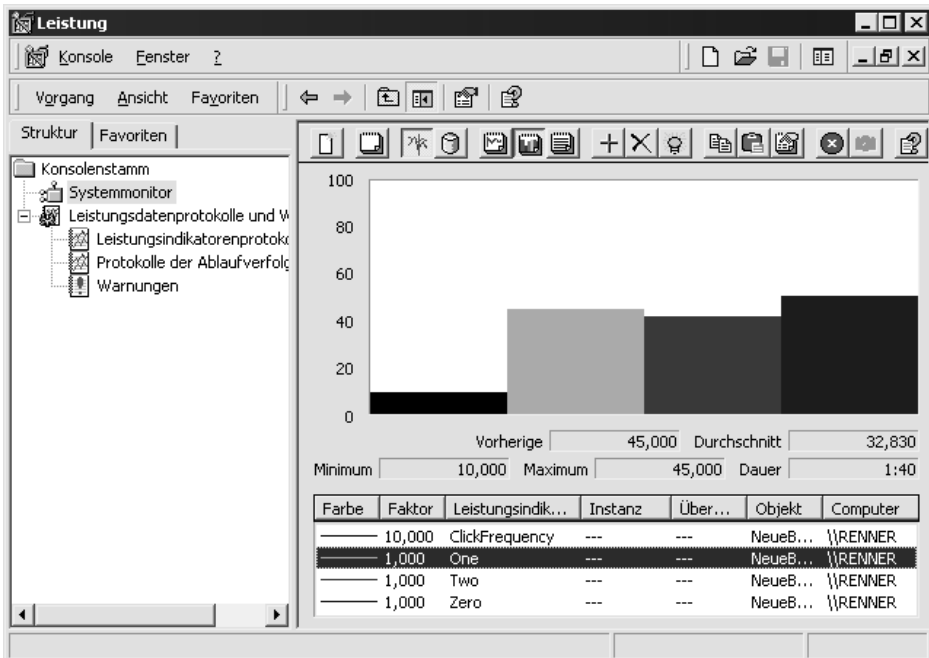


Abbildung 279: Die vier Counter in einer Histogramm-Darstellung

## 304 Zeiten mit hoher Auflösung messen

Zeitmessungen im Millisekunden-Bereich können nicht über `DateTime.Now` oder ähnliche Methoden erfolgen, da die Auflösung betriebssystembedingt bei ca. 10 ms liegt und je nach Systembelastung stark schwanken kann. Alternativ steht ein Hardware-Counter mit hoher Auflösung zur Verfügung, der über eine API-Funktion abgefragt werden kann.

Die API-Funktion `QueryPerformanceCounter` (Listing 5) ist dabei für die Abfrage des absoluten Zählerstandes zuständig, während mit der API-Funktion `QueryPerformanceFrequency` die Frequenz, mit der der Zähler hochgezählt wird, abgefragt werden kann. Für eine Zeitmessung benötigt man den Zählerstand zu Beginn und den Zählerstand am Ende des Zeitintervalls. Die Differenz, geteilt durch die Frequenz, gibt dann die Zeitspanne in Sekunden an.

```
Public Declare Auto Function QueryPerformanceCounter Lib _
    "Kernel32.dll" (ByRef performanceCount As Long) As Boolean
```

```
Public Declare Auto Function QueryPerformanceFrequency Lib _
    "Kernel32.dll" (ByRef frequency As Long) As Boolean
```

Listing 548: Abfragen von Zählerstand und Frequenz des Hardware-Counters

Zur Veranschaulichung zeigen wir Ihnen, wie die Intervallzeiten eines Windows-Timers gemessen werden können. Ein Testprogramm (Abbildung 280) verwendet ein Timer-Steuerelement, dessen `Interval`-Eigenschaft über ein Eingabefeld gesetzt werden kann. Bei Betätigen der START-Schaltfläche wird der Timer initialisiert und der Anfangswert des Counters festgehalten (Listing 6). Im Ereignis-Handler des Timers (Listing 7) wird der aktuelle Zählerstand abgefragt und im Array gespeichert. Nach der zehnten Messung werden der Timer gestoppt und die Intervallzeiten berechnet und in der ListBox dargestellt.

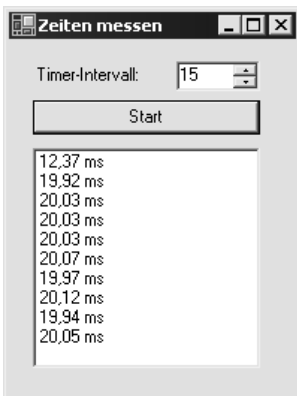


Abbildung 280: Messung der Intervallzeiten eines Windows-Timers

```
Private Sub BTNStart_Click(ByVal sender As ...

    Counter = 1
    LBValues.Items.Clear()
```

Listing 549: Start der Messung

Basics

Datum/  
Zeit

Anwen-  
dungen

Zeich-  
nen

Bild-  
bearbei-  
tung

Win-  
dows  
Forms

Controls

Proper-  
tyGrid

Datei-  
system

Netz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaft

Ver-  
schiede-  
nes

---

```

' Anfangszählerstand abfragen
API.QueryPerformanceCounter(Values(0))

' Intervall setzen und Timer starten
Timer1.Interval = CInt(NUDIntervall.Value)
Timer1.Enabled = True

```

End Sub

---

*Listing 549: Start der Messung (Forts.)*

---

```

Private Sub Timer1_Tick(ByVal sender As System.Object...

' Zählerstand speichern
API.QueryPerformanceCounter(Values(Counter))
Counter += 1

' Nach 10 Messungen abbrechen
If Counter > 10 Then

    Timer1.Enabled = False

' Frequenz abfragen
Dim freq As Long
API.QueryPerformanceFrequency(freq)

Dim diff As Long
For i As Integer = 1 To 10
    ' Zeitdifferenz berechnen und in ListBox ausgeben
    diff = Values(i) - Values(i - 1)
    Dim t As String
    t = String.Format("{0:00.00} ms", diff / freq * 1000)
    LBValues.Items.Add(t)
Next
End If

End Sub

```

---

*Listing 550: Zeiten messen und anzeigen*

---

Hinweis

Auch wenn die Abfragen mit `QueryPerformancCounter` eine wesentlich höhere Zeitauflösung bieten als z.B. `DateTime.Now`, sollten Sie bei Zeitmessungen einkalkulieren, dass eine Reihe von Störfaktoren die Messung beeinflussen kann. Windows ist kein Echtzeit-Betriebssystem. Ihr Prozess ist nicht der einzige, der auf der Maschine ausgeführt wird. Durch das Multitasking/Multithreading-Verfahren kann Ihr Prozess bzw. der zu messende Thread jederzeit unterbrochen werden. Sie können also nie sicher sein, dass Sie tatsächlich nur die Ausführungszeit Ihres Programms messen.



Wie Sie der Abbildung entnehmen können, hat die eingestellte Intervallzeit recht wenig mit den gemessenen Zeiten zu tun. Das Timer-Steuerelement arbeitet mit Zeitintervallen von ca. 10 Millisekunden. Bruchteile davon können nicht berücksichtigt werden.

## 305 API-Fehlermeldungen aufbereiten

Fehler in API-Funktionen führen meist nicht zu einer Exception, sondern entweder (wenn sie von der API-Funktion erkannt werden) zum Setzen einer globalen Fehlernummer oder (wenn sie eben nicht erkannt werden) zu unvorhersehbarem Verhalten des Programms.

Letztere Situation lässt sich leider nicht abfangen, da die Ursachen vielfältig sein können (falsche Zeiger, zu kleine Buffer-Arrays usw.). Wenn die API-Funktion jedoch einen Fehler erkennt, dann lässt sich dieser auch abfragen. Die Methode `System.Runtime.InteropServices.Marshal.GetLastWin32Error` gibt die besagte Fehlernummer zurück. Viele API-Funktionen geben als Funktionswert zurück, ob sie einen Fehler erkannt haben oder nicht. Liegt kein Fehler vor, gibt `GetLastWin32Error` den Wert 0 zurück, ansonsten die zugeordnete Fehlernummer.

Nun ist die Angabe der Fehlernummer meist nicht besonders hilfreich. Eine Meldung im Klartext ist in jedem Fall sinnvoller. Zur Abfrage des Meldungstextes stellt das Windows-API die Methode `FormatMessage` zur Verfügung (Listing 8). Da die Methode eine Reihe von Parametern benötigt, haben wir sie in der Klasse `ApiVbNet` in zwei anderen Methoden gekapselt (Listing 9).

```
Public Declare Auto Function FormatMessage Lib "kernel32.dll" _
    (ByVal dwFlags As Integer, ByVal lpSource As IntPtr, _
    ByVal dwMessageId As Integer, ByVal dwLanguageId As Integer, _
    ByVal lpBuffer As System.Text.StringBuilder, _
    ByVal nSize As Integer, ByVal Arguments() As String) As Integer
```

*Listing 551: Die API-Funktion `FormatMessage` gibt die Meldungstexte zu Fehlernummern zurück*

```
Public Shared Sub ThrowAPIError(ByVal errorNumber As Integer)

    ' Kein Fehler bei dem Wert 0
    If errorNumber = 0 Then Return

    ' Buffer für Text anlegen
    Dim sb As New System.Text.StringBuilder(2000)

    ' Meldungstext abrufen
    API.FormatMessage(&H1000, IntPtr.Zero, errorNumber, 0, sb, _
        sb.Capacity, Nothing)

    ' Exception auslösen
    Throw New Exception(sb.ToString())

End Sub

Public Shared Sub ThrowAPIError()
    ' Fehlernummer abfragen und bearbeiten
```

*Listing 552: Hilfsfunktionen zum Prüfen von API-Fehlern*

Basics

Datum/  
ZeitAnwen-  
dungenZeich-  
nenBild-  
bearbei-  
tungWin-  
dows  
Forms

Controls

Proper-  
tyGridDatei-  
systemNetz-  
werk

System

Daten-  
banken

XML

Wissen-  
schaftVer-  
schiede-  
nes

---

```
ThrowAPIError( _  
    System.Runtime.InteropServices.Marshal.GetLastWin32Error())  
End Sub
```

---

*Listing 552: Hilfsfunktionen zum Prüfen von API-Fehlern (Forts.)*

ThrowAPIError legt einen Buffer für die Textabfrage an und ruft FormatMessage auf. Der erhaltene Meldungstext wird zum Auslösen einer Exception verwendet. Sie können wahlweise ThrowAPIError ohne Parameter aufrufen oder eine bekannte Fehlernummer übergeben. Im ersten Fall wird die Fehlernummer mit GetLastError ermittelt.

Ein Aufruf wie

---

```
ApiVNet.ThrowAPIError(2)
```

---

erzeugt eine Exception wie z.B.:

---

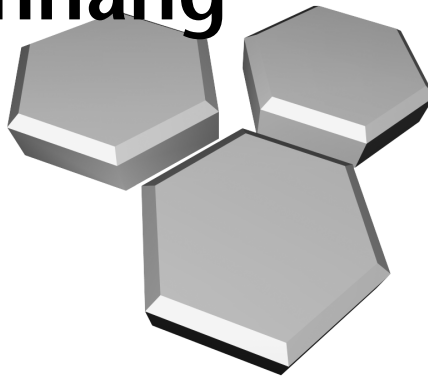
Eine nicht behandelte Ausnahme des Typs 'System.Exception' ist in TimeMeasurement.exe aufgetreten.

Zusätzliche Informationen: Das System kann die angegebene Datei nicht finden

---

# **TEIL III**

## **Anhang**





# Visual Basic .NET

Wenngleich wir an dieser Stelle auch keine vollständige Einführung in die Objektorientierte Programmierung mit Visual Basic .NET geben können, wollen wir doch zumindest die wichtigsten und im Buch oft benötigten Grundlagen ansprechen. Aber bitte beachten Sie, dass dies kein Ersatz für das Erlernen der Objektorientierten Programmierung ist, sondern nur als Kurzreferenz zur Klärung einiger weit verbreiteter Unklarheiten dienen soll.

## 1 Klassen – Referenzen – Objekte

Eine Klasse ist ein Bauplan für Objekte. Sie beschreibt, wie ein Objekt aussehen soll. Ein Objekt wiederum ist eine Instanz einer Klasse. Für jedes Objekt wird Speicherplatz auf dem Heap reserviert. Auf ein Objekt kann man nur über Referenzen zugreifen. Referenzen sind verwaltete Zeiger, die auf genau ein Objekt oder auf nichts (Nothing) zeigen. Ein Beispiel für eine Klasse sehen Sie in Listing 1.

---

```
Public Class Vehicle

    ' Member-Variablen
    Public Manufacturer As String
    Public VehicleType As String
    Public MaxSpeed As Double
    Public LicenceNumber As String

    ' Konstruktor
    Public Sub New(ByVal manufacturer As String, ByVal vtype As _
        String, ByVal maxspeed As Double, ByVal licenceNumber As String)

        Me.Manufacturer = manufacturer
        Me.VehicleType = vtype
        Me.MaxSpeed = maxspeed
        Me.LicenceNumber = licenceNumber

    End Sub

    ' Methode
    Public Function Move(ByVal speed As Double) As Boolean

        If speed > MaxSpeed Then Return False

        Debug.WriteLine(String.Format( _
            "Fahrzeug [{0}] fährt mit {1} km/h über die Autobahn", _
            Me.LicenceNumber, speed))

        Return True
    End Function
```

---

Listing 1: Klasse Vehicle definiert Member-Variablen, einen Konstruktor und eine Methode

---

End Class

---

*Listing 1: Klasse Vehicle definiert Member-Variablen, einen Konstruktor und eine Methode (Forts.)*

Um Instanzen einer Klasse anzulegen, wird das Schlüsselwort `New` verwendet. `New` führt drei Schritte aus:

1. Reservierung des Speicherplatzes für das Objekt auf dem Heap
2. Aufruf eines Konstruktors zur Initialisierung
3. Rückgabe der Referenz auf das neue Objekt

Eine Instanz der beschriebenen Klasse könnte also so angelegt werden:

---

```
Dim v As Vehicle = New Vehicle("Audi", "A8", 220, "QQ-C1234")
```

---

`v` ist eine Referenzvariable, die nach Ausführung der Zuweisung auf das Objekt zeigt. In Visual Basic .NET ist auch eine verkürzte Schreibweise erlaubt, die zum gleichen Ziel führt:

---

```
Dim v As New Vehicle("Audi", "A8", 220, "QQ-C1234")
```

---

Sie können beliebig viele Objekte erzeugen, aber auch beliebig viele Referenzen auf ein Objekt anlegen:

---

```
Dim myCar As Vehicle = v
```

---

`myCar` ist eine neue Referenzvariable, die ebenfalls auf das zuvor angelegte Objekt verweist. `v` und `myCar` haben somit denselben Wert. Das Beispiel zeigt sehr deutlich, dass eine gebräuchliche Ausdruckweise, bei der man umgangssprachlich ein Objekt so nennt, wie die Referenzvariable, die darauf verweist, irreführend und falsch ist, denn sonst hätte das Objekt jetzt schon zwei Namen. Objekte sind anonym und haben keinen Namen.

Der Konstruktor ist eine Methode, die keinen Rückgabewert hat und die nur im Zusammenhang mit der Instanzierung (hier mit `New`) aufgerufen wird. Er soll zur Initialisierung der zum Objekt gehörenden Werte dienen. Die Implementierung ist jedoch Sache des Programmiers. Im Beispiel werden die als Parameter übergebenen Werte in die Member-Variablen des Objektes eingetragen.

Über eine Referenz können Sie auf das Objekt zugreifen:

---

```
myCar.MaxSpeed = 200
```

---

setzt die Member-Variablen `MaxSpeed` des Objektes auf 200. Analog zu diesem schreibenden Zugriff können Sie auch lesend darauf zugreifen:

---

```
Dim speed As Double = myCar.MaxSpeed
```

---

Methoden eines Objektes rufen Sie ebenfalls über die Referenz auf:

---

```
myCar.Move(150)
```

---

führt zur Ausgabe von:

---

```
Fahrzeug [QQ-C1234] fährt mit 150 km/h über die Autobahn
```

---

oder mit Auswertung des Funktionswertes:

---

```
Debug.WriteLine(myCar.Move(300))
```

---

erzeugt die Ausgabe:

---

```
False
```

---

## 2 Strukturen (Wertetypen)

Strukturen, zu denen auch Datentypen wie `Integer`, `Double`, `Boolean`, `DateTime` usw. gehören, sind abgeleitet von der Klasse `System.ValueType` und unterliegen gesonderten Regeln:

- ▶ Eine Struktur kann nicht von einer anderen Struktur oder einer Klasse erben
- ▶ Eine Struktur kann nicht als Basisklasse für eine andere Klasse dienen
- ▶ Lokale Variablen und Methoden-Parameter, deren Typ eine Struktur ist, werden auf dem Stack abgelegt
- ▶ Wird eine Referenz auf eine Struktur benötigt, erfolgt automatisch »Boxing«

Der letzte Punkt ist ganz wesentlich. Bei folgendem Code:

---

```
Dim obj As Object = 123
```

---

wird auf dem Heap ein Objekt angelegt, das den Wert (123) verpackt (also quasi in eine Schachtel steckt, daher der Begriff »Boxing«). Der Wert des Objektes kann nicht geändert werden. Wenn Sie wieder einen Wertetyp benötigen, müssen Sie einen `Typecast` mit `CType` vornehmen:

---

```
Dim obj As Object = 123  
Dim i As Integer = CType(obj, Integer)
```

---

oder vereinfacht bei `Integer`:

---

```
Dim k As Integer = CInt(obj)
```

---

Dieser Vorgang wird auch »Unboxing« genannt.

»Boxing« erfolgt natürlich auch dann, wenn Sie einen Wertetyp an eine Methode übergeben, die eine `Object`-Referenz erwartet. Ein typischer Anwendungsfall ist z.B. `ArrayList.Add` (siehe Beschreibung der Listen weiter unten).

### 3 (Instanz-)Methoden

In einer Klasse können Sie beliebig viele Methode definieren, die zu einem Objekt gehören (Instanzmethoden). Die Parameterliste können Sie frei vorgeben, ähnlich, wie es in VB6 der Fall ist. Im Gegensatz zu VB6 ist die Voreinstellung für die Übergabeart allerdings jetzt ByVal statt vorher ByRef. Optionale Parameter sind zwar noch möglich, sind aber nicht .NET-konform und sollten daher vermieden werden.

#### Überladen

Neu ist, dass Methoden überladen werden können. D.h., dass Sie unter einem Methodennamen mehrere Implementierungen vornehmen können, die sich in den Typen der Parameter unterscheiden. Für das obige Beispiel ließe sich daher zusätzlich eine Methode wie in Listing 1.2 definieren.

---

```
Public Sub Move(ByVal destination As String, ByVal speed As Double)
    If speed > MaxSpeed Then Return

    Debug.WriteLine(String.Format( _
        "Fahrzeug [{0}] fährt mit {1} km/h nach {2}", _
        LicenceNumber, speed, destination))
End Sub
```

---

*Listing 2: Überladene Methode Move mit anderen Parametern*

Beim Aufruf erkennt der Compiler anhand der Parameter, welche Methode aufzurufen ist:

---

```
myCar.Move("Frankfurt", 140)
myCar.Move(150)
```

---

Hier wird erst die Methode mit zwei Parametern aufgerufen, danach die Methode mit einem Parameter. Überladungen werden immer zur Compile-Zeit aufgelöst.

Visual Studio zeigt Ihnen beim Aufruf einer überladenen Methode die möglichen Parameter in einer einzeiligen Liste an (Abbildung 1). Über die Schaltflächen oder die Pfeiltasten können Sie die Liste durchblättern.

```
Dim myCar As Vehicle = v
myCar.Move(|
nd Sub 2 von 2 Move (destination As String, speed As Double)
```

*Abbildung 1: Liste der verfügbaren Methodenüberladungen in Visual Studio*

Parameter können als Wert (ByVal) oder als Referenz (ByRef) übergeben werden. Die Art der Übergabe legt fest, was auf den Stack gelegt wird. Im ersten Fall wird der Wert eines Ausdrucks übergeben, im zweiten Fall die Adresse einer Variablen.

Achtung

Verwechseln Sie ByVal und ByRef auf keinen Fall mit Werte- und Referenztypen. Beides hat nichts miteinander zu tun. Sie können sowohl Werte- als auch Referenztypen als Parameter übergeben und in beiden Fällen festlegen, ob die Übergabe ByVal oder ByRef erfolgen soll.



Eine Übergabe von Variablen als Referenz wird relativ selten benötigt (daher die geänderte Voreinstellung). Ein Beispiel hierfür ist das Austauschen von Variablen-Inhalten:

---

```
Public Sub Exchange(ByRef v1 As Double, ByRef v2 As Double)
    Dim v As Double = v1
    v1 = v2
    v2 = v
End Sub
```

---

und der Aufruf:

---

```
Dim s1 As Double = 100
Dim s2 As Double = 150
Exchange(s1, s2)
```

---

führt dazu, dass anschließend s1 den Wert 150 und s2 den Wert 100 hat. Bei einer Übergabe `ByVal` hätte sich an den Variablen-Inhalten nichts geändert.

## Me

Bei einem Aufruf wie `myCar.Move(...)` wird die Referenz des Objektes, für das die Methode aufgerufen wird, als »unsichtbarer Parameter« übergeben und steht innerhalb der Methode mit dem Schlüsselwort `Me` zur Verfügung (siehe Beispiele). Über `Me` können Sie also innerhalb einer Instanz-Methode auf das Objekt zugreifen. `Me` kann weggelassen werden, wenn der Bezeichner dahinter eindeutig ist. Der Compiler setzt es dann bei der Kompilierung automatisch ein.

Der Typ von `Me` ist nicht zwangsläufig identisch mit der Klasse, in der die Methode definiert ist. `Me` kann ja auch auf eine Instanz einer abgeleiteten Klasse verweisen (siehe weiter unten).

## 4 Statische Methoden und statische Variablen

Eine statische Methode gehört zur Klasse und nicht zum Objekt. D.h., man kann die Methode aufrufen, ohne dass Instanzen einer Klasse vorhanden sein müssen. Demzufolge benötigen Sie auf keine Referenz, um die Methode aufrufen zu können. Stattdessen wird eine statische Methode über den Namen ihrer Klasse aufgerufen:

---

```
Vehicle.BuildCar(...)
```

---

und mit dem Schlüsselwort `Shared` implementiert:

---

```
Public Shared Function BuildCar() As Vehicle
    ' Return ...
End Function
```

---

Leider lässt Visual Basic .NET bislang auch den Aufruf einer statischen Methode über eine Referenz zu. An der Ausführung ändert sich dadurch jedoch nichts. Es führt auch nicht zu Vereinfachungen, sondern allenfalls zu Verwirrungen. Sie sollten daher statische Methoden immer ausschließlich über den Klassennamen aufrufen.

Genauso verhält es sich mit statischen Variablen. Sie gehören zur Klasse, nicht zum Objekt. Auf eine statische Variable können Sie jederzeit über den Klassennamen zugreifen.

## 5 Module

VB Classic-Module gehören zu den Dingen, die in .NET nichts mehr zu suchen haben. Leider werden sie von VB.NET weiterhin unterstützt und täuschen die Existenz globaler Variablen und Methoden vor.

In Wirklichkeit sind Module unter VB.NET aber nur Klassen, deren Methoden und Member-Variablen automatisch als statisch (*Shared*) deklariert werden. Um Verwirrungen zu vermeiden sollten Sie daher grundsätzlich statt Modulen Klassen verwenden.

Auch im Framework wird so vorgegangen. Eine typische Klasse, die nur statische Elemente enthält, ist die Klasse *Math*. Funktionen wie *Sin* oder *Round* benötigen keine Instanz der Klasse, sondern können wie globale Methoden über den Klassennamen aufgerufen werden:

---

```
x = Math.Sin(5)
```

---

Die in Listing 1.3 abgebildeten Funktionen hätte man in VB Classic sicher innerhalb eines Moduls definiert. Unter .NET ist es jedoch üblich, stattdessen statische Methoden (*Shared*) explizit als solche anzulegen.

---

```
Public Class TrigDeg

    ' Trigonometrische Funktionen mit Winkelangaben in Grad

    Public Shared Function DegreeToRadiant(ByVal angle As Double) _
        As Double

        Return angle * Math.PI / 180
    End Function

    Public Shared Function RadiantToDegree(ByVal radiant As Double) _
        As Double

        Return radiant * 180 / Math.PI
    End Function

    Public Shared Function Sin(ByVal angle As Double) As Double
        Return Math.Sin(DegreeToRadiant(angle))
    End Function

    Public Shared Function Cos(ByVal angle As Double) As Double
        Return Math.Cos(DegreeToRadiant(angle))
    End Function

    Public Shared Function Atan(ByVal value As Double) As Double
```

---

*Listing 3: Statt eines Moduls ist es besser, eine Klasse anzulegen und die enthaltenen Methoden mit »Shared« als statisch zu definieren*

---

```
Return RadiantToDegree(Math.Atan(value))  
End Function
```

```
End Class
```

---

*Listing 3: Statt eines Moduls ist es besser, eine Klasse anzulegen und die enthaltenen Methoden mit »Shared« als statisch zu definieren (Forts.)*

Beispiel für Aufrufe:

---

```
Debug.WriteLine(TrigDeg.Sin(45))  
Debug.WriteLine(TrigDeg.Atan(1))
```

---

## 6 Eigenschaften (Properties)

Die Eigenschaften kennen Sie schon aus VB Classic. Sie dienen einer syntaktischen Vereinfachung, indem zwei Methoden (Get und Set) zusammengefasst und im rufenden Programm wie eine Member-Variable behandelt werden. Definiert werden sie über das Schlüsselwort Property:

---

```
Public Property SpeedInKMH() As Double  
Get  
Return kmh  
End Get  
Set(ByVal Value As Double)  
kmh = Value  
End Set  
End Property
```

---

Meist greifen sie, wie in diesem Beispiel, auf private oder geschützte Member-Variablen zu:

---

```
Private kmh As Double
```

---

Die Implementierung kann aber beliebig vorgenommen werden. Z.B. für Umrechnungen könnte eine zweite Eigenschaft definiert werden:

---

```
Public Property SpeedInMPH() As Double  
Get  
Return kmh / 1.6  
End Get  
Set(ByVal Value As Double)  
kmh = Value * 1.6  
End Set  
End Property
```

---

Auch diese Eigenschaft greift auf die Member-Variable `kmh` zu, rechnet die Werte jedoch in Meilen pro Stunde um. Aufgerufen werden die Eigenschaften z.B. so:

---

```
myCar.SpeedInKMh = 150
Debug.WriteLine(myCar.SpeedInMPH & " miles/h")
myCar.SpeedInMPH = 100
Debug.WriteLine(myCar.SpeedInKMh & " km/h")
```

---

wodurch die Ausgaben

---

```
93,75 miles/h
160 km/h
```

---

erzeugt werden.

Auch Properties können statisch definiert werden und gehören dann, wie statische Methoden, zur Klasse und nicht zu einer Instanz.

## 7 Vererbung

Eine Klasse kann von einer bestehenden abgeleitet werden und erbt dann, bis auf die Konstrukto-  
ren, alles, was in der Basisklasse definiert ist. Listing 1.4 zeigt ein Beispiel für eine abgeleitete  
Klasse.

---

```
Public Class VWGolf
    Inherits Vehicle

    Public NumberOfVersion As Integer

    Public Sub New(ByVal licenceNumber As String, _
        ByVal versionNr As Integer)

        MyBase.New("VW", "Golf", 170, licenceNumber)
        NumberOfVersion = versionNr

    End Sub
End Class
```

---

*Listing 4: Die von Vehicle abgeleitete Klasse VWGolf*

Eine Instanz von VWGolf kennt dieselben Methoden wie eine Instanz von Vehicle. Somit funk-  
tioniert folgender Code:

---

```
Dim golf As New VWGolf("ZZ-X999", 2)
golf.Move(120)
```

---

Da eine Instanz von VWGolf ja auch ein Vehicle ist, ist es auch zulässig, einer Referenz-Variablen  
vom Typ Vehicle die Referenz eines VWGolf-Objektes zuzuweisen

---

```
Dim v As Vehicle = golf
```

---

und so allgemeiner mit der spezialisierten Klasse umzugehen. Eine Klasse, für die keine Basis-klassse definiert wird, erbt automatisch von der Klasse `System.Object`. `Object` steht daher immer an der Spitze einer Klassenhierarchie. Also gilt auch diese Zuweisung:

---

```
Dim o As Object = golf
```

---

Viele Framework-Methoden machen sich diese Möglichkeit zunutze, indem sie Algorithmen ganz allgemein mit `Object`-Referenzen implementieren und so universell einsetzbar sind.

In der anderen Richtung, wenn Sie also eine Referenz vom Typ `Object` haben, die auf ein Objekt einer spezielleren Klasse verweist, und Sie z.B. auf eine Member-Variable dieser Klasse zugreifen wollen, dann müssen Sie etwas mehr Aufwand treiben. Ein Aufruf wie `o.Move(...)` kann nicht funktionieren, da der Compiler nicht weiß, auf welchen Objekttyp `o` verweist. Das müssen Sie ihm mit Hilfe eines `TypeCasts` bekannt geben. Zwei Möglichkeiten gibt es in Visual Basic .NET:

---

```
Dim v1 As Vehicle = CType(o, Vehicle)
Dim v2 As Vehicle = DirectCast(o, Vehicle)
```

---

`DirectCast` ist schneller, aber nur in Zusammenhang mit Klassen (wie hier) erlaubt.

## Methoden überschreiben

Methoden, die in der Klasse, in der sie zuerst definiert worden sind, als überschreibbar (`Overridable`) markiert worden sind, können in abgeleiteten Klassen überschrieben werden. Dabei setzt die Methodenimplementierung der abgeleiteten Klasse die der Basisklasse außer Kraft. Im Sprachgebrauch der Objektorientierten Programmierung werden überschreibbare Methoden auch »virtuelle« Methoden genannt. Eine Definition der Methode `Move` wird in der Basisklasse `Vehicle` wie folgt als virtuelle Methode definiert:

---

```
Public Class Vehicle
    ...

    Public Overridable Sub Move(ByVal start As String, _
        ByVal destination As String)

        Debug.WriteLine(String.Format( _
            "Fahrzeug [{0}] fährt von {1} nach {1}", Me.LicenceNumber, _
            start, destination))
    End Sub

End Class
```

---

In der abgeleiteten Klasse `VWGoIf` kann sie dann mit neuem Inhalt definiert werden:

---

```
Public Class VWGoIf
    Inherits Vehicle

    ...

    Public Overloads Overrides Sub Move(ByVal start As String, _
        ByVal destination As String)
```

---

```

        Debug.WriteLine(String.Format( _
            "Ein Golf {0} fährt von {1} nach {2}", _
            Me.NumberOfVersion, start, destination))
    End Sub
End Class

```

Wird die Methode `Move` nun für eine Instanz von `Vehicle` und eine Instanz von `VWGolf` aufgerufen, offenbart sich der Unterschied:

```

Dim v1 As New Vehicle("Audi", "A8", 220, "QQ-C1234")
Dim v2 As Vehicle = New VWGolf("ZZ-X999", 2)
v1.Move("A", "B")
v2.Move("A", "B")

```

führt zur Ausgabe von:

```

Fahrzeug [QQ-C1234] fährt von A nach A
Ein Golf 2 fährt von A nach B

```

Obwohl beide Referenz-Variablen vom Typ `Vehicle` sind, wird im zweiten Fall die Methode `Move` der abgeleiteten Klasse `VWGolf` aufgerufen. Erst zur Laufzeit wird anhand des Objekttyps entschieden, welche Methode aufgerufen werden muss.

Überschreibungen werden also immer zur Laufzeit aufgelöst, Überladungen immer zur Compile-Zeit.

Die Entwicklungsumgebung nimmt Ihnen die Arbeit ab, für eine Überschreibung die Methodendeklaration einzugeben. Wenn Sie im Code-Fenster der abgeleiteten Klasse in der linken oberen ComboBox »(Überschreibungen)« auswählen, dann können Sie in der rechten ComboBox eine Methode auswählen, für die automatisch der Code generiert wird (Abbildung 2).



Abbildung 2: Automatische Code-Generierung zum Überschreiben von Methoden

## Zugriffsmodifizierer

Über fünf verschiedene Zugriffsmodifizierer lässt sich steuern, wie auf Klassen-Member zugegriffen werden kann. Tabelle 1 erklärt deren Bedeutung.

| Zugriffsmodifizierer | Bedeutung                                                         |
|----------------------|-------------------------------------------------------------------|
| Public               | Öffentlich. Alle Zugriffe erlaubt                                 |
| Protected            | Geschützt. Zugriff nur innerhalb der Vererbungshierarchie möglich |
| Private              | Privat: Zugriff nur innerhalb derselben Klasse möglich            |

Tabelle 1: Zugriffsmodifizierer in Visual Basic .NET

| Zugriffsmodifizierer | Bedeutung                                                       |
|----------------------|-----------------------------------------------------------------|
| Friend               | Projekt: Zugriff nur innerhalb desselben Projektes erlaubt      |
| Protected Friend     | Vereinigungsmenge (nicht Schnittmenge) von Protected und Friend |

Tabelle 1: Zugriffsmodifizierer in Visual Basic .NET (Forts.)

## Abstrakte Klassen

Klassen können als »abstrakt« (Schlüsselwort `MustInherit`) gekennzeichnet werden, um zu verhindern, dass sie instanziiert werden können. Sie können dann nur für Ableitungen verwendet werden (wie das VB-Schlüsselwort schon andeutet).

Im Gegensatz zu C++ muss eine abstrakte Klasse nicht zwangsläufig abstrakte Methoden enthalten.

## Abstrakte Methoden

Methoden, die in einer Basisklasse nicht implementiert werden sollen, können als abstrakte Methoden deklariert werden (Schlüsselwort `MustOverride`). Das hat folgende Auswirkungen:

- ▶ Da die Methode über eine Referenz des Typs der Klasse, in der sie abstrakt definiert wurde, aufrufbar ist, darf es keine Instanzen dieser Klasse geben. Sonst würde zur Laufzeit eine nicht existierende Methode aufgerufen. Als Konsequenz ergibt sich daraus, dass die Klasse selbst auch abstrakt ist.
- ▶ Eine Klasse, die von einer abstrakten Klasse abgeleitet wird, muss entweder alle abstrakten Methoden der Basisklasse implementieren oder sie ist selbst abstrakt

Listing 1.5 zeigt ein Beispiel für eine abstrakte Basisklasse und eine nicht abstrakte Ableitung.

---

```
Public MustInherit Class Animal

    Public MustOverride Sub Move()

End Class

Public Class Fisch
    Inherits Animal

    Public Overrides Sub Move()
        Debug.WriteLine("Der Fisch schwimmt...")
    End Sub

End Class
```

---

Listing 5: Definition einer abstrakten Methode in einer abstrakten Klasse und Implementierung der Methode in einer abgeleiteten Klasse

Eine Deklaration wie

---

```
Dim a As New Animal
```

---

ist dann nicht mehr möglich, da `Animal` abstrakt ist und keine Instanzen zulässt. Erlaubt ist:

---

```
Dim f As New Fish
f.Move()
Dim a As Animal = f
a.Move()
```

---

## Schnittstellen (Interfaces)

Eine Schnittstelle ist eine abstrakte Klasse, die nur aus abstrakten Methoden, abstrakten Eigenschaften, Typdefinitionen oder Ereignissen besteht. Es dürfen keine Implementierungen vorgenommen werden. Member-Variablen sind nicht erlaubt.

Der Name einer Schnittstelle beginnt meist mit einem großen »I«. Z.B.:

---

```
Public Interface ICanSwim
    Sub Swim()
End Interface
```

---

Implementiert wird eine Schnittstelle von einer Klasse durch Verwendung des Schlüsselworts `Implements`:

---

```
Public Class Fish
    Inherits Animal
    Implements ICanSwim

    ...
    Public Sub Swim() Implements ICanSwim.Swim
        Debug.WriteLine("Der Fisch schwimmt im Teich")
    End Sub

End Class
```

---

Bei einer Implementierung dieser Art ist der direkte Aufruf der Methode `Swim` über eine Referenz vom Typ `Fish` möglich:

---

```
f.Swim()
```

---

Es lässt sich aber auch, ohne Kenntnis der Klassen `Animal` oder `Fish`, die Methode `Swim` für ein `Fish`-Objekt aufrufen:

---

```
Dim swimmableObject As ICanSwim = f
swimmableObject.Swim()
```

---

Über eine Referenz vom Typ der Schnittstelle lassen sich alle Schnittstellenmethoden aufrufen. Der Aufrufer benötigt dazu keine Informationen über das Objekt. Diese Konstruktion wird oft im Framework verwendet (siehe z.B. die Schnittstellen `Comparable` und `Comparer`).



Eine Klasse kann beliebig viele Schnittstellen implementieren. Für jede implementierte Schnittstelle müssen alle definierten Methoden und Eigenschaften implementiert werden. Anderenfalls muss die Klasse als abstrakt gekennzeichnet werden.

## 8 Arrays

Häufig benötigt man Konstruktionen, um gleichartige Daten zusammenhängend zu speichern und zu bearbeiten. Fast alle Programmiersprachen und VB.NET natürlich auch bieten hierfür die Möglichkeit, Arrays zu definieren. Ein Array ist ein zusammenhängender Speicherbereich, der aus einer vorgegebenen Anzahl von Elementen gleichen Typs besteht. Der Zugriff auf einzelne Elemente erfolgt über Indizes. Indizes sind ganze Zahlen, die größer oder gleich Null sind.

Während Sprachen wie C++ die Möglichkeit bieten, Stack-basierte Arrays anzulegen, werden unter .NET Arrays grundsätzlich dynamisch auf dem Heap angelegt. Das heißt: Arrays sind Objekte! Array-Variablen sind also immer Referenzvariablen und verhalten sich teilweise etwas anders, als Sie es von den VB6-Arrays gewohnt sind. Daher wollen wir den Umgang mit Arrays etwas genauer beschreiben.

### Eindimensionale Arrays definieren

Nach wie vor funktioniert eine Arraydefinition, wie Sie sie aus VB6 kennen:

---

```
Dim Numbers(10) As Integer
```

---

Diese Anweisung legt ein Array mit 11 Integer-Zellen an. `Feld` ist ein Verweis auf das Array und nicht das Array selbst. Der Zugriff erfolgt über einen Index. So weisen Sie dem Element mit dem Index 5 einen Wert zu bzw. geben es in das Debug-Fenster aus:

---

```
Numbers (5) = 123  
Debug.WriteLine(Numbers (5))
```

---

Die Indizierung eines Arrays lässt sich nicht mehr steuern wie in VB6. Sie beginnt immer bei Null! Das Array hat also 11 Elemente mit den Indizes 0 bis 10. Ein Array hat somit immer ein Element mehr, als Sie bei der Definition angegeben haben.

Die aus den alten VB-Dialekten übernommene Syntax für die Array-Definition verbirgt leider, was alles im Hintergrund passiert. Betrachten Sie einmal die analoge C#-Syntax:

---

```
Int []Numbers = New int[11];
```

---

Links steht der Typ der zu definierenden Variablen: ein Array von Integer-Werten. `New int[11]` legt auf dem Heap ein neues Objekt an, das die 11 Integer-Werte aufnehmen kann. Als Ergebnis liefert `New` die Referenz dieses Objektes.

Versucht man, diese Schreibweise in die Syntax von VB umzusetzen, stößt man auf ein Problem:

---

```
Dim Numbers As Integer() = New Integer(10)
```

---

Der Compiler interpretiert die 10 in den runden Klammern bei `Integer(10)` als Parameter für einen Konstruktor und erzeugt eine Fehlermeldung. Hier zeigt sich der Nachteil von VB, auch für

die Array-Indizierung die runden Klammern einzusetzen. Durch einen Kniff lässt sich dennoch die Syntax verwenden:

---

```
Dim Numbers As Integer() = New Integer(10) {}
```

---

Die geschweiften Klammern deuten dem Compiler an, dass hier Daten für die Initialisierung des Arrays übergeben werden sollen. In diesem Fall interpretiert der Compiler die 10 in der Klammer als Größenangabe für das Array.

In dieser Definition wird deutlich, dass die Array-Definition Speicherplatz auf dem Heap anlegt. Sie ist identisch mit der ersten Definition, die sie aus VB6 kennen. Letztlich bleibt es Ihnen überlassen, welche Variante Sie vorziehen. Folgende Varianten haben die gleiche Wirkung:

---

```
Dim Numbers(10) As Integer
Dim Numbers As Integer() = New Integer(10) {}
Dim Numbers() As Integer = New Integer(10) {}
```

---

Der Datentyp der Array-Elemente kann beliebig gewählt werden. D.h., Sie können sowohl Wertetypen als auch Referenztypen einsetzen. Existiert beispielsweise eine Klasse `Vehicle`, dann legt folgende Definition ein Array von `Vehicle`-Referenzen an:

---

```
Dim Vehicles(5) As Vehicle
```

---

Hier wird ein Array mit sechs Elementen angelegt, die auf ein `Vehicle`-Objekt verweisen können.

Hinweis

Wenn Sie ein Array eines Referenztyps definieren, dann sind alle Array-Elemente Referenzen. Sie werden mit `Nothing` vorbelegt. Instanzen des Referenztyps werden dabei nicht angelegt! Das müssen Sie zusätzlich vornehmen.

Arrays enthalten niemals Objekte, sondern nur Verweise auf Objekte! Vermeiden Sie daher möglichst Formulierungen wie »Fünf Fahrzeuge in einem Array speichern«.

## Arrays mit Werten vorbelegen

Wie schon angedeutet, können in VB.NET einem Array bereits bei der Definition Werte zugewiesen werden. Eine Zuweisung von Konstanten sieht beispielsweise so aus:

---

```
Dim Numbers As Integer() = New Integer() {1, 4, 6, 83}
```

---

Hier wird ein eindimensionales Array mit vier Elementen angelegt, denen die in den geschweiften Klammern angegebenen Werte zugewiesen werden. Die einzelnen Werte werden mit Kommas voneinander getrennt.

Wenn Sie das Array mit Werten vorbelegen, können Sie auf die Angabe der Array-Größe verzichten. Sie ergibt sich ja aus der Anzahl der Werte. Gegen Sie sie jedoch an, dann müssen Sie genauso viele Werte definieren, wie das Array Elemente hat.

Sie sind hier nicht auf Konstanten beschränkt. Vielmehr können Sie beliebige Werte, auch Referenzen, vorgeben. Das folgende Beispiel zeigt eine Vorbelegung für ein Array von Fahrzeugen mit neuen Fahrzeug-Instanzen:

---

```
Dim Vehicles As Vehicle() = New Vehicle() _
{
    New Vehicle("VW", "Golf"),           _
    New Vehicle("Opel", "Astra"),        _
    New Vehicle("Mercedes", "500S")      _
}
```

---

Vorausgesetzt, es gibt eine Klasse `Vehicle`, die etwa folgendermaßen definiert ist, werden drei Instanzen von `Vehicle` angelegt und deren Referenzen in ein neues Array mit drei `Vehicle`-Referenzen eingetragen. Die Variable `Vehicles` verweist anschließend auf dieses Array.

---

```
Class Vehicle
    Public Sub New(ByVal company As String, ByVal vtype As String)
        ...
    End Sub
End Class
```

---

So lassen sich Arrays zur Laufzeit mit beliebigen Daten initialisieren. Die beschriebene Form ist allerdings nur in Verbindung mit der Array-Definition zulässig. Ist das Array bereits angelegt, können Sie die vereinfachte Schreibweise mit den geschweiften Klammern nicht mehr verwenden, sondern müssen über die Indizierung jedem Element einzeln die Werte zuweisen.

## Arrays und der Zuweisungsoperator

Zuweisungen verhalten sich etwas anders, als Sie es vielleicht von VB6 gewohnt sind:

---

```
Dim NumbersA As Integer() = New Integer() {1, 4, 6, 83}
Dim NumbersB As Integer() = NumbersA
```

---

Sie müssen sich hier darüber bewusst sein, dass wie bei anderen Objekten, die Referenzen und nicht die Daten kopiert werden. `NumbersB` verweist daher nach der Zuweisung auf dasselbe Objekt wie `NumbersA`. Das Array existiert nur einmal.

---

```
NumbersB(0) = 999
Debug.WriteLine(NumbersA(0))
Debug.WriteLine(NumbersB(0))
```

---

Dieser Code gibt zweimal 999 aus, dass beide Variablen auf dasselbe Array verweisen.

Vermeiden Sie daher möglichst, von einem Array namens *NumbersA* zu sprechen, nur weil Sie die Variable so genannt haben. Objekte haben keinen Namen und es können beliebig viele Referenzvariablen auf ein Objekt verweisen.

## Arrays als Funktionsparameter übergeben

Sie können Funktionen programmieren, die Arrays als Parameter entgegennehmen. Wie bereits erwähnt, können Sie nur eine Referenz eines Arrays übergeben, aber niemals dessen Werte. Das Array-Objekt seinerseits kennt dafür einige nützliche Eigenschaften, die es Ihnen erleichtern, Methoden zu programmieren, die mit beliebigen Array-Größen umgehen können. Betrachten Sie die Funktion in Listing 1.6.

---

```
Public Function GetSumOfArray(ByVal numbers As Integer()) _
    As Integer

    'Schleifenvariable
    Dim i As Integer
    'Summenvariable
    Dim sum As Integer = 0

    'Feld durchlaufen und Elemente aufaddieren
    For i = 0 To numbers.GetUpperBound(0)
        sum += numbers(i)
    Next

    'Ermittelte Summe zurückgeben
    Return sum

End Function
```

---

*Listing 6: Ermittlung der Summe aller Elemente in einem Array beliebiger Größe*

Die Funktion nimmt ein beliebiges Integer-Array als Parameter entgegen und berechnet die Summe der Elemente. Mit der Funktion `GetUpperBound` wird der höchstzulässige Index ermittelt. So kann die Funktion ein unbekanntes Array vollständig durchlaufen, ohne zusätzlich über dessen Größe informiert zu werden, wie es beispielsweise in C++ erfolgen müsste.

Der Parameter, der bei `GetUpperBound` übergeben werden muss, ist die Nummer der Dimension, für die Sie den höchsten Index ermitteln wollen. Mehrdimensionale Arrays werden weiter unten beschrieben.

Mit Hilfe der Eigenschaft `Length` lässt sich die Gesamtgröße eines Arrays bestimmen. Im vorliegenden Fall des eindimensionalen Arrays ist das `GetUpperBound(0) + 1`. Bei mehrdimensionalen Arrays gibt `Length` die Anzahl aller Elemente an.

In der Parameterliste sehen Sie, dass `numbers` als Typ `Array` von `Integer` übergeben wird. `ByVal` bedeutet in diesem Fall, dass eine Kopie der Referenz übergeben wird, nicht eine Kopie der Daten!

Änderungen, die Sie innerhalb der Funktion am Array vornehmen, wirken sich daher auch auf die rufende Methode aus. Wollen Sie das nicht, müssen Sie selbst eine Kopie des Arrays anlegen.

## Arraydimensionen nachträglich ändern

Gelegentlich kommt es vor, dass die Größe eines Arrays nachträglich verändert werden muss. Solche Situationen treten beispielsweise auf, wenn zum Zeitpunkt der Array-Definition noch nicht festgestellt werden kann, wie viele Elemente gespeichert werden müssen.

Die Frage ist nur, ob man ein Array nachträglich vergrößern kann. Da ein Array als geschlossener Speicherbereich auf dem Heap angelegt wird, muss die Antwort lauten:

Nein!

Der auf dem Heap reservierte Speicherbereich kann nicht vergrößert oder verkleinert werden. Arrays sind, einmal angelegt, unveränderlich.

Der einzige Ausweg besteht darin, ein neues Array mit der gewünschten Größe anzulegen und die Daten in dieses neue Array zu kopieren. Dazu bietet die Klasse `System.Array` die Methode `Copy`.

---

```
'ursprüngliches Array
Dim Numbers As Integer() = New Integer() {1, 4, 6, 83}
'neues Array mit 6 Elementen
Dim NumbersNew(5) As Integer
'Kopieren der alten Werte
Array.Copy(Numbers, NumbersNew, Numbers.Length)
'Weiter mit dem neuen Array arbeiten
Numbers = NumbersNew
```

---

Nun werden Sie sich vielleicht daran erinnern, dass Sie dieses Problem in VB6 doch mit `ReDim` lösen konnten. Auch das können Sie in VB.NET weiter verwenden:

---

```
ReDim Preserve Numbers(5)
```

---

Diese Anweisung führt zum gleichen Ergebnis wie der vorherige Code. Sicher ist die Anweisung syntaktisch kürzer, aber machen Sie sich bewusst, dass hinter `ReDim Preserve` der gleiche Algorithmus steckt. Es muss ein neues Array angelegt und die Werte kopiert werden. Das kostet Prozessorzeit und wird bei leichtfertigem Einsatz von `ReDim Preserve` leicht übersehen.

In den meisten Fällen, in denen Sie nachträglich ein Array vergrößern müssen, ist das Array als Datenstruktur die falsche Wahl. Hier können die Listen ihre Vorzüge ausspielen. Mehr dazu lesen Sie weiter unten.

## Mehrdimensionale Arrays

Es gibt zwei Möglichkeiten, Arrays mit mehreren Dimensionen anzulegen:

1. Wirkliche mehrdimensionale Arrays (rechteckige Arrays)

Das sind Arrays mit beispielsweise  $5 \cdot 7 \cdot 10$  Dimensionen. Die Anzahl der Elemente eines solchen Arrays ergibt sich aus dem Produkt der einzelnen Größen der Dimensionen. Sie werden rechteckig genannt, weil man sie sich als eine rechteckige geometrische Figur (in diesem Fall einen Quader) vorstellen kann.

2. Jagged (unregelmäßige, ungleichförmige oder verzweigte) Arrays

Hierbei handelt es sich nicht wirklich um mehrdimensionale Arrays. Vielmehr definiert man eindimensionale Arrays, die als Elemente Verweise auf weitere eindimensionale Arrays haben.

Mehrdimensionale Arrays können Sie definieren, wie Sie es aus VB6 gewohnt sind. Alternativ können Sie die an C# angelehnte Schreibweise verwenden:

---

```
Dim Numbers(2, 3) As Integer
Dim Numbers As Integer(,) = New Integer(2, 3) {}
```

---

Beide Anweisungen haben die gleiche Wirkung. Es wird ein Array mit  $3 \cdot 4$  Elementen angelegt. Auch hier können Sie wieder Vorbelegungen vornehmen:

---

```
Dim Numbers(,) As Integer = {{10, 11}, {20, 21}, {30, 31}}
```

---

Diese Anweisung legt ein Array mit der Dimension  $3 \cdot 2$  an. Innerhalb der äußeren geschweiften Klammern befinden sich die Werte für die erste Dimension (der linke Index). Sie sehen drei Werte, nämlich drei Arrays, bestehend aus jeweils zwei Zahlen. Für die zweite Dimension sind dann die Werte innerhalb der inneren Klammern vorgesehen. Insgesamt ergibt sich folgende Zuordnung:

---

```
Numbers(0, 0) = 10
Numbers(0, 1) = 11
Numbers(1, 0) = 20
Numbers(1, 1) = 21
Numbers(2, 0) = 30
Numbers(2, 1) = 31
```

---

Sie können beliebig viele Dimensionen definieren. Für die Syntax gilt, dass Sie für  $n$  Dimensionen  $n-1$  Kommas vorsehen müssen. Möchten Sie größeren Arrays bei der Definition Werte mitgeben, empfiehlt es sich, die Definition auf mehrere Zeilen umzubreaken. Hier ein Beispiel mit drei Dimensionen:

---

```
Dim Numbers3D(,,) As Integer = {{ _
                                {1000, 1001}, _
                                {1010, 1011}, _
                                {1020, 1021}}, _
                                { _
                                {1100, 1101}, _
                                {1110, 1111}, _
                                {1120, 1121}}, _
                                { _
                                {1200, 1201}, _
                                {1210, 1211}, _
                                {1220, 1221}}}
```

---

Hier erkennen Sie drei Blöcke, bestehend jeweils aus drei Zweierblöcken. Die Dimensionierung ist somit  $3 \cdot 3 \cdot 2$ . Selbstverständlich können Sie für jede Dimension die Anzahl der Elemente ermitteln:

---

```
Debug.WriteLine("1. Dimension: " & Numbers3D.GetUpperBound(0) + 1)
Debug.WriteLine("2. Dimension: " & Numbers3D.GetUpperBound(1) + 1)
Debug.WriteLine("3. Dimension: " & Numbers3D.GetUpperBound(2) + 1)
```

---

Denken Sie daran, dass `GetUpperBound` nicht die Anzahl der Elemente, sondern den höchstzulässigen Index zurückgibt. Und der ist immer um eins kleiner als die Anzahl. Die Ausgabe aus vorstehendem Beispiel ergibt:

---

```
1. Dimension: 3
```

---

---

```
2. Dimension: 3
3. Dimension: 2
```

---

Alternativ können Sie auch die Member-Funktion `GetLength` verwenden. Diese gibt Ihnen direkt die Größe einer Dimension an. Die Eigenschaft `Length` hingegen gibt die Anzahl aller Elemente des Arrays an:

---

```
Debug.WriteLine("1. Dimension: " & Numbers3D.GetLength(0))
Debug.WriteLine("2. Dimension: " & Numbers3D.GetLength(1))
Debug.WriteLine("3. Dimension: " & Numbers3D.GetLength(2))
Debug.WriteLine("Gesamtlänge: " & Numbers3D.Length)
```

---

Ausgabe:

---

```
1. Dimension: 3
2. Dimension: 3
3. Dimension: 2
Gesamtlänge: 18
```

---

Möchten Sie eine Funktion schreiben, die ein mehrdimensionales Array als Parameter annimmt, müssen Sie bei der Parameter-Definition die Anzahl der Dimensionen, nicht jedoch deren Größe festlegen. Listing 1.7 zeigt ein Beispiel einer Funktion, die ermittelt, wie viele Zahlen eines Arrays durch 3 teilbar sind.

---

```
'Als Parameter muss ein dreidimensionales Array übergeben werden
'Das wird durch (,,) kenntlich gemacht
```

```
Public Function CountDiv3(ByVal numbers(,,) As Integer) As Integer
```

```
    Dim i, j, k As Integer
    Dim counter As Integer = 0

    'Verschachtelte Schleifen für jede Dimension
    For i = 0 To numbers.GetUpperBound(0)
        For j = 0 To numbers.GetUpperBound(1)
            For k = 0 To numbers.GetUpperBound(2)
                'Testen, ob Zahl durch drei teilbar ist
                If (numbers(i, j, k) Mod 3) = 0 Then counter += 1
            Next
        Next
    Next
    Return counter
End Function
```

---

*Listing 7: Funktion zur Bearbeitung eines dreidimensionalen Arrays, dessen Dimensionen zur Compilezeit nicht bekannt sind*

Ausgabe:

---

Anzahl durch drei teilbare Zahlen: 6

---

Bei mehrdimensionalen Arrays ist die Anzahl der Elemente eines Unterelementes innerhalb einer Dimension immer gleich. Im obigen Beispiel hat jedes Element der ersten Dimension genau drei Elemente. Jedes Element der zweiten Dimension hat genau zwei Elemente.

Soll die Anzahl variabel gestaltet werden, greift man auf die o.g. *Jagged-Arrays* zurück. Hierzu wird zunächst ein eindimensionales Array definiert. Dieses Array enthält selbst keine Werte, sondern nur Referenzen auf Unter-Arrays. Im nächsten Schritt werden die Unter-Arrays angelegt und die Referenzen im Haupt-Array eingetragen. Die Unterarrays können beliebige Größen haben, müssen allerdings vom gleichen Typ sein. In Listing 1.8 sehen Sie ein Beispiel für ein Jagged-Array mit Integer-Zahlen.

---

```
'Array mit vier Referenzen auf Integer-Arrays anlegen
Dim Jagged(3)() As Integer
'nun die Unterarrays anlegen und die Referenzen setzen
Jagged(0) = New Integer(8) {}
Jagged(1) = New Integer() {1, 2, 3, 4}
Jagged(2) = New Integer() {123}
Jagged(3) = New Integer() {10, 20, 30, 40, 50}

'Array durchlaufen und ausgeben
Dim i, j As Integer
'Äußeres Array durchlaufen
For i = 0 To Jagged.GetUpperBound(0)
    'Inneres Array durchlaufen
    For j = 0 To Jagged(i).GetUpperBound(0)
        Debug.Write(Jagged(i)(j) & " ")
    Next
    Debug.WriteLine("")
Next
```

---

Listing 8: Anlegen und Durchlaufen eines Jagged-Arrays

An der Ausgabe des Beispielcodes erkennt man den Ursprung der Bezeichnung *Jagged* (*Unregelmäßig*):

---

```
0 0 0 0 0 0 0 0 0
1 2 3 4
123
10 20 30 40 50
```

---

Beachten Sie in der inneren Schleife, dass die obere Indexgrenze hier für jedes Element des äußeren Arrays neu ermittelt werden muss. Das geschieht mit

---

```
Jagged(i).GetUpperBound(0)
```

---



Sie ermitteln hier also die Indexgrenze des Elementes `Jagged(i)` und nicht die Indexgrenze der zweiten Dimension! Das ist der Unterschied zu mehrdimensionalen Arrays.

Wichtig in diesem Zusammenhang ist auch, dass Sie sich klarmachen, dass der Code aus Listing 1.8 nicht ein Array, sondern insgesamt fünf erzeugt. Neben dem Array, auf das die Variable `Jagged` verweist, gibt es vier Integer-Arrays, die von `Jagged(0)` etc. referenziert werden.

Ein Nachteil dieser Konstruktion gegenüber mehrdimensionalen Arrays ist die geringere Zugriffsgeschwindigkeit. Bei einem mehrdimensionalen Array kann direkt aus der Dimensionierung und der Größe des enthaltenen Datentyps zu jedem vorgegebenen Index die Adresse des Elementes schnell und einfach errechnet werden. Die Prozessoren stellen hierfür optimierte Befehle zur Verfügung. Bei der *Jagged*-Variante muss bei jedem Zugriff die gesamte Verweiskette durchlaufen werden. Das dauert deutlich länger und kann bei sehr großen Arrays ein wesentlicher Performance-Nachteil sein.

Es gibt allerdings noch einen praktischen Vorteil der Jagged-Arrays. Im Gegensatz zu mehrdimensionalen Arrays können Sie Referenzen kompletter Unterarrays z.B. an Funktionen weitergeben. Aus dem Array aus Listing 1.8 können Sie beispielsweise das letzte Unterarray an die Summenfunktion aus Listing 1.6 übergeben:

---

```
Dim sum As Integer = GetSumOfArray(Jagged(3))
```

---

Wie Sie schon gesehen haben, ist die Vorbelegung eines Jagged-Arrays etwas aufwändiger. Auch diese können Sie natürlich wieder in einer Anweisung zusammenfassen. Die obige Definition können Sie auch so schreiben:

---

```
Dim Jagged()() As Integer = New Integer()() { _
    New Integer(8) {}, _
    New Integer() {1, 2, 3, 4}, _
    New Integer() {123}, _
    New Integer() {10, 20, 30, 40, 50} _
}
```

---

VB.NET bietet Ihnen mehrere syntaktische Varianten für die Definition von Arrays. Wählen Sie die, die Ihnen am meisten zusagt.

## Arrays und For Each-Schleifen

Arrays bringen die Voraussetzungen mit, um sie mit einer `For Each`-Schleife durchlaufen zu können. Einsetzbar sind solche Schleifen dann, wenn jedes Element eines Arrays gleichermaßen behandelt werden muss und die Reihenfolge der Abarbeitung keine Rolle spielt. Denn die Reihenfolge können Sie nicht vorgeben. Sie ist fest verdrahtet:

---

```
Dim Numbers(,) As Integer = {{1, 2}, {3, 4}, {5, 6}}
Dim i As Integer
For Each i In Numbers
    Debug.Write(i & " ")
Next
```

---

Der Code erzeugt die folgende Ausgabe:

---

```
1 2 3 4 5 6
```

---

Müssen die Dimensionen in einer anderen Reihenfolge durchlaufen werden, können Sie `For Each` nicht einsetzen und müssen stattdessen auf verschachtelte Index-gesteuerte Schleifen zurückgreifen.

Bei Jagged-Arrays durchläuft eine `For Each`-Schleife nur das eindimensionale Feld. Für alle Unterarrays müssen Sie weitere Schleifen bilden.

## 9 Listen

Immer dann, wenn die endgültige Anzahl zu speichernder Elemente nicht festgelegt werden kann, sollten Sie statt Arrays Auflistungen verwenden. Eine allerdings nicht mehr: Die gute alte `Collection`-Klasse aus VB6 sollten Sie nicht mehr einsetzen, da sie VB-spezifisch ist.

Das Framework stellt eine ganze Reihe von Auflistungsklassen zur Verfügung, die für die unterschiedlichsten Einsatzzwecke vorgesehen sind. Die wichtigsten davon wollen wir Ihnen kurz vorstellen.

### ArrayList

Nahe verwandt mit Arrays ist die Klasse `ArrayList`. Sie benutzt intern ein `Object`-Array zur Speicherung. Über Methoden wie `Add`, `AddRange`, `Insert` und `InsertRange` können nach Belieben `Object`-Referenzen hinzugefügt werden. Reicht die Größe des internen Arrays nicht aus, wird es automatisch durch ein größeres ersetzt und die Inhalte kopiert. Als Benutzer eines `ArrayList` bemerken Sie davon nichts.

Über eine indizierte Default-Eigenschaft lässt sich ein `ArrayList`-Objekt ansprechen, als wäre es ein Array:

---

```
Dim list As New ArrayList
list.Add("One")
list.Add("Two")
Debug.WriteLine(list(1))
```

---

Denken Sie immer daran, dass intern Referenzen vom Typ `Object` gespeichert werden. Möchten Sie mit einer Referenz weiterarbeiten, benötigen Sie einen `Typecast`:

---

```
Dim t As String
t = DirectCast(list(1), String)
```

---

oder:

---

```
t = CStr(list(1))
```

---

Natürlich lassen sich auch Wertetypen der Liste hinzufügen. Diese werden automatisch »geboxt« und müssen beim Rücklesen ebenfalls gecastet werden:

---

```
list.Add(123)
Dim k As Integer = CInt(list(2))
```

---

Die Methode `Add` liefert beim Hinzufügen eines Elementes als Rückgabewert dessen Index. Über diesen Index können Sie das gerade hinzugefügte Element in der Liste erreichen. Allerdings funktioniert das nur, solange keine Elemente vor dieser Position gelöscht oder hinzugefügt werden, da sich die Indizierung dadurch ändern würde.

Durchlaufen können Sie ein `ArrayList` wahlweise mit einer Iterationsschleife, bei der die Eigenschaft `Count` die Anzahl der Elemente wiedergibt:

---

```
For i As Integer = 0 To list.Count - 1
    Debug.WriteLine(list(i))
Next
```

---

oder mit einer `For Each`-Schleife, bei der wahlweise eine allgemeine `Object`-Referenz oder, wenn nur Referenzen eines bestimmten Typs gespeichert sind, eine speziellere Referenz als Schleifenvariable vorgesehen wird:

---

```
For Each obj As Object In list
    Debug.WriteLine(obj)
Next
```

---

Während Sie mit `Add` oder `Insert` nur jeweils ein einziges Element hinzufügen können, können Sie mit `AddRange` oder `InsertRange` eine Liste oder ein Array einflechten:

---

```
Dim strings() As String = {"aaa", "bbb", "ccc"}
list.AddRange(strings)
list.InsertRange(3, New String() {"Three", "Four", "Five"})
```

---

Hinweis

Wenn Sie Strukturen in einem `ArrayList` speichern wollen, bedenken Sie, dass diese via `Boxing` in Objekte gewandelt werden und die Referenzen dieser Objekte im `ArrayList` gespeichert werden. Sie haben dann keine Möglichkeit, einzelne Member einer solchen Struktur zu ändern!

Mit `RemoveAt` oder `RemoveRange` können Sie analog ein Element oder eine Teilliste aus der Auflistung entfernen:

---

```
list.RemoveAt(3)
list.RemoveRange(2, 4)
```

---

Mit der Methode `Clear` löschen Sie den gesamten Inhalt der Liste. Die Methoden `Sort` und `BinarySearch` werden zum Sortieren bzw. binären Suchen in einer sortierten Liste verwendet. Sie arbeiten genauso wie `Array.Sort` bzw. `Array.BinarySearch` und werden im Rezepteteil beschrieben.

Eine besonders nützliche Methode ist `ToArray`, mit deren Hilfe der gesamte Inhalt der Liste in ein Array kopiert werden kann. Die erste Überladung erzeugt ein `Object`-Array:

---

```
Dim elements() As Object = list.ToArray()
```

---

Die andere Überladung kann ein spezialisiertes Array anlegen und füllen. Hierzu muss dem Aufruf von `ToArray` das entsprechende Typ-Objekt übergeben werden:

---

```
Dim vlist As New ArrayList
vlist.Add(New VWGolf("ZZ-X999", 2))
vlist.Add(New VWGolf("ZZ-Z001", 1))
vlist.Add(New VWGolf("ZZ-X997", 2))
vlist.Add(New VWGolf("ZZ-X337", 3))

Dim golfList() As VWGolf
golfList = DirectCast(vlist.ToArray(GetType(VWGolf)), VWGolf())
```

---

So können Sie die Liste zunächst mit einer unbekannten Anzahl von Elementen aufbauen und abschließend ein Array vom gewünschten Typ generieren.

Ein `ArrayList`-Objekt lässt sich auch gegen Löschen und Hinzufügen von Elementen sichern. Die statische Methode `ArrayList.ReadOnly` erzeugt einen Wrapper (Hülle) um ein `ArrayList`-Objekt, der weder das Hinzufügen noch das Entfernen von Elementen erlaubt. So erstellen Sie ein schreibgeschütztes `ArrayList`-Objekt:

---

```
'Wrapper aus vorhandener Liste Vehicles erstellen
Dim ReadOnlyVehicles As ArrayList = ArrayList.ReadOnly(Vehicles)

'Die folgenden beiden Zeilen führen zu einem Laufzeitfehler
ReadOnlyVehicles.Add(New Vehicle("Mercedes", "600S", 220))
ReadOnlyVehicles.RemoveAt(0)
```

---

Wenn die `Add`- oder die `RemoveAt`-Methode aufgerufen wird, wird ein Laufzeitfehler generiert. So wird sichergestellt, dass die Liste nicht verändert wird.

Der Schutz erstreckt sich aber nicht auf die referenzierten Objekte:

---

```
'Die Objekte selbst sind nicht geschützt
CType(ReadOnlyVehicles(0), Vehicle).Manufacturer = "xxx"
```

---

Da die Liste ja lediglich die Referenzen der Objekte beinhaltet, lassen sich die Klasseninstanzen selbst beliebig ändern und können nicht geschützt werden.

## Hashtable

Bei einer `Hashtable` handelt es sich um eine `Key/Value`-Auflistung. Hinzugefügt werden jeweils die Referenz eines beliebigen Daten-Objektes in Verbindung mit der Referenz eines Schlüssel-Objektes. Der Schlüssel kann prinzipiell beliebig aufgebaut sein, meist werden aber Zeichenketten verwendet:

---

```
Dim list As New Hashtable
list.Add("eins", New VWGolf("A-A001", 2))
list.Add("zwei", New VWGolf("A-A002", 1))
list.Add("drei", New VWGolf("A-A003", 3))
```

---

Über eine indizierte Standard-Eigenschaft kann ein Element in der Liste gesucht werden. Hierzu wird als Index der Schlüssel übergeben:

---

```
Dim golf As VWGolf = DirectCast(list("zwei"), VWGolf)
Debug.WriteLine(golf.LicenceNumber)
```

---

führt zur Ausgabe von:

---

A-A002

---

Mit Remove kann unter Angabe des Schlüssels ein Element wieder aus der Liste entfernt werden:

---

```
list.Remove("drei")
```

---

Schlüssel- und Werteliste können auch getrennt voneinander abgefragt und durchlaufen werden:

---

```
For Each car As VWGolf In list.Values
    Debug.WriteLine(car.LicenceNumber)
Next

Debug.WriteLine("*****")

For Each key As String In list.Keys
    Debug.WriteLine("Schlüssel: " & key)
Next
```

---

erzeugen die Ausgabe:

---

A-A002  
A-A001  
A-A003  
\*\*\*\*\*  
Schlüssel: zwei  
Schlüssel: eins  
Schlüssel: drei

---

## SortedList

Eine SortedList verhält sich wie eine Mischung aus Array und Hashtable. Einerseits arbeitet sie genau wie eine Hashtable mit Key/Value-Paaren, andererseits kann über die Methode `GetByIndex` auch über einen Index auf ein Element zugegriffen werden. Das Hinzufügen von Elementen und Durchlaufen der Liste erfolgt genauso wie bei der Hashtable:

---

```
Dim list As New SortedList
list.Add("eins", New VWGolf("A-A001", 2))
list.Add("zwei", New VWGolf("A-A002", 1))
list.Add("drei", New VWGolf("A-A003", 3))

For Each car As VWGolf In list.Values
    Debug.WriteLine(car.LicenceNumber)
Next
```

---

Allerdings wird hier die Ausgabe nach den Schlüsseln aufsteigend sortiert:

---

```
A-A003 - Schlüssel: drei
A-A001 - Schlüssel: eins
A-A002 - Schlüssel: zwei
```

---

Da im Beispiel die Schlüssel String-Objekt sind, wird alphabetisch sortiert. Die Schlüssel müssen entweder `IComparable` implementieren oder der Liste muss ein `IComparer`-implementierendes Objekt zugewiesen werden.

Die `SortedList` ist meist langsamer als eine `Hashtable`, bietet aber mehr Funktionalität und Flexibilität.

## StringDictionary

Eine Sonderform einer `Hashtable` bildet die Klasse `StringDictionary`. Grundsätzlich handelt es sich dabei um eine `Hashtable`, bei der sowohl der Typ für die Schlüssel als auch für die Werte auf den Datentyp `String` festgelegt wurde. So könnte z.B. eine Anwendung eines `StringDictionary`-Objektes aussehen:

---

```
Dim list As New System.Collections.Specialized.StringDictionary
list.Add("AW", "www.Addison-Wesley.de")
list.Add("MS", "www.microsoft.com")
list.Add("T", "www.tagesschau.de")

Debug.WriteLine(list("AW"))
```

---

Durch die Spezialisierung auf Strings ist `StringDictionary` in der Regel schneller als eine `Hashtable`.

## Weitere Auflistungen

In den Namensräumen `System.Collection` und `System.Collections.Specialized` finden Sie noch eine Reihe weiterer Auflistungsklassen, die alle für spezielle Anwendungsfälle gedacht sind. Lesen Sie in der MSDN-Doku nach, wie diese Klassen eingesetzt werden können.

Sie können auch eigene typsichere Auflistungen programmieren. Hierzu dienen die Klassen `CollectionBase` und `DictionaryBase` als Basisklassen.

## 10 CLS-Kompatibilität

Einige Konstruktionen, die die Syntax von Visual Basic .NET zulässt und die auch innerhalb vom Visual Basic .NET genutzt werden können, sind nicht CLS-konform. Andere Sprachen können sie nicht oder zumindest nur eingeschränkt nutzen. Dazu gehören beispielsweise Eigenschaften mit Parametern, die nicht mit dem Schlüsselwort `Default` als Standard-Eigenschaft gekennzeichnet sind sowie Methoden mit optionalen Parametern.

Um sicherstellen zu können, dass eine Assembly oder eine Klassendefinition CLS-konform ist und ohne Einschränkung in anderen Sprachen genutzt werden kann, ist das Attribut `CLSCompliant` vorgesehen, das Sie jeder Definition voranstellen können und auch global in der Assembly setzen können. Verstöße gegen die CLS-Kompatibilität werden dann beim Kompilieren gemeldet. Allerdings gilt für Visual Basic .NET in Visual Studio 2003 bisher eine Ausnahme. Zitat aus der MSDN-Doku:

»Der aktuelle Microsoft Visual Basic-Compiler generiert bewusst keine CLS-Kompatibilitätswarnungen. Künftige Versionen des Compilers werden diese Warnung allerdings ausgeben.«

Auch, wenn es zurzeit noch keine Hilfe ist, ist es langfristig gesehen durchaus sinnvoll, bereits jetzt in Klassenbibliotheken, die zukünftig auch in anderen Projekten genutzt werden sollen, das `CLSCompliant`-Attribut zu verwenden.





# Visual Studio

Visual Studio bietet allerlei, teilweise versteckte nützliche Features. Einige der etwas weniger bekannten wollen wir Ihnen hier kurz vorstellen.

## 11 Texte in der Toolbox zwischenspeichern

Wenn Sie einen Textblock in mehreren Code-Fenstern wieder verwenden wollen, können Sie ihn einfach zur Zwischenspeicherung auf die Toolbox ziehen (siehe Abbildung 3). Wollen Sie den Text über einen längeren Zeitraum dort ablegen, sollten Sie ihm über das Kontextmenü einen sinnvollen Namen geben. Zur Wiederverwendung ziehen Sie den Eintrag aus der Toolbox einfach an die gewünschte Stelle im Code-Fenster.

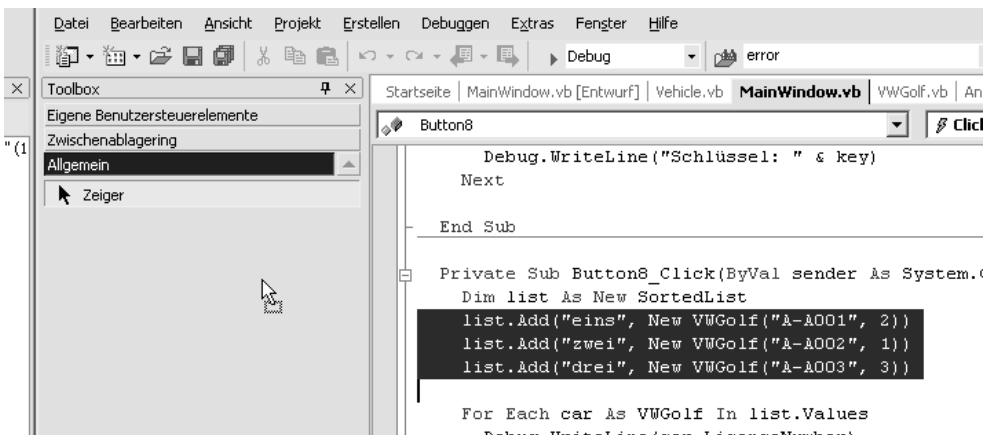


Abbildung 3: Mehrfach benötigte Texte können Sie auf die Toolbox ziehen

Falls Sie viel mit Copy&Paste arbeiten, werfen Sie einmal einen Blick in die Registerkarte »Zwischenablagierung« der Toolbox. Dort werden in derselben Weise die mit »Kopieren« in die Zwischenablage gelegten Textblöcke in der Toolbox aufgeführt. Auch diese können Sie wieder in ein Code-Fenster ziehen.

## 12 Standard-Einstellungen für Option Strict

Um nicht in jeder Datei die Einstellungen für Option Strict, Option Explicit und Option Compare wiederholen zu müssen, können Sie sie auch projektweit einstellen. Rufen Sie hierzu den Menüpunkt PROJEKT/EIGENSCHAFTEN auf und wählen Sie dann in der linken TreeView-Darstellung den Eintrag ERSTELLEN. Es wird der in Abbildung 4 gezeigte Einstellungsdialog angezeigt.

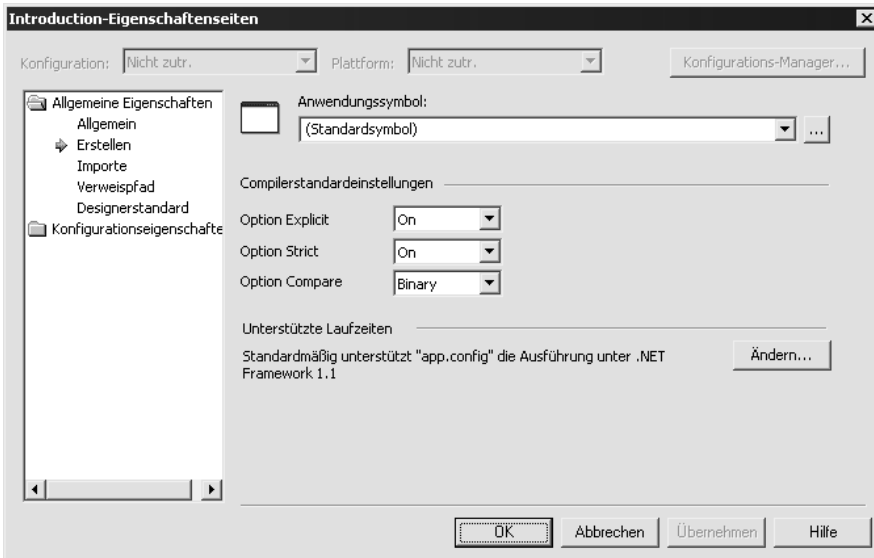


Abbildung 4: Projektweite Einstellung für Option Strict und andere Optionen

## 13 Projektweite Imports-Einstellungen

Benötigen Sie bestimmte Namensräume im gesamten Projekt und wollen sie nicht in jeder Datei mit Imports aufführen müssen, dann können Sie auch diese projektweit hinzufügen bzw. wieder entfernen. Wiederum über die Projekteigenschaften wählen Sie hierzu den Eintrag **Importe** aus. In dem in Abbildung 5 gezeigten Dialog können Sie neue Namensräume hinzufügen oder vorhandene löschen. Die entsprechenden Imports-Anweisungen können dann entfallen.

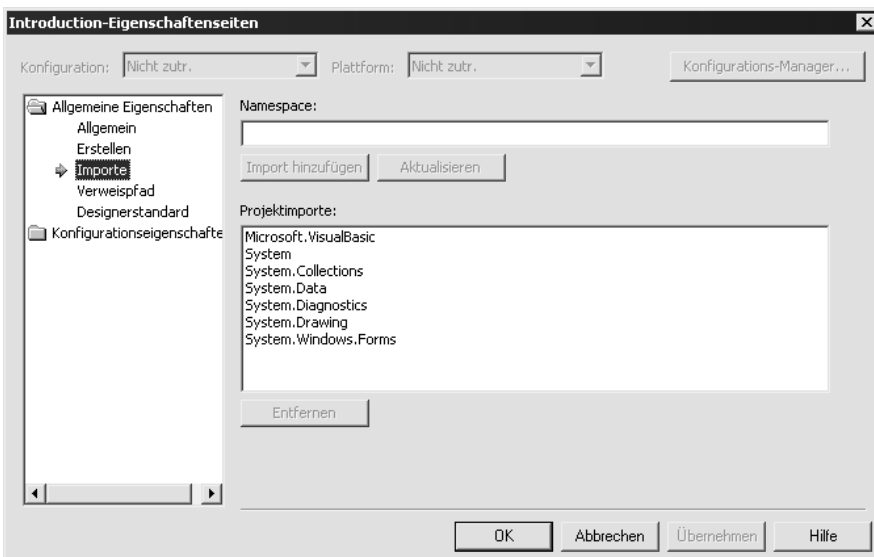


Abbildung 5: Namensräume können allen Dateien eines Projektes automatisch hinzugefügt werden

## 14 Steuerelemente und Fensterklasse im Entwurfsmodus debuggen

Verhalten sich die selbst programmierten Steuerelemente oder abgeleiteten Fensterklassen im Designer einmal nicht so, wie sie sollten, dann wird schnell der Ruf nach Debug-Möglichkeiten laut. Während Ihnen zur Laufzeit alle Möglichkeiten des Debuggers offen stehen, werden im Entwurfsmodus Ausgaben mit `Debug.WriteLine` oder `MessageBox.Show` sowie Breakpoints ignoriert. Wäre das nicht so, hätte man wahrscheinlich ein rekursives Debug-Problem, das nur Verwirrung stiften würde.

Trotzdem lassen sich Steuerelemente und Fenster auch im Entwurfsmodus mit dem Debugger testen. Voraussetzung ist, dass sich die zu testenden Klassen in einer Klassenbibliothek befinden, die von dem Projekt, in dem sie eingesetzt werden sollen, referenziert wird. Als Beispiel soll die Klassenbibliothek *GuiControls.dll* dienen, die die oben beschriebenen Steuerelemente enthält.

Um eine Klassenbibliothek debuggen zu können, müssen Sie in den Projekteinstellungen ein Programm wählen, das gestartet werden soll (Abbildung 6). Zweckmäßigerweise gibt man hier den Pfad der Entwicklungsumgebung an. Wenn bereits ein Projekt existiert, das die Bibliothek verwendet, können Sie dessen Pfad unter `STARTOPTIONEN / BEFEHLSZEILENARGUMENTE` eingeben. Beim Start des Debuggers wird nun eine zweite Instanz der Entwicklungsumgebung gestartet. Sie können wie gewohnt ein Projekt laden, ein neues anlegen oder mit dem ggf. automatisch geöffneten Projekt arbeiten.

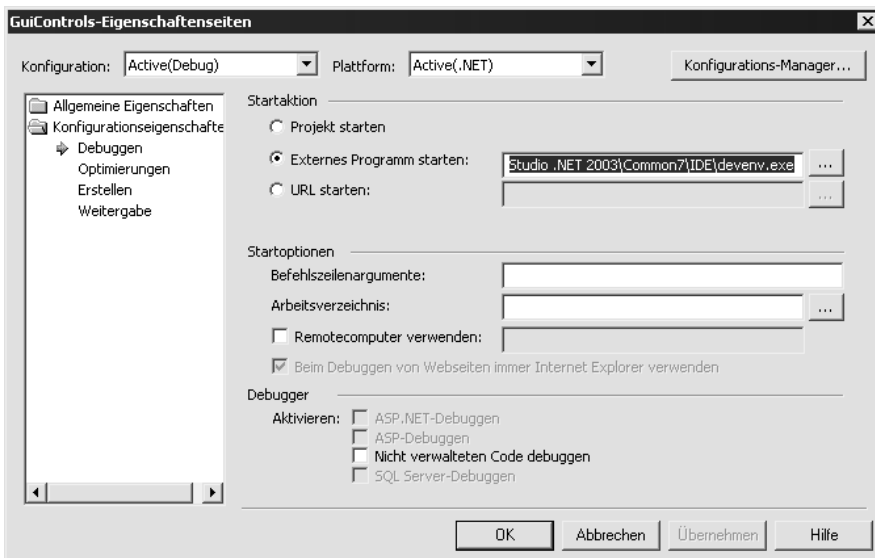


Abbildung 6: Der Debugger soll eine zweite Instanz des Visual Studios starten

In der ersten Instanz der Entwicklungsumgebung, die ja die zu testenden Klassen enthält, können Sie nach Belieben Breakpoints setzen, Einzelschritte ausführen, Variablen-Inhalte einsehen und ändern und so weiter. Es steht Ihnen die ganze Palette der Debugger-Funktionen zur Verfügung. Setzen Sie z.B. einen Haltepunkt auf die Wertezuweisung in der Eigenschaft `Ticks` des `TimeSliceControls`. Wenn Sie im anderen Projekt dieses Steuerelement einsetzen und im Eigenschaftsfenster des Designers der Eigenschaft einen anderen Wert zuweisen, wird der Ablauf an diesem

Haltepunkt unterbrochen (siehe Abbildung 7 und Abbildung 8). Es wird automatisch die andere Instanz der Entwicklungsumgebung in den Vordergrund geholt, so dass Sie direkt den Debugger nutzen können.

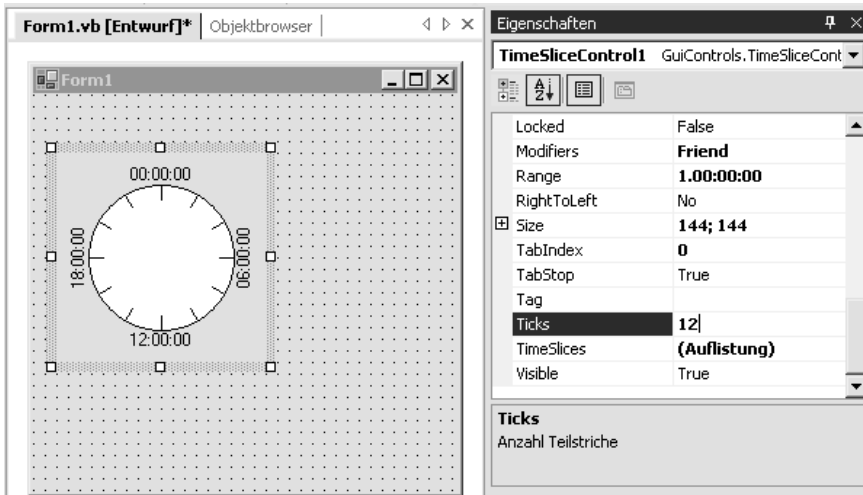


Abbildung 7: Änderung der Eigenschaft Ticks lässt den Debugger stoppen

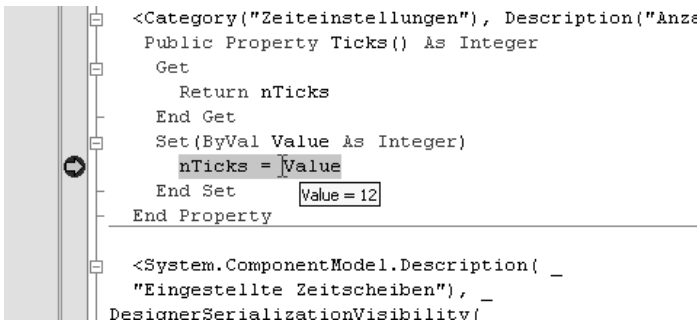


Abbildung 8: Reaktion auf die Wertänderung der Eigenschaft Ticks im Designmodus

Um im zweiten Projekt weiterarbeiten zu können, müssen Sie die erste Instanz wieder in den Run-Modus versetzen. Vergessen Sie nicht, dass der Prozess, der mit einem Breakpoint unterbrochen ist, die zweite Instanz der Entwicklungsumgebung ist. Ihre Klassenmethode, in der der Haltepunkt angelaufen wird, ist Bestandteil dieses Prozesses!

#### Achtung

Nutzen Sie die beschriebene Vorgehensweise wirklich nur zur Fehlersuche in der Klasse des Basisfensters bzw. des Steuerelementes. Seien Sie besonders achtsam, wenn Sie in der zweiten Instanz der Entwicklungsumgebung Änderungen vornehmen, die nicht sofort gespeichert werden. Wird der Programmablauf in der zu untersuchenden Klassenbibliothek abgebrochen, sei es durch einen Laufzeitfehler oder durch eine Benutzeraktion, dann wird damit die zweite Instanz der Entwicklungsumgebung (schlagartig) beendet. Eine Abfrage, ob Änderungen gespeichert werden sollen, kann nicht mehr erfolgen. Alle Änderungen sind dann verloren.

## 15 Verknüpfung einer Datei einem Projekt hinzufügen

Wenn Sie über PROJEKT / VORHANDENES ELEMENT HINZUFÜGEN... eine Datei dem Projekt hinzufügen, dann wird eine Kopie der Datei im Projektverzeichnis angelegt. Möchten Sie aber mit der Originaldatei arbeiten, dann benötigen Sie eine Verknüpfung. Hierzu klicken Sie auf den kleinen Pfeil rechts neben der ÖFFNEN-Schaltfläche im Dialog (siehe Abbildung 9).

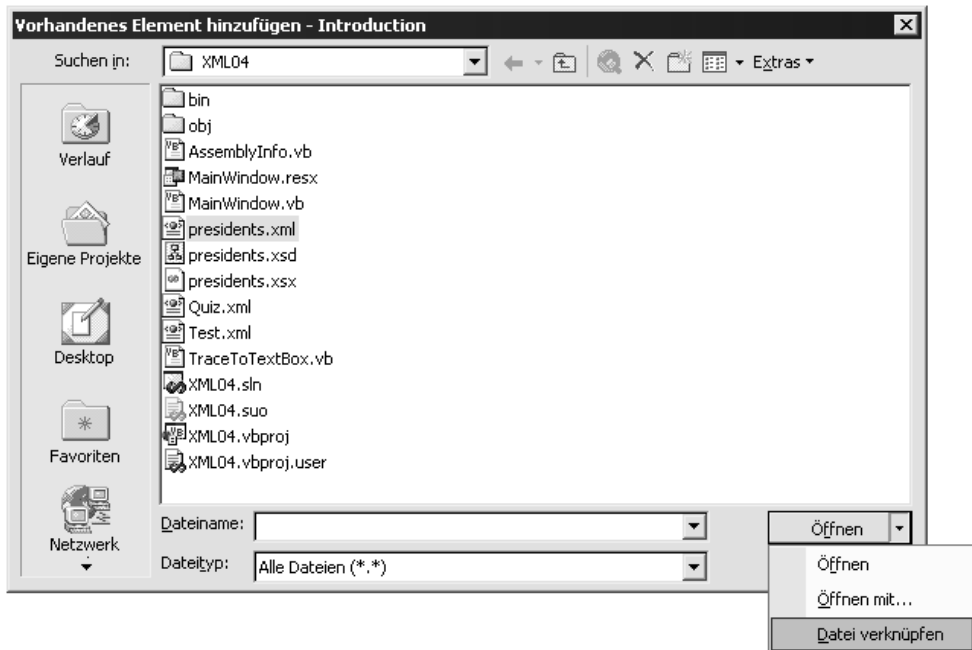


Abbildung 9: Datei nicht kopieren, sondern verknüpfen

## 16 Dokumentgliederung anzeigen und nutzen

Besonders interessant für XML-Dateien ist die Möglichkeit, eine Gliederungsansicht anzuzeigen (Abbildung 10). Wird das entsprechende Fenster nicht dargestellt, können Sie es über ANSICHT / ANDERE FENSTER / DOKUMENTGLIEDERUNG anzeigen lassen.

In der TreeView-Ansicht (in der Abbildung auf der linken Seite) können Sie einen Knoten auswählen. Die Datei-Ansicht wird dann automatisch so positioniert, dass dieser Knoten sichtbar ist. Die Dokumentgliederung hat allerdings den Nachteil, dass sie für große XML-Dateien sehr viel Zeit zum Aufbau benötigt. Glücklicherweise geschieht dieser Aufbau im Hintergrund, so dass andere Tätigkeiten nicht behindert werden.

## 17 Tabellenansicht einer XML-Datei

Sofern die Struktur einer XML-Datei es erlaubt, können Sie die Daten auch in Tabellenform ansehen und ändern. Zum Umschalten klicken Sie auf die Schaltfläche DATEN unterhalb des XML-Editor-Fensters (Abbildung 11).

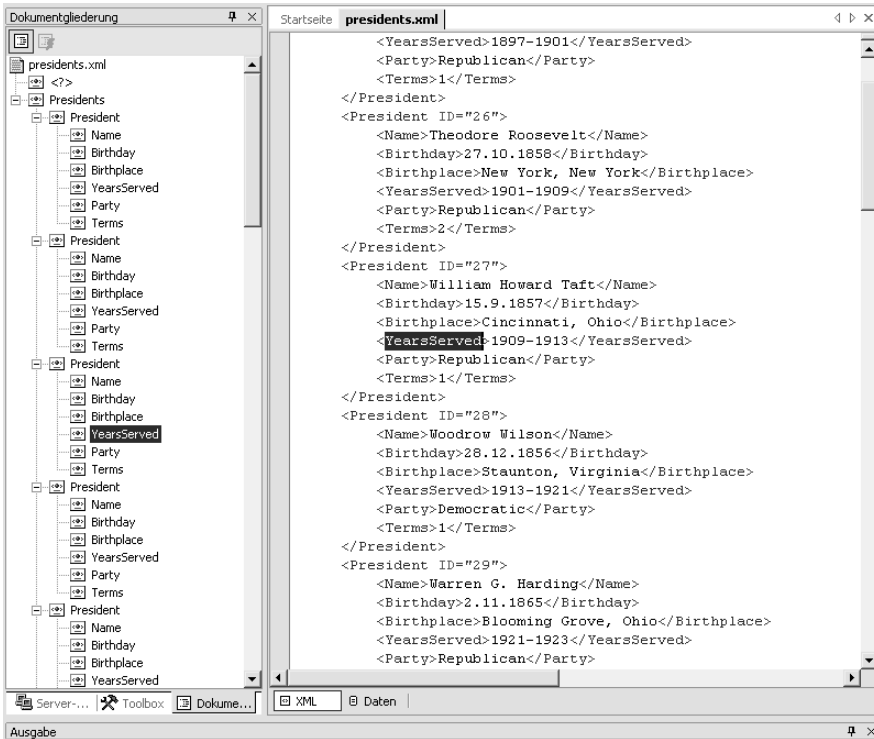


Abbildung 10: Dokumentgliederung einer XML-Datei

Startseite **presidents.xml**

Datentabellen: Daten:

| President | Daten für President       |            |                     |             |            |                  |    |
|-----------|---------------------------|------------|---------------------|-------------|------------|------------------|----|
|           | Name                      | Birthday   | Birthplace          | YearsServed | Party      | Terms            | ID |
|           | William McKinley          | 29.1.1843  | Niles, Ohio         | 1897-1901   | Republican | 1                | 25 |
|           | Theodore Roosevelt        | 27.10.1858 | New York, New York  | 1901-1909   | Republican | 2                | 26 |
|           | William Howard Taft       | 15.9.1857  | Cincinnati, Ohio    | 1909-1913   | Republican | 1                | 27 |
|           | Woodrow Wilson            | 28.12.1856 | Staunton, Virginia  | 1913-1921   | Democratic | 1                | 28 |
|           | Warren G. Harding         | 2.11.1865  | Bloomington, O      | 1921-1923   | Republican | 1 - died in off  | 29 |
|           | John Calvin Coolidge      | 4.7.1872   | Plymouth, Vermont   | 1923-1929   | Republican | 1 - took over    | 30 |
|           | Herbert Hoover            | 10.8.1874  | West Branch, Iowa   | 1929-1933   | Republican | 1                | 31 |
|           | Franklin D. Roosevelt     | 30.1.1882  | Hyde Park, New York | 1933-1945   | Democratic | 4 - died in 4t   | 32 |
|           | Harry S. Truman           | 8.5.1884   | Lamar, Missouri     | 1945-1953   | Democratic | 2 - took over    | 33 |
|           | Dwight D. Eisenhower      | 14.10.1890 | Denison, Texas      | 1953-1961   | Republican | 2                | 34 |
|           | John F. Kennedy           | 29.05.1917 | Brookline, Massach  | 1961-1963   | Democratic | 1 - assassinated | 35 |
|           | Lyndon B. Johnson         | 27.08.1908 | Stonewall, Texas    | 1963-1969   | Democratic | 1 - took over    | 36 |
|           | Richard M. Nixon          | 09.01.1913 | Yorba Linda, Calif  | 1969-1974   | Republican | 2 - resigned     | 37 |
|           | Gerald R. Ford            | 14.07.1913 | Omaha, Nebraska     | 1974-1977   | Republican | Took over for    | 38 |
|           | Jimmy Carter              | 01.10.1924 | Plains, Georgia     | 1977-1981   | Democratic | 1                | 39 |
|           | Ronald Reagan             | 06.02.1911 | Tampico, Illinois   | 1981-1989   | Republican | 2                | 40 |
|           | George Bush               | 12.06.1924 | Milton, Massachuse  | 1989-1993   | Republican | 1                | 41 |
|           | William Jefferson Clinton | 19.08.1946 | Hope, Arkansas      | 1993-2001   | Democratic | 2                | 42 |
|           | George W. Bush            | 06.07.1946 | New Haven, Conne    | Present     | Republican |                  | 43 |

\*

Abbildung 11: Tabellenansicht einer XML-Datei

## 18 XML-Schema für vorhandene XML-Datei erstellen und bearbeiten

Wird eine XML-Datei im Editor von Visual Studio angezeigt, dann können Sie über das Menü XML / SCHEMA ERSTELLEN ein Schema automatisch generieren lassen. In der XML-Datei wird dann der Namensraum hinzugefügt, z.B. so:

---

```
<Presidents xmlns="http://tempuri.org/presidents.xsd">
```

---

Die erzeugte Schema-Datei können Sie dann entweder in der DataSet-Ansicht (Abbildung 12) oder in der XML-Ansicht (Abbildung 13) bearbeiten. Beachten Sie allerdings, dass automatisch generierte Schema-Dateien relativ simpel aufgebaut sind.

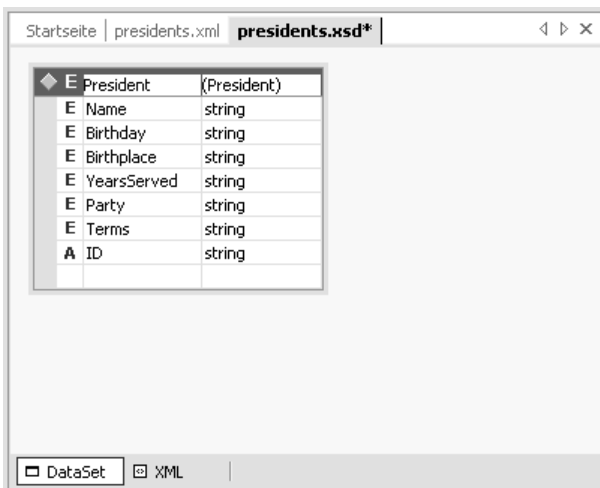


Abbildung 12: XML-Schema in der DataSet-Ansicht

Über die DataSet-Ansicht können Sie ein Schema bequem anlegen oder erweitern.

## 19 Navigation über die Klassenansicht

Bei komplexeren Klassenstrukturen ist es manchmal einfacher, über die baumförmige Klassenansicht zu navigieren, als über den Projektmappen-Explorer. Falls die Ansicht nicht dargestellt wird, aktivieren Sie sie über ANSICHT / KLASSENANSICHT. Durch einen Doppelklick auf eine Methode, Konstruktor, Member-Variable etc. gelangen Sie direkt zur Definition im entsprechenden Code-Fenster. Sie können diese Ansicht nach verschiedenen Kriterien sortieren und gruppieren.



Abbildung 13: XML-Schema in der XML-Ansicht

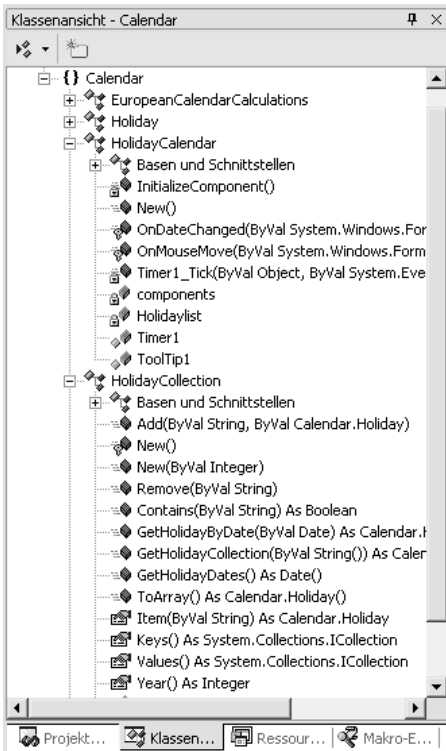


Abbildung 14: Gegliederte Ansicht aller zum Projekt gehörenden Klassen



# Internetquellen

Nicht nur bei Microsoft, sondern auf vielen privaten und kommerziellen Websites finden Sie heute Informationen zu .NET, zum Framework, zu Visual Basic .NET und allem, was dazugehört. Die Erfahrung zeigt, dass sich Antworten auf eine aktuelle Fragestellung mit großer Wahrscheinlichkeit im Internet finden lassen. Es gibt weltweit eine große Internetgemeinde, die sich der Diskussion und Lösung von .NET-Problemen verschrieben hat.

## 20 Websites zu .NET

Nachfolgend einige Adressen zu Themen rund um .NET und Visual Basic .NET. Dieses ist bei weitem keine vollständige Liste, sondern stellt nur eine kleine Auswahl dar.

### MSDN

<http://msdn.microsoft.com/>

Dies ist die Homepage des Microsoft Developer Networks. Hier finden Sie alle Informationen rund um die Entwicklung unter Windows. Unter

<http://msdn.microsoft.com/developercenters/>

finden Sie den Direkteinstieg zu den verschiedenen Programmierthemen. Während Sie einen großen Teil der Informationen auch auf den mit Visual Studio ausgelieferten MSDN-CDs wiederfinden, werden die Web-Seiten ständig aktualisiert und durch Fachartikel ergänzt. Die Sprache ist dort allerdings in der Regel Englisch.

### Visual Basic .NET Resource Kit

<http://msdn.microsoft.com/vbasic/vbrkit/>

Viele Beispiele zu unterschiedlichen Rubriken

### Got Dot Net

<http://www.gotdotnet.com/>

Beispiele und Diskussionen

Artikel mit Beispielen unter

<http://samples.gotdotnet.com/quickstart/howto/>

### The Code Project

<http://www.codeproject.com/vb/net/>

Gute Artikel mit Beispielen und Downloads

### .NET 247

<http://www.dotnet247.com/247reference/namespaces.aspx>

Beispiele und Diskussionen

## George Shepherd's Windows Forms FAQ

<http://www.syncfusion.com/FAQ/Winforms/>

Fragen und Antworten zu Windows Forms

## .NET Xtreme

<http://www.dotnetextreme.com/vb.asp>

Artikel und Beispiele

## .NET Show

<http://msdn.microsoft.com/theshow/Archive.asp>

Interviews zu diversen .NET-Themen als TV-Sendung (schneller Internetzugang erforderlich).

## VB.NET Heaven

<http://www.vbdotnetheaven.com/>

Beispiele und Diskussionen

## VB-2-The-Max

<http://www.vb2themax.com>

Beispiele

## DirectX

<http://msdn.microsoft.com/library/Default.asp?url=/library/en-us/dndxgen/html/directx9devfaq.asp>

MSDN-Startseite zu DirectX

## DOTNETPRO

<http://www.dotnetpro.de/magazine/>

Deutsches Magazin zu diversen .NET-Themen

## Reguläre Ausdrücke

[http://www.rrz.uni-hamburg.de/RRZ/W.Wiedl/Skripte/CGI-Perl/Regulaere\\_Ausdruecke/](http://www.rrz.uni-hamburg.de/RRZ/W.Wiedl/Skripte/CGI-Perl/Regulaere_Ausdruecke/)

[http://www.devmag.net/webprog/regulaere\\_ausdruecke.htm](http://www.devmag.net/webprog/regulaere_ausdruecke.htm)

## 21 Newsgroups

Zu den verschiedenen .NET-Themen gibt es bereits eine Reihe von Newsgroups. Auch deutschsprachige Newsgroups sind im Angebot. Auf dem Server

[msnews.microsoft.com](http://msnews.microsoft.com)

finden Sie unter anderen diese Gruppen:

- ▶ [microsoft.Public.de.german.entwickler.dotnet.vb](#)
- ▶ [microsoft.Public.de.german.entwickler.dotnet.datenbank](#)
- ▶ [microsoft.Public.de.german.entwickler.dotnet.framework](#)

- ▶ microsoft.Public.de.german.entwickler.dotnet.vstudio
- ▶ microsoft.Public.de.german.entwickler.dotnet.csharp
- ▶ microsoft.Public.de.german.entwickler.dotnet.asp

Zusätzlich finden Sie im Internet viele weitere, meist fremdsprachige Newsgroups zu den Themen. Der Microsoft-Server wird von vielen anderen Servern gespiegelt. So finden Sie beispielsweise die genannten Newsgroups auch unter *news.t-online.de*. Einige Provider bieten auf ihren Spiegelserversn aber nicht alle Gruppen an.

Zugreifen können Sie auf eine Newsgroup am besten über ein Programm wie Outlook Express. Dort richten Sie ein neues Konto ein, für das Sie Ihren Namen, eine EMail-Adresse und den Server angeben müssen. Anschließend können Sie die gewünschten Gruppen abonnieren.

#### Achtung

Vorsicht bei der Angabe der EMail-Adresse. Leider werden die EMail-Adressen aus den Newsgroups zunehmend für unseriöse Zwecke (Spam-Mails) missbraucht. Sie sollten daher Ihre EMail-Adresse durch Zusätze so verändern, dass kein automatisches Programm sie nutzen kann, dass aber trotzdem andere Newsgroup-Teilnehmer Sie erreichen können:

z.B. *Keine.Werbung.MeinName@MeinProvider.de*

Ihren Namen sollten Sie vollständig angeben. Abgekürzte Namen oder gar Phantasienamen gelten als äußerst unhöflich und werden regelmäßig gerügt. Es gelten gewisse Regeln (Netiquette oder Netikette genannt), die beachtet werden sollten. Hinweise finden Sie unter

<http://support.microsoft.com/Default.aspx?scid=fh;DE;NGNetikette>

<http://www.afaik.de/usenet/faq/>

<http://www.chemie.fu-berlin.de/outerspace/netnews/netiquette.html>

<http://groups.google.com/groups?hl=en&group=de.newusers.infos>

In Newsgroups werden gestellte Fragen von anderen Teilnehmern beantwortet bzw. diskutiert. Die Qualität der Diskussionen hängt stark vom Umgangston ab. Newsgroups sind keine anonymen Chatrooms, in denen jeder beliebigen Unsinn von sich geben kann, sondern dienen der technischen Diskussion. Sie selbst können Fragen stellen oder auch auf Fragen anderer antworten. Oft lernt man auch aus den Fragen anderer Teilnehmer und der anschließenden Diskussion einiges zur .NET-Programmierung hinzu. Scheuen Sie sich auch nicht, einen Blick in die C#-Newsgroups zu riskieren. Denn oft werden dort Themen diskutiert, die sich nicht auf die Sprache beziehen, sondern eher den Umgang mit dem Framework betreffen.

Bevor Sie allerdings selbst eine Frage in einer Newsgroup stellen, sollten Sie versuchen, die Frage mithilfe der MSDN-Doku oder einer Google-Recherche (siehe weiter unten) zu klären.

Sie können auch über einen Internet-Browser auf die Newsgroups zugreifen unter

<http://support.microsoft.com/newsgroups/Default.aspx>

Allerdings ist der Weg über das Internet oft extrem langsam. Außerdem kann es etwas lästig werden, dass Sie bei jedem neuen Posting, das Sie absetzen wollen, Ihren Namen und Ihre EMail-Adresse erneut eingeben müssen.

## 22 Recherche mit Google

Die Diskussionen der wichtigsten Newsgroups werden von Suchmaschinen wie z.B. Google (<http://www.google.de/>) archiviert (leider auch Ihre EMail-Adresse, siehe weiter oben). Oft finden Sie die Antwort auf eine Frage bereits mit einer Suche in den Newsgroups. Ein Beispiel für eine Google-Recherche sehen Sie in Abbildung 15.

Auf der Google-Homepage klicken Sie auf den Kartenreiter GROUPS und geben anschließend Ihre Suchbegriffe ein. Wenn Sie in den deutschen .NET-Newsgruppen suchen wollen, empfiehlt es sich, »dotnet german« den Suchbegriffen voranzustellen. Gesucht werden Beiträge, die alle Suchbegriffe enthalten. Im Beispiel wird in den deutschen Newsgroups nach »RotateTransform« gesucht.

Da die Entwicklung schnell voranschreitet und Diskussionen zur Beta2 von 2001 für Sie vielleicht nicht mehr interessant sind, sollten Sie die gefundenen Ergebnisse nach Datum sortieren lassen. Die neuesten Beiträge befinden sich dann oben. Gelegentlich, wie auch in der Abbildung zu sehen, schlägt Google leicht abgewandelte alternative Suchbegriffe vor. Dies kann manchmal hilfreich sein, wenn die Schreibweise nicht klar ist oder mehrere Schreibweisen für einen Begriff möglich sind.

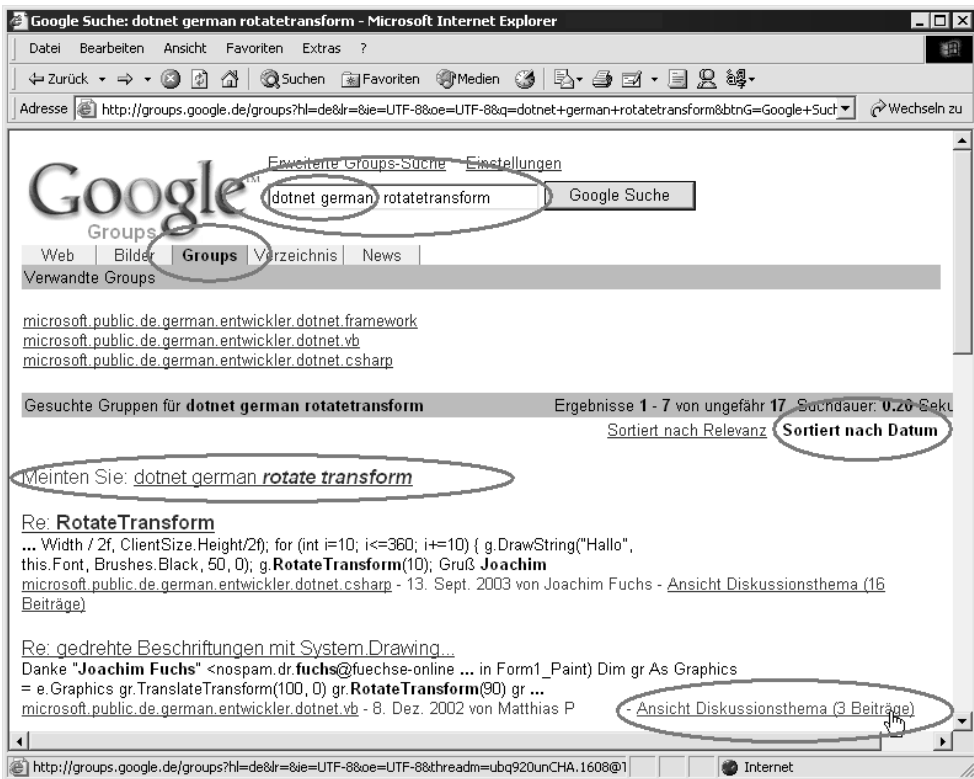


Abbildung 15: Newsgroup-Recherche mit Google

In der Liste der gefundenen Diskussionen können Sie entweder die spezielle Frage oder Antwort anklicken, oder Sie wählen das gesamte Diskussionsthema aus (wie in Abbildung 15 rechts unten markiert). Dann wird Ihnen eine Seite ähnlich der in Abbildung 16 angezeigt.

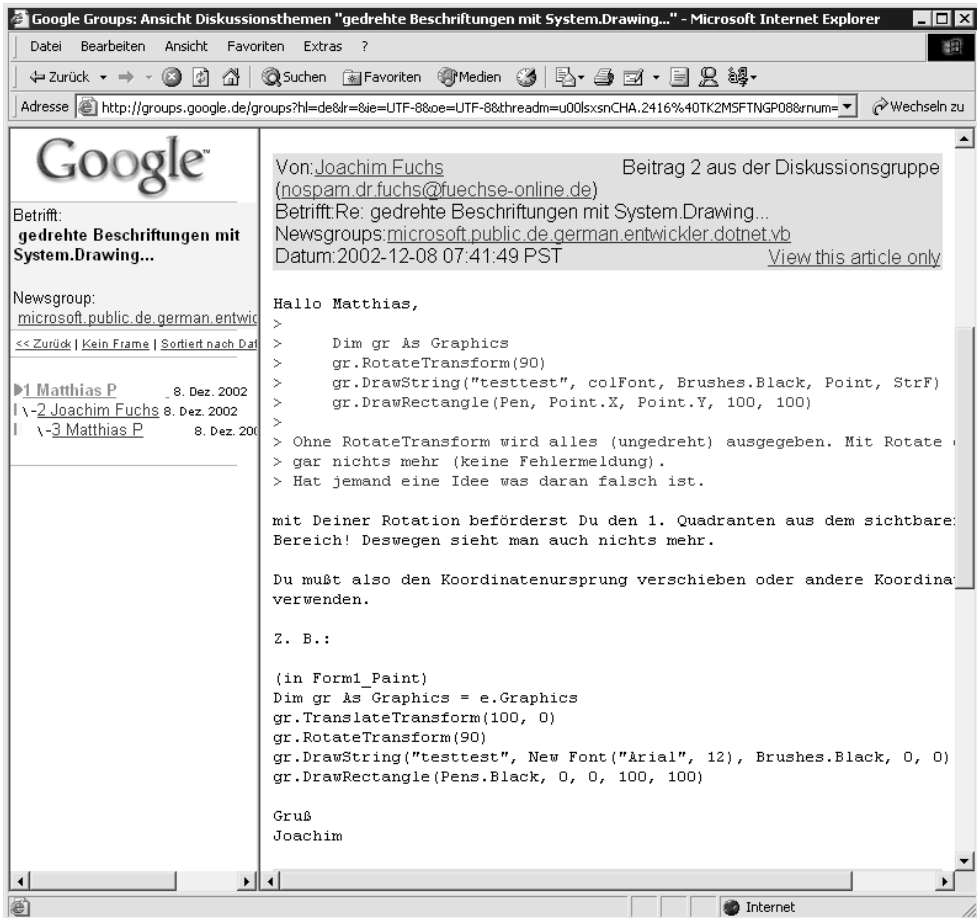


Abbildung 16: Vollständiger Thread eines Diskussionsthemas bei Google

In dieser Ansicht können Sie meist alle Diskussionsbeiträge einsehen, beginnend bei der jeweiligen Fragestellung über alle, oft verzweigten Antworten. Meist finden Sie so schon eine Lösung oder zumindest einen Ansatz für Ihre Problemstellung. Nutzen Sie diese Möglichkeit, auch um Fragestellungen in den Newsgroups zu vermeiden, die schon etliche Male diskutiert und beantwortet worden sind.



# Grundlagen weiterer Technologien

## 23 Kurzer Überblick über WMI

Wenn es um die Beschaffung von Informationen zu Betriebssystem, Hardware und Software geht, gab es lange Jahre nur die Möglichkeit, über spezielle Aufrufe des Betriebssystems sich diese Informationen zu beschaffen. Dies war dann für jedes Betriebssystem, aber auch eventuell für jede Variante eines Betriebssystems mit zusätzlicher Programmieraufgabe verbunden. Ein sehr ineffektives Verfahren, in dem sich auch nicht jeder Programmierer tummeln wollte/konnte. Um die ständige Programmierung immer wieder desselben Themas zu überwinden, schlossen sich 1996 einige Firmen zusammen, zu der WEB-BASED ENTERPRISE MANAGEMENT Gruppe (WBEM). In Zusammenarbeit mit der DESKTOP MANAGEMENT TASK FORCE (DMTF, <http://www.dmtf.org>) wurde das COMMON INFORMATION MODEL (CIM) geschaffen, ein Modell, mit dem allen Software-Herstellern eine portierbare und überschaubare Plattform für Systeminformationen an die Hand gegeben werden sollte. Die Adaption des WBEM-Standards wurde 1998 von Microsoft mit dem Service Pack 4 von Windows NT 4.0 ausgeliefert. Der Name: WINDOWS MANAGEMENT INSTRUMENTATION (WMI).

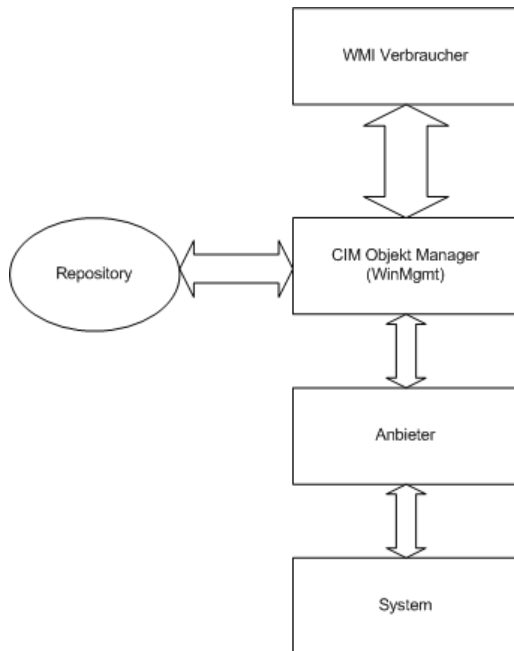


Abbildung 17: WMI-Aufbau

Wie in Abbildung 17 zu sehen ist, wurde WMI im Schichtsystem aufgebaut. Auf der untersten Ebene ist die Systemschicht angelegt, also das Betriebssystem, Dienste des Betriebssystems, allgemeine Programme, Hardwareressourcen. Die Anbieter-Schicht ist ein Vermittler zwischen der Systemschicht und dem eigentlichen Management-System (CIMOM). Hier werden die systemna-

hen Schnittstellen in das COM-System von Windows überführt. Der CIM Objekt Manager verwaltet alle Informationen, die ihm über die diversen Systeme geliefert werden. Statische Informationen werden in einer eigenen objektorientierten Datenbank abgelegt, dem Repository. Daten, die einer ständigen Veränderung unterworfen sind, werden direkt über die Anbieterschicht abgefragt.

Seit diesem Zeitpunkt (1998) hat die Bedeutung von WMI stetig zugenommen, was aber bei den Entwicklern zum Großteil nicht wahrgenommen wurde. Aber Teile so kleiner Applikationen wie dem Exchange Server, dem System Management Server, dem Internet Information Server, Teile des IBM Tivoli, HP Openview basieren auf WMI. Mit Windows 2000 wurden die WMI-Bibliotheken zum Standardbestandteil des Betriebssystems. Für die Betriebssysteme Windows 9x und NT kann man sich entsprechende Bibliotheken von den Webseiten der Firma Microsoft herunterladen. Dort gibt es auch eine Fülle weiterer Informationen, hauptsächlich zum Thema Scripting (VBScript) und WMI.

Die Vorteile von WMI liegen (kurz gefasst) in den folgenden Eigenschaften begründet:

- ▶ Erweiterbarkeit
- ▶ Remote steuerbar
- ▶ Event-Unterstützung
- ▶ SQL-ähnliche Abfragesprache (WQL)
- ▶ Scriptfähig
- ▶ Objektorientiert
- ▶ Nicht nur Abfragen, auch Aktionen möglich

So kann man sich über die Remote- und Event-Unterstützung von wichtigen Ereignissen auf anderen Computern informieren lassen, zum Beispiel wenn der Festplattenplatz zur Neige geht, oder die CPU-Auslastung einen Schwellwert überschreitet.

Im.NET-Umfeld wird WMI durch die Klassen `System.Management` und `System.Management.Instrumentation` repräsentiert. Wie man mit diesen Klassen an die gewünschten Informationen kommt, wird in den einzelnen Rezepten erläutert. Für die Entwicklungsumgebung Visual Studio 2003 kann man sich zusätzlich ein Tool herunterladen, welches die einzelnen Klassen von WMI typfest ummantelt. Der Name ist allerdings etwas missverständlich: »Management (WMI) Extensions for Visual Studio .NET 2003 Server Explorer« (`ManagementExtensions2003.msi`). Im .NET-SDK wird ein entsprechendes Befehlszeilen-Kommando mitgeliefert: `Mgmtclassgen.exe` (siehe Kasten).

Zurzeit gibt es für verwaltete Systeme einige Einschränkungen im Bereich `Instrumentation`. Diese Einschränkungen gelten nicht für nicht-verwalteten-Code (z.B. klassisches C++).

Nach dem Download der Installationsdatei kann diese Erweiterung auf dem üblichen Weg installiert werden. Dabei sollte die Entwicklungsumgebung tunlichst nicht geöffnet sein.

Startet man diese nach der Installation der Erweiterung, so fallen im Server-Explorer die entsprechenden Einträge auf.

Man kann sich in diesem Baum bis zu der (Sub-)Klasse durchklicken, die einen interessiert. Anschließend zieht man diese Klasse auf die Form, wie man es von anderen Wizards gewohnt ist. Man kann auch mit der rechten Maustaste auf diese Klasse klicken und im Kontextmenü »Generate Managed Class« auswählen. Klicken Sie mit der rechten Maustaste bei »Management Classes«, so können Sie über den Kontextmenü-Punkt »Add Classes ...« ein hilfreiches Fenster öffnen, in dem Sie nach bestimmten Klassen suchen können.



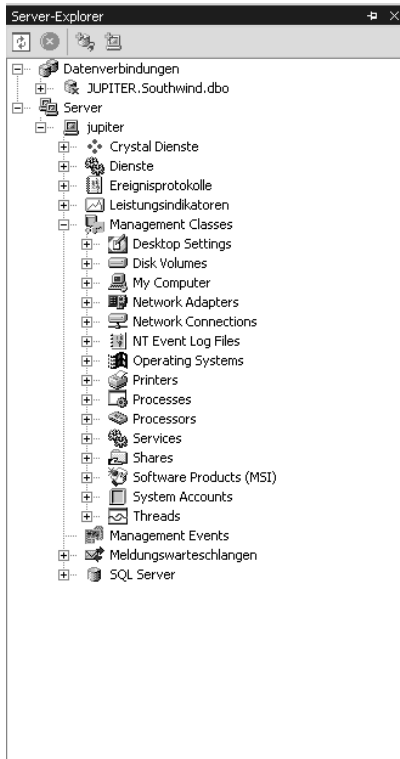


Abbildung 18: Erweiterung im VS

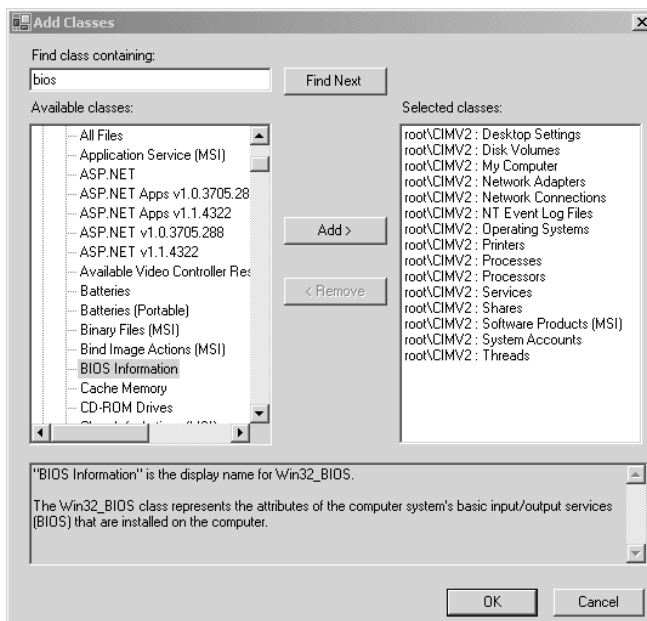


Abbildung 19: WMI-Klassensuche mit dem Server-Explorer

## Hinweis

**Das Management Strongly Typed Class Generator Tool**

Das .NET-SDK liefert ein Tool mit, welches zur Erstellung von strengen Typklassen für WMI vorgesehen ist. Der Aufruf erfolgt nach dem Schema

```
mgmtclassgen WMIKlasse [Optionen]
```

So kann man mit

```
mgmtclassgen Processors /L VB /P C:\TEMP\Prozessor.vb
```

eine Datei für die Prozessorklasse von WMI erstellen. Diese Datei muss dann in das VS-Projekt importiert werden und kann dann verwandt werden.

| Option      | Beschreibung                                                                                                      |
|-------------|-------------------------------------------------------------------------------------------------------------------|
| /L Sprache  | Als Sprache können VB, CS und JS angegeben werden                                                                 |
| /M Computer | Für welchen Computer soll die Klasse angelegt werden. Wird diese Option nicht angegeben, gilt der lokale Computer |
| /P Pfad     | Angabe des Verzeichnisses, in der die Datei erstellt werden soll                                                  |

Tabelle 2: Optionen des MgmtClassGen-Kommandos

Der Klassenaufbau von WMI ist ähnlich dem von .NET. Auch hier gibt es Namensräume und innerhalb dieser Namensräume sind die einzelnen Klassen angesiedelt. Die Anzahl der Klassen ist stetig gewachsen und soll inoffiziellen Schätzungen zufolge in der Version für *Windows Server 2003* bei ca. 5000 Klassen liegen! Um sich hier einen Überblick zu verschaffen, kann man neben dem erweiterten Server-Explorer dies auch nutzen, um sich einen ersten Geschmack für die Programmierung von WMI zu holen.

## 24 XML DOM-Grundlagen

Vorwärtsgerichtete Operationen können mit den Klassen `XmlTextReader` bzw. `XmlTextWriter` einfach und effektiv programmiert werden, wie wir in den Rezepten demonstriert haben. Komplexere Operationen, bei denen die Navigation durch den XML-Baum beliebig gewählt werden kann, sind damit nicht möglich. Hierfür definiert das W3C das Document Object Model, kurz DOM genannt.

Auch das .NET Framework implementiert das DOM. Abbildung 20 zeigt die in diesem Zusammenhang wichtigsten Klassen. Die Basis bildet die von der abstrakten Klasse `XmlNode` abgeleitete Klasse `XmlDocument`. Eine Instanz von `XmlDocument` verweist mit der Eigenschaft `DocumentElement` auf den Root-Knoten des Baums. Dieser wiederum ist eine Instanz von `XmlElement` und implementiert über zahlreiche Verzweigungen die Verknüpfungen der einzelnen Knoten.

Alle wesentlichen Klassen sind von der Klasse `XmlNode` abgeleitet. Sie stellt einen beliebigen Knoten im Sinne des DOMs dar. Auch Attribute, Textwerte oder das Dokument selbst sind Knoten. Ein `XmlElement`-Objekt ist das Pendant zum XML-Element im DOM. Es ist über zahlreiche Verweise mit den anderen Elementen eines XML-Baums verbunden (`ChildNodes`, `FirstChild`, `LastChild`, `ParentNode`, `NextSibling`, `PreviousSibling` usw.). Über diese Eigenschaften können Sie sich nach Belieben durch den Baum bewegen und jedes Element erreichen.

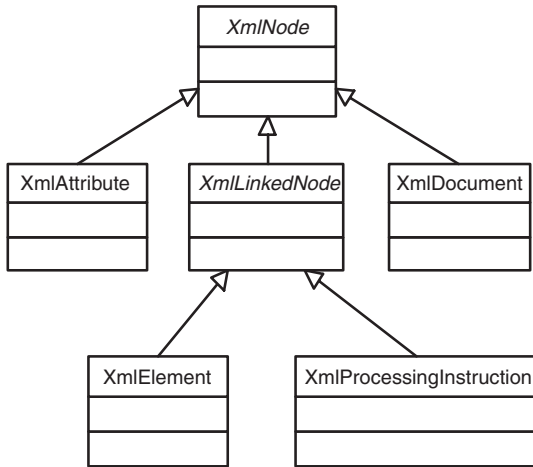


Abbildung 20: Die wichtigsten Klassen für den Umgang mit dem Document Object Model

`XmlDocument` stellt eine Reihe von Create-Methoden zur Verfügung, um untergeordnete Knoten (Elemente, Kommentare, Attribute, Textknoten usw.) anzulegen. Sie können beispielsweise mit `AppendChild`, `InsertAfter` oder `InsertBefore` der Baumstruktur hinzugefügt werden. Mit `Load` bzw. `Save` können die XML-Daten aus Dateien oder Streams geladen bzw. in ihnen gespeichert werden, mit `LoadXML` können sie aus einem String geladen werden.

`InnerXML` und `OuterXML` geben die Textdarstellung (Markup) des Knotens zurück. `OuterXML` schließt dabei den Knoten selbst mit ein, `InnerXML` nicht.

Die `Value`-Eigenschaft gibt den Wert eines Knotens zurück. Sie können sie allerdings nicht nutzen, um den Text eines `XmlElement`-Objektes zu ermitteln, denn der kann ja aus mehreren Knoten zusammengesetzt sein. `Value` ist gültig und sinnvoll für z.B. Kommentar-, Attribut-, Processing-Instruction-, `XmlDeclaration`- und natürlich Text-Knoten.

Mit `CreateNavigator` unterstützen alle von `XmlNode` abgeleiteten Klassen das Anlegen eines `XPath-Navigator`-Objektes für noch umfangreichere Navigationsmöglichkeiten und XPath-Abfragen.



# API-Funktionen

Die im Buch verwendeten API-Funktionen haben wir in der Klasse `API` untergebracht. Gängige Zahlenkonstanten, die üblicherweise mit `#define` in den C-Header-Dateien als Makro definiert werden, haben wir in Enumerationen gekapselt. Listing 5.1 zeigt die Klasse `API`.

---

```
Imports System.Runtime.InteropServices

Public Class API

    ' Window-Management
    Public Declare Function GetDesktopWindow Lib "user32.dll" () _
        As IntPtr

    Public Declare Function GetTopWindow Lib "user32.dll" _
        (ByVal hWnd As IntPtr) As IntPtr

    Public Declare Unicode Function GetWindowText Lib "user32.dll" _
        Alias "GetWindowTextW" (ByVal hWnd As IntPtr, _
        ByVal text As String, ByVal nMaxCount As Integer) As Integer

    Public Declare Function GetWindow Lib "user32.dll" ( _
        ByVal hWnd As IntPtr, ByVal uCmd As Integer) As IntPtr

    Public Declare Function GetForegroundWindow Lib "user32.dll" () _
        As IntPtr

    Public Declare Function SetForegroundWindow Lib "user32.dll" _
        (ByVal hWnd As IntPtr) As Boolean

    Public Declare Function GetWindowRect Lib "user32.dll" _
        (ByVal hWnd As IntPtr, ByRef lpRect As RECT) As Int32

    Public Declare Function ShowWindow Lib "user32.dll" _
        (ByVal hWnd As IntPtr, ByVal nCmdShow As Integer) As Boolean

    Public Declare Auto Function IsIconic Lib "user32.dll" _
        (ByVal hWnd As IntPtr) As Boolean

    Public Declare Auto Function IsZoomed Lib "user32.dll" _
        (ByVal hWnd As IntPtr) As Boolean

    Public Enum ShowWindowConstants
        SW_HIDE = 0
        SW_SHOWNORMAL = 1
        SW_NORMAL = 1
        SW_SHOWMINIMIZED = 2
```

---

Listing 9: Kapselung der im Buch verwendeten API-Funktionen in der Klasse `API`

---

```

SW_SHOWMAXIMIZED = 3
SW_MAXIMIZE = 3
SW_SHOWNOACTIVATE = 4
SW_SHOW = 5
SW_MINIMIZE = 6
SW_SHOWMINNOACTIVE = 7
SW_SHOWNA = 8
SW_RESTORE = 9
SW_SHOWDEFAULT = 10
SW_FORCEMINIMIZE = 11
SW_MAX = 11
End Enum

Public Declare Function GetWindowInfo Lib "user32.dll" _
    (ByVal hwnd As IntPtr, ByRef pwi As WINDOWINFO) As Boolean

Public Structure WINDOWINFO
    Public cbSize As Int32
    Public rcWindow As RECT
    Public rcClient As RECT
    Public dwStyle As Int32
    Public dwExStyle As Int32
    Public dwWindowStatus As Int32
    Public cxWindowBorders As Int32
    Public cyWindowBorders As Int32
End Structure

Public Declare Function ScreenToClient Lib "user32.dll" _
    (ByVal hWnd As IntPtr, ByRef lpPoint As POINTAPI) As Int32

Public Enum GetWindowConstants
    GW_HWNDFIRST = 0
    GW_HWNDLAST = 1
    GW_HWNDNEXT = 2
    GW_HWNDPREV = 3
    GW_OWNER = 4
    GW_CHILD = 5
End Enum

' Device Context
Public Declare Function GetDC Lib "user32.dll" _
    (ByVal hWnd As IntPtr) As IntPtr

Public Declare Function ReleaseDC Lib "user32.dll" _
    (ByVal hWnd As IntPtr, ByVal hdc As IntPtr) As Int32

' GDI-Funktionen
Public Declare Function StretchBlt Lib "gdi32.dll" _
    (ByVal hdc As IntPtr, ByVal x As Int32, ByVal y As Int32, _
    ByVal nWidth As Int32, ByVal nHeight As Int32, _
    ByVal hSrcDC As IntPtr, ByVal xSrc As Int32, _

```

---

---

```

    ByVal ySrc As Int32, ByVal nSrcWidth As Int32, _
    ByVal nSrcHeight As Int32, ByVal dwRop As Int32) As Int32

Public Declare Function BitBlt Lib "gdi32.dll" _
    (ByVal hdc As IntPtr, ByVal x As Int32, ByVal y As Int32, _
    ByVal nWidth As Int32, ByVal nHeight As Int32, _
    ByVal hSrcDC As IntPtr, ByVal xSrc As Int32, _
    ByVal ySrc As Int32, ByVal dwRop As Int32) As Int32

Public Declare Auto Function SelectObject Lib "gdi32.dll" _
    (ByVal hdc As IntPtr, ByVal hgdibj As IntPtr) As IntPtr

Public Declare Auto Function PathCompactPath Lib "shlwapi.dll" _
    (ByVal hdc As IntPtr, ByVal pszPath As _
    System.Text.StringBuilder, ByVal dx As Integer) As Boolean

Public Declare Function GetDeviceCaps Lib "gdi32.dll" _
    (ByVal hdc As IntPtr, ByVal nIndex As Integer) As Integer

Public Enum GetDeviceCapsConstants
    TECHNOLOGY = 2
    HORZSIZE = 4
    VERTSIZE = 6
    HORZRES = 8
    VERTRES = 10
    BITPIXEL = 12
End Enum

Public Structure POINTAPI
    Public x As Int32
    Public y As Int32
End Structure

Public Structure RECT
    Public Left As Int32
    Public Top As Int32
    Public Right As Int32
    Public Bottom As Int32
End Structure

Public Enum ROPConstants
    SRCCOPY = &HCC0020
End Enum

' Dateiinformationen wie Icons und Dateityp abfragen
Public Declare Auto Function SHGetFileInfo Lib "shell32.dll" ( _
    ByVal pszPath As String, _
    ByVal dwFileAttributes As Integer, _
    ByRef psfi As SHFILEINFO, _
    ByVal cbFileInfo As Integer, _

```

---

Listing 9: Kapselung der im Buch verwendeten API-Funktionen in der Klasse API (Forts.)

---

```

ByVal uFlags As Integer) _
As IntPtr

' Datenstruktur SHFILEINFO für den Aufruf von SHGetFileInfo
<StructLayout(LayoutKind.Sequential, CharSet:=CharSet.Auto)> _
Public Structure SHFILEINFO
    Public hIcon As Int32
    Public iIcon As Int32
    Public dwAttributes As Int32
    <MarshalAs(UnmanagedType.ByValTStr, SizeConst:=260)> _
    Public szDisplayName As String
    <MarshalAs(UnmanagedType.ByValTStr, SizeConst:=80)> _
    Public szTypeName As String
End Structure

' Gebräuchliche Konstanten für SHGetFileInfo
Public Enum SHGetFileInfoConstants
    SHGFI_TYPENAME = &H400
    SHGFI_ATTRIBUTES = &H800
    SHGFI_EXETYPE = &H2000
    SHGFI_LARGEICON = 0
    SHGFI_SMALLICON = 1
    SHGFI_ICON = &H100
End Enum

' Benötigte Methode zur Freigabe eines Icon-Handles
Public Declare Auto Function DestroyIcon _
    Lib "user32.dll" (ByVal hicon As IntPtr) As Boolean

'SendMessage
Public Declare Function SendMessage Lib "user32.dll" Alias _
    "SendMessageW" (ByVal hwnd As IntPtr, _
    ByVal message As Integer, ByVal wParam As IntPtr, _
    ByVal lParam As IntPtr) As Integer

Public Declare Function SendMessage Lib "user32.dll" Alias _
    "SendMessageW" (ByVal hwnd As IntPtr, _
    ByVal message As Integer, ByVal wParam As Int32, _
    ByVal lParam As Int32) As Integer

Public Declare Function SendMessageRef Lib "user32.dll" Alias _
    "SendMessageW" (ByVal hwnd As IntPtr, _
    ByVal message As Integer, ByVal wParam As Int32, _
    ByRef lParam As Int32) As Integer

' Konstanten für Multiline-TextBox
Public Enum TBMultiline
    EM_GETLINECOUNT = &HBA
    EM_LINEFROMCHAR = &HC9
    EM_FMTLINES = &HC8

```

---



---

```

    EM_LINESCROLL = &HB6
    EM_GETMODIFY = &HB8
    EM_SETMODIFY = &HB9
    EM_LINEINDEX = &HBB
    EM_GETTHUMB = &HBE
    EM_SETTABSTOPS = &HCB
    EM_GETFIRSTVISIBLELINE = &HCE
    EM_POSFROMCHAR = &HD6
    EM_CHARFROMPOS = &HD7
End Enum

' SHFileOperation für Dateioperationen mit Windows-Shell
Public Declare Auto Function SHFileOperation Lib "shell32.dll" _
    (ByRef lpFileOp As SHFILEOPSTRUCT) As Integer

' Unterstützte Kommandos für SHFileOperation
Public Enum SHFileOpConstants
    Move = 1
    Copy = 2
    Delete = 3
    Rename = 4
End Enum

' Struktur für SHFileOperation
<StructLayout(LayoutKind.Sequential, Pack:=1, _
    CharSet:=CharSet.Auto)> _
Public Structure SHFILEOPSTRUCT
    Public hwnd As IntPtr
    Public wFunc As SHFileOpConstants
    Public pFrom As String
    Public pTo As String
    Public fFlags As SHFileOpFlagConstants
    Public fAnyOperationsAborted As Boolean
    Public hNameMappings As IntPtr
    Public lpszProgressTitle As String
End Structure

' Flag-Definitionen für SHFileOperation
<Flags(> Public Enum SHFileOpFlagConstants As Integer
    Multitdestfiles = 1
    Confirmmouse = 2
    Silent = 4
    RenameOnCollision = &H8
    NoConfirmation = &H10
    WantMappingHandle = &H20
    AllowUndo = &H40
    FilesOnly = &H80
    SimpleProgress = &H100
    NoConfirmMakeDir = &H200
    NoErrorUI = &H400
    NoCopySecurityAttribs = &H800

```

---

---

End Enum

```

' Verzeichnis-Funktionen
Public Declare Auto Function PathRelativePathTo _
Lib "shlwapi.dll" (ByVal relPath As System.Text.StringBuilder, _
ByVal pathFrom As String, ByVal attrFrom As Integer, _
ByVal pathTo As String, ByVal attrTo As Integer) As Boolean

Public Declare Auto Function PathCanonicalize Lib "shlwapi.dll" _
(ByVal dst As System.Text.StringBuilder, ByVal src As String) _
As Boolean

Public Enum FileAttribute
[ReadOnly] = 1
Hidden = 2
System = 4
Directory = &H10
Archive = &H20
Normal = &H80
Temporary = &H100
End Enum

' Sound-Ausgabe
Public Declare Auto Function sndPlaySound Lib "Winmm.dll" _
(ByVal path As String, ByVal fuSound As SNDPlaySoundConstants) _
As Boolean

<Flags(> Public Enum SNDPlaySoundConstants
Async = 1
[Loop] = 8
Memory = 4
NoDefault = 2
NoStop = &H10
Sync = 0
End Enum

' Abfragen von Zählerstand und Frequenz des Hardware-Counters
Public Declare Auto Function QueryPerformanceCounter Lib _
"Kernel32.dll" (ByRef performanceCount As Long) As Boolean

Public Declare Auto Function QueryPerformanceFrequency Lib _
"Kernel32.dll" (ByRef frequency As Long) As Boolean

' Fehlermeldungen
Public Declare Auto Function FormatMessage Lib "kernel32.dll" _
(ByVal dwFlags As Integer, ByVal lpSource As IntPtr, _
ByVal dwMessageId As Integer, ByVal dwLanguageId As Integer, _
ByVal lpBuffer As System.Text.StringBuilder, _
ByVal nSize As Integer, ByVal Arguments() As String) As Integer

End Class

```

---

# Stichwortverzeichnis

!

( 748

<< 49

>> 49

@@IDENTITY 610

\_\_CLASS 517

\_\_namespace 516

3D-Effekte 154

## A

Abfrage, WQL 517

AcceptButton 242

Achsenbeschriftung 723

adaptiven Schärfe 227

Add, ListBox 297

AddDays 74

AddExtensionObject 679

AddHandler 294, 477

AddMessageFilter 317

Advent 87

AlarmTimeUIEditor 403

AllowDrop 269

AllowedEffect 270

Alpha-Wert 313

And 50

Animierte Dialogfenster 490

Anmeldung

NT-Security (SQL-Server) 587

SQL-Server Verfahren 587

Anwendung 103

mehrfachen Start verhindern 121

Anwendungskonfigurationsdatei 103

API-Fehlermeldungen 773

API-Funktionen 825

ApiVBNet 773

App.config 104

AppendChild 637

Application, EnableVisualStyles 282

Application.Run 124

Application.ThreadException 124

ApplicationData 110, 113

appSettings 105

ARGB 313

Array 789

Control 293

Größe 792

mehrdimensional 793

Parameter-, von Funktionen 556

Werte zuweisen 790

Array.Sort 64

ArrayList 67

Ascend 148

Ascent 144

ASCII 69

Assembly-Datei 458

AssemblyDescription 460

AssignHandle 315f.

Auflösung 530

Monitor 529

AutoScroll 305

AutoScrollMinSize 305

Auto-Vervollständigen, ComboBox 303

## B

BackColor 313

Backup, siehe Datensicherung 597

Backup-Set 592

Bandgeräte 592

Base64 629, 632

Basisklasse 784

BeginInvoke 237

Benutzerdefinierte Steuerelemente 351

Benutzereingaben, Validierung 249

Benutzersteuerelement 302, 305, 309, 320

Betriebssystem

Informationen zum 570

Name des 572

Bild vergrößern 162

Bilddatei, Blockierung verhindern 191

Bilddateien, Formate 207

Bilder

Drehen und spiegeln 205

GIF-Format 209

in Graustufen wandeln 213

invertieren 212

JPEG-Format 208

maximieren 186

zeichnen 159

Bildladezeiten 193

Bildmanipulationen 215

Binäre Suche 67

BinarySearch 67

BinHex 629

BIOS 518, 520

BiosChar 520  
 Bit  
   löschen 51  
   prüfen 49  
   setzen 50  
   umschalten 52  
   Wert zuweisen 51  
 BitBlt 257  
 BitConverter 56  
 Bitmap 191  
   aus Array erzeugen 217  
   in Array kopieren 216  
 BitmapData 216  
 Blockschrift 138  
 BoldedDates 93  
 Boolean-Array, in Integer wandeln 53  
 Boot-Status 524  
 Border3DStyle 194, 309  
 Boxing 779  
 Browsable-Attribut 305  
 BrowsableAttribute 386, 428  
 ByRef 780  
 ByVal 780

## C

CancelButton 242  
 CancelEventArgs 250  
 CanConvertFrom 395  
 CanConvertTo 395  
 CanPauseAndContinue 558  
 CanStop 560  
 CategoryAttribute 386, 428  
 CausesValidation 251  
 CChar 34  
 CDATA 629  
 channel, RSS 657  
 Char, als Literal 33  
 CheckBox, auf TreeView 341  
 CheckState 345  
 chiseled, Shift 152  
 CIM 819  
 CIMOM 819  
 Clipboard 456  
 Clipboard 277  
 Clipping 134, 232  
 Clipping-Bereich 155  
 Clipping-Region 171  
 Clip-Region 182  
 CloseFigure 150  
 CLSCompliant 803  
 CLS-Kompatibilität 803  
 CollectionEditor 417  
 Color.Transparent 351

ColorMatrix 184, 212  
 Colormatrix 215  
 ComboBox 299, 303  
   Auto-Vervollständigen 303  
   mehrspaltig 299  
 CommandBuilder 614  
 CommonApplicationData 110  
 CommonProgramFiles 110  
 Compare 334  
 CompareTo 64, 336  
 Component 302, 361  
 config 103  
 Config-Datei 755  
 configSections 105, 107  
 ConfigurationSettings 105  
 Container 304  
 Container-Control 306  
 ContainerControl 302, 304  
 Contro 302  
 ControlAdded 295  
 Control-Arrays 293  
 ControlDesigner 357  
 ControlPaint 193, 309  
 Controls 293  
   Auflistung 342  
 Convert, Klasse 35  
 Convert.FromBase64String 632  
 Convert.ToBase64String 631  
 Convert.ToString 35  
 CounterCreationData 767  
 CounterCreationDataCollection 767  
 Create 445  
 CreateAndShow 241  
 CreateDocument, Fax 580  
 CreateGraphics 170  
 CreateNavigator 647  
 CreateSubKey 569  
 CreateTextNode 636  
 CreateThumbnailfiles 210  
 CShort 34  
 CSng 34  
 CSV-Datei 622  
 CultureInfo 39, 41  
 CurrentCulture 40  
 CurrentDirectory 483

## D

Database Management Objects 583  
 DataGrid 616  
 DataRow 683  
 DataSet 681  
 Dataset 473, 610  
 DataSource 616

- Date 73
- Datei
  - Attribute 478
  - Delimited 472
  - Existenz prüfen 453
  - Größe 460
  - Icon 486
  - Informationen 468
  - kopieren 457
  - suchen 466
  - überwachen 475
  - umbenennen 456
  - verschieben 456
  - Version 458
- Dateien
  - temporäre Namen 464
  - vergleichen 462
- Dateiendung, Programm starten über 543
- Dateiname 482
  - laufender Prozess 442
- Dateinamen, 8.3 Format 454
- Dateioperationen 490
- Dateipfad, kürzen 286
- Dateisystem, Überwachung 475
- Dateityp 486
- Datenbank
  - erstellen 601
  - Tabelle 589
  - zur Bearbeitung festlegen 589
- Datenquelle 722
- Datensicherung
  - Bandgerät 592
  - Datenbank, einfache 590
  - Datenbank, zurückspielen 591
  - Device, SQL-Server 594
  - Einspielen 597
  - Geräteliste 596
- DateTime 73
- DateTime.Now 771
- DateTime.Parse 100
- DateTimeFormat 42
- DateTimeFormatInfo 43
- Datum 73
  - auf Feiertag überprüfen 92
  - Gregorianisch 95
  - im ISO-Format 99
  - Julianisch 95
- DayOfWeek 76
- Debug.WriteLine 755
- Debuggen, im Entwurfsmodus 807
- Decimal, als Literal 33
- DefaultProperty 391
- DefaultValue 390
- Delegate 243
- Descend 148
- Descent 144
- Description-Attribut 386
- DescriptionAttribute 428
- Designer 303, 305
- DesignerAttribute 304
- DesignerSerializationVisibility 306, 361
- DesignerVerb 421
- DesignerVerbCollection 421
- DestroyIcon 487
- DeviceID 536
- Diagramm 723, 727
  - kontinuierliche Darstellung 742
- Diagramme 721
- Dialoge 238
  - kapseln 238
- DialogResult 242
  - Cancel 242
  - OK 242
- Dia-Show 166
- DictionaryBase 82
- DictionarySectionHandler 106
- Dienst
  - anhalten 557
  - fortsetzen 559
  - starten 554
  - Startparameter 556
  - stoppen 560
- Dienste 553
  - Gerätetreiber 554
- Differenzsicherung 595
- DIN 1355 77
- DirectoryConfigFile 116
- DirectoryInfo 444, 446
- DirectX 753, 814
  - Sound 753
- DMO 583
- DMTF 522, 819
- Dns, .NET Klasse 505
- Document – View 238
- Document Object Model 633
- DoDragDrop 274, 281
- Dokumentgliederung 809
- DOM 633
- Domäne 525
  - Rolle des PCs 526
- Double 697
- DoubleComparer 697
- Drag & Drop 265, 269, 271
- DragEnter 270
- DragEventArgs 270
- DrawBackground 298
- DrawBorder3D 193f.
- DrawImage 159, 212

DrawItem 296, 301  
DrawItemEventArgs 298  
DrawMaximizedPicture 188  
DrawMode 296, 301  
DrawPath 127  
DropDown-Liste, mehrspaltig 299  
DropDownStyle 303  
DTD 668  
Dualzahlen 35

## E

Editor 363  
Editor-Attribut 400  
EditorAttribute 407  
Effect - Eigenschaft 270  
Eigenschaften 783  
Eigenschaftsfenster 383  
Eingabetaste abfangen 284  
eMail 578  
embossed, Schrift 152  
EN 28601 77  
EnableNotifyMessage 314  
EnableVisualStyles 282  
Encoder 212  
    Bilddateien 207  
    GIF 209  
    JPEG 208  
    Text 69  
Encoding, XML-Dateien 621  
EntryWritten-Ereignis 765  
Entwurfsmodus 305  
Entwurfszeit 303  
    Maus-Ereignisse abfangen 356  
Enum  
    Basistyp 59  
    Ein-/Auslesen 60  
    Werte und Bezeichner abfragen 62  
Enum.IsDefined 63  
EnumerationOptions 517  
Environment 570  
Ereignisanzeige 764  
ErrorProvider 251  
Erweiterung, Dateiname 482  
Evaluate 662  
    XPath 650  
Eventlog, überwachen 765  
EventLogEntryType 763  
Eventlogs, eigene definieren 762  
EventLogTraceListener 757, 761  
Excel 617  
Exception-Handling, zentral 124  
Exceptions 124  
Exclude 155

ExcludeClip 181  
ExpandableObjectConverter 395  
Explorer  
    Server 532  
    Steuerelement 330  
Exponent 697  
Extension 482

## F

Fading 168  
Farbmatrix 213  
Farbvektor 212  
Farbwerte 212  
Fax, -Queue 581  
Fax. 580  
faxcom.dll 580  
Feiertag, abfragen 89  
Feiertage 80f.  
    bewegliche 85  
    feste 84  
Fenster  
    halbtransparent 231  
    ohne Titelleiste 229  
    transparent 231  
    verschieben 229, 234  
File 457, 467  
File System Objects 449  
FileDrop 275, 279  
FileInformation 331, 334  
FilenameEditorAttribute 410  
FileStream 462f., 614  
FileSystemObject 456, 478  
FileSystemWatcher 475  
FileVersionInfo 460  
FillPath 127  
Filter 222  
Filterfunktion 220  
Filterung 315  
FindString 303  
Flächenmaße 708  
Flags 393  
    Attribut 61  
FlatStyle 282  
FolderBrowser 192  
FolderBrowserDialog 319, 453  
FolderBrowser-Steuerelement 341  
Font-Metrics 144  
Fontmetrics 148  
Format  
    General 35  
    länderspezifisch 39  
    positive und negative Werte 36  
Format8bppIndexed 216

Formatbezeichner 36  
FormatMessage 773  
Format-Provider 38  
Fortschrittsanzeige 236  
Friend 787  
FSO 449, 468  
Funktionswert 721

## G

Gauss, Schärfefilter 222  
Geräte, Datensicherung 596  
Gerätetreiber, Dienste 554  
Geschwindigkeit 710  
GetAttribute, XML 626  
GetBytes 69  
GetCellAscent 144  
GetCellDescent 144  
GetCommandLineArgs 544  
GetConfig 106  
GetCurrentProcess 123  
GetData 271  
GetDataObject 277  
GetDataPresent 271  
GetDesktopWindow 259  
GetDevices 554  
GetDirectories 324  
GetDirectoryName 482  
GetEnvironmentVariable 466  
GetEventLogs 765  
GetExecutingAssembly 442  
GetFile 456  
GetFileName 482  
GetFiles 451  
GetFolderPath 440  
GetFormats 271  
GetFullPath 483  
GetHdc 287  
GetHeight 144  
GetHostByName 505  
GetHostName 505  
GetImageEncoders 207  
GetLastWin32Error 773  
GetLineSpacing 144  
GetLogicalDrives 471  
GetNames, Enum 62  
GetPaintValueSupported 400  
GetPixel 216  
GetProcesses 548f.  
GetProcessesByName 123  
GetProperties 426  
GetResponse 263  
GetResponseStream 263  
GetStandardValues 399

GetString 69  
GetTempFileName 464  
GetTempPath 464  
GetTopWindow 256  
GetValues, Enum 62  
GetWeekOfYear 76  
GetXml 473  
GIF-Format 209  
GiveFeedback 274, 281  
Gleitkommazahlen, vergleichen 697  
Gliederungsansicht 809  
Global, Unique Identifier 465  
Google 816  
GraphicsPath 127, 134, 155, 177  
Graustufen 213  
GregorianCalendar 73  
Gregorianischer Kalender 73  
GUID 465

## H

Handle, Windows- 549  
HandleCreated 316  
HandleDestroyed 316  
Handles 293  
Hardware-Counter 771  
HasAttributes, XML 626  
Hashtable 800  
HasValue, XML 626  
Heap 777  
Height, FontFamily 144  
Helligkeit 215  
Helligkeitsverlauf 154  
Hexadecimal 34  
Hexadezimal, nach Dezimal umrechnen 504  
Hexadezimalzahlen 35  
HighDoubleWord 56  
HighWord 56  
Hintergrund 351  
Hintergrundfarbe, transparent 313  
HitTest 93  
HTML 664, 674

## I

IComparable 64, 81, 336  
IComparer 64, 334  
Icon  
    Benutzersteuerelement 312  
    einer Datei 331, 486  
IConfigurationSectionHandler 108  
ICustomTypeDescriptor 424  
IDataObject 271  
Identitätsmatrix 148  
IdictionaryEnumerator 453

IFormattable 38  
ImageAttributes 184  
ImageCodecInfo 207  
ImageEncoders 207  
IMessageFilter 284, 317  
Imports 806  
IndentLevel 757  
IndentSize 757  
InitializeComponent 294, 361, 390  
InsertAfter 637  
InsertBefore 637  
Instanzmethoden 780  
Integer  
    gesetzte Bits abfragen 54  
    in BooleanArray wandeln 52  
    nicht gesetzte Bits abfragen 55  
Interface 788  
Internetquellen 813  
IntPtr 550  
IP-Adresse 502, 504  
IPv4 501  
IPv6 501  
IsDefined, Enum 63  
ISerializable 693  
IsIconic 123  
IsLeapDay 75  
IsLeapMonth 75  
IsLeapYear 75  
ISO 8601 77, 99  
ISO-Format 99

## J

Jagged-Array 796  
JPEG-Format 208  
Julianische Tageszählung 95

## K

Kalender 73  
Kalender-Steuerelement, Feiertage 93  
Kalenderwoche 78  
    Beginn 76, 79  
Kalenderwochen, Anzahl 77  
Kategorien, Eigenschaftsfenster 386  
kaufmännisch runden 48  
KeyPreview 284  
KeyUp 303  
Kirchenjahr 87  
Klasse 777  
    abstrakte 787  
Klassenansicht 811  
Kodierung, Strings 69  
Komprimierung 209  
Konfiguration, globale Daten 104

Konfigurationsdatei 103, 755  
Konfigurationsdateien, Anwenderdaten 110  
Konfigurationsmanager 116  
Konstruktor 303, 778  
Konvertierung, String in Byte-Array 69  
Konvertierungen, zwischen Byte, Word, DWord und Long 56  
Koordinatensystem 353  
Kopieren, von Dateien 490  
Kurvendiagramm 727

## L

Ländereinstellungen 40  
Länge 700  
Längengrad 73  
Längenmaße 706  
Largelcon 333, 486  
LastSaved 119  
LastStarted 119  
Laufwerk 531  
    Informationen 470  
Laufwerke 320  
    freigegebene 509  
Leerlaufprozess 553  
Leistung 769  
Leistungsindikatoren, anlegen 766  
Lichtquelle 154  
LinearGradientBrush 130, 142, 155  
LinearMeasures 706  
LineSpacing 144  
Linienschreiber 741  
LinkClicked 513  
LinkLabel 667  
ListBox 296  
    Einträge selber zeichnen 296  
Liste, typisiert 415  
listeners, Konfigurationsdatei 761  
ListView 313, 330, 650  
    Sortieren 334  
ListViewItem 334  
ListViewItemCollection 334  
ListViewItemSorter 338  
Literale 33  
Load 294  
LoadPictureThread 196  
LocalApplicationData 110  
LocalPathConfigFile 116  
LockBits 216  
Löschen, von Dateien 490  
Loginsecure 586  
Lokalzeit 100  
Long, als Literal 33  
Loop 751



LowDoubleWord 56

LowWord 56

## M

MAC-Adresse 507

MailMessage 579

ManagementClass 516

ManagementObjectSearcher 517, 520, 575

ManagementPath 516

ManagementScope 516, 575

Mantisse 697

Marshal.ReadByte 216

MarshalByRefObject 693

Masse 700

Maßeinheiten

  Typsicherheit 699

  umrechnen 716

Maßstab 723

Math, Klasse 353

Math.Round 48

Matrix 212, 215

  invertiert 148

Matrix.Invert 149

Matrix.Scale 148

Matrix.Shear 148

Matrix-Operationen 147

Matrizen-Multiplikation 148

MaxValue 44

MDAC 616

Me 781

MeasureItem 296

MeasureItemEventArgs 297

MeasurementAttribute 704

MeasurementBase 701

MeasureString 298

Menü, mit gekürzten Pfadangaben 289

Messwert 723

Methode 780

  abstrakte 787

  statische 781

  überladen 780

  überschreiben 785

mgmtclassgen 522, 822

Mgmtclassgen.exe 820

Miniaturen 210

Miniaturenansicht 191

MinValue 44

Module 782

Monitor, Auflösung 530

MonthCalendar 79, 93

Mousewheel 199

MSDE 583

MSDN 813

MSI 539

mStride 218

Multiline 373

MultiplyTransform 148

MustInherit 303, 787

MustOverride 787

Mutex 121

## N

Nachrichten 664

  Windows 313

Nachrichtenschleife 314

Namensraum, XML 643

Namespace, XML 643

NamespaceURI 644

NativeWindow 314f., 344

New 778

NewGuid 465

Newsfeeder 657, 664

Newsgroups 814

NextSample, Leistungsindikatoren 564

NextValue, Leistungsindikatoren 564

NodeType, XML 626

Now 74

NT-Security 585

NumberFormat 42

NumberFormatInfo 43

## O

Objekt 777

Oktal 34

Oktalzahlen 35

OLE 269

OleDbConnection 617

OleDbDataAdapter 617

OnHandleCreated 285

OnHandleDestroyed 285

OnNotifyMessage 314

OnPaint 160

OnPaintBackground 310

Opacity 231

OpenSubKey 567

OperatingSystem 571

Option Compare 805

Option Explicit 805

Option Strict 805

Ordnerauswahl, mit Miniaturbildern 191

Ortszeit 99

Osterdatum 80, 85

Outline 138

Outline-Schrift 127

Overridable 785

OwnerDraw 296

**P**

Paint-Ereignis 160  
 Papierkorb 490  
 ParamArray 556  
 Parameter  
     optionale 780  
     Übergabe, Programmzeile 544  
 Parameterarray 556  
 ParameterDirection.Output 610  
 ParentControlDesigner 304  
 Parse 74  
     DateTime 100  
 Partition 536  
 Path 482  
     CloseFigure 150  
 PathCanonicalize 483  
 PathCompactPath 286  
 PathPoints 128  
 PathRelativePathTo 484  
 PathTypes 128  
 Paused 558  
 Performance Counter 563  
 PerformanceCounter, anlegen 766  
 PerformanceCounterCategory 767  
 Personal 110  
 Perspektivisch verzerren 142  
 Pfad  
     absolut 483  
     Bestandteile 482  
     kanonisch 483  
     relativ 483f.  
 Pfade, kürzen 286  
 Physikalische Größen 700  
 Physikalisch-Technische Bundesanstalt 701  
 Pi 748  
 Pinguin 692  
 PlaySound 751  
 Positionierung von Schriftzügen 144  
 PreFilterMessage 285, 317  
 PrependChild 637  
 Primary Key 610  
 Private 786  
 Process 549  
 ProcessModul 549  
 ProcessStartInfo 547, 549  
 ProcessThread 549  
 Programm, über Dateiendung starten 543  
 Programm, externes, starten 542  
 Programme, installierte 539  
 Projekteinstellungen 805f., 809  
 Property 783  
 PropertyDescriptorCollection 426, 437  
 PropertyGrid 42, 383

Aktionen 419  
 Arrays und Listen 411  
 Auflistungen 411  
 BrowsableAttribute 386  
 CategoryAttribute 386  
 Datei-Dialog 410  
 DefaultProperty 391  
 DescriptionAttribute 386  
 DropDown-Editor 401  
 DropDown-Liste für Werte 397  
 Eigenschaften dynamisch erstellen 423  
 Enumerations 392  
 geschachtelte Eigenschaften 393  
 Gültigkeitsprüfung 389  
 Hyperlink-Tasten 419  
 mehrere Objekte gleichzeitig 388  
 Miniaturbilder 399  
 modaler Dialog 405  
 schreibgeschützte Eigenschaften 391  
 Sprache lokalisieren 431  
 Standardwerte 390  
 Tab-Flächen 434  
 TypeConverter 395  
 Verbs 419  
 PropertyTabAttribut 434  
 Protected 786  
 Protected Friend 787  
 Prozess 121  
     abbrechen 561  
     starten 542  
 Prozesse 546, 549  
 Prozessor  
     allg. Informationen 578  
     Auslastung 577  
     Bitbreite 577  
     Geschwindigkeit 575  
 PTB 701  
 Public 786

**Q**

QueryPerformanceCounter 771  
 QueryPerformanceFrequency 771

**R**

RawValue, Leistungsindikatoren 564  
 RDF 657, 664  
 ReadElementString, XML 626  
 ReadOnlyAttribute 392  
 ReadString, XML 626  
 ReadXml 682  
 ReadXmlSchema 682  
 Rechtecke, Ecken runden 150  
 Referenz 777

Region.Complement 181  
Registry 566, 616  
    GetValue 616  
    Key anlegen 568  
    Key löschen 569  
ReleaseHandle 316  
remove, Konfigurationsdatei 761  
RemoveMessageFilter 317  
ResizeRedraw 306  
Resource Description Framework 657  
Resource Kit 813  
ResourceManager 431  
Ressource-Datei 431  
Restore, siehe Datensicherung 597  
RichTextBox 513  
root 517  
roperityDescriptor 426  
RotateFlip 205  
RotateFlipType 205  
RotateTransform 205  
RoundedRectangle 150  
RSS 657, 664  
RTF-Zeichenkette 512  
RtlMoveMemory 219  
Rücksicherung, siehe Datensicherung 597  
Runden 48  
Rundungsfehler 697  
RunOnce 567  
RunOnceEx 567

## S

Sättigung 215  
ScaleInfo 724  
ScaleTransform 145  
Schärfefilter 219  
    Boxcar 226  
    Gauss 222  
    Laplace 226  
    Sobel 225  
Schärfenberechnung 227  
Schaltfläche  
    Abbrechen 245  
    OK 245  
    Übernehmen 243, 245  
Schaltjahr 73, 75  
Schatten  
    für Grafiken 140  
    verzerren 147  
Schema, XML 668  
Scherung, mit Matrix-Operation 148  
Schnittstelle 788  
    IComparable 336  
    IComparer 334

Schrift  
    hervorgehoben 152  
    mit Hintergrundbild 133  
    vertieft 152  
Schwarz / Weiß-Bilder 213  
ScreenHeight 529  
Screenshot 254  
ScreenToClient 257  
ScreenWidth 529  
Scripting Runtime 468  
Scripting-Engine 448  
ScrollableControl 199, 302  
ScrollableControlDesigner 305  
Scroll-Balken, im Entwurfsmodus 305  
scrnun.dll 449  
Seife 692  
SelectAncestors 649  
SelectChildren 649  
SelectDescendants 649  
SelectedObjects 388  
SelectNodes 639, 645  
SelectObject 287  
SelectSingleNode 639, 645  
sender 295  
SendKeys 285  
SendMessage 372  
Serialisieren  
    SoapFormatter 692  
    XML 685  
SerializableAttribute 693  
Server, SMTP 578  
Server-Explorer 532  
ServiceController 553  
ServiceControllerStatus 557  
ServiceProcess, System. 553  
ServiceProcess.DLL 553  
SetAttribute, XML 637  
SetClip 155, 176  
SetColorMatrix 212  
SetDataObject 277, 456  
SetError 251  
SetForegroundWindow 123, 257  
SetPixel 216  
SetStyle 306, 351  
Shared 781  
Shear 148  
SHFileOperation 490  
SHFILEOPSTRUCT 490  
SHGetFileInfo 486  
Short, als Literal 33  
ShortPath 456  
ShowDialog 242  
ShowWindow 123, 257  
SI-Einheit 700

Single 697  
  als Literal 33  
SingleTagSectionHandler 105  
SizeMode 159  
Skalenwerte 723  
Skalierung, berechnen 723  
Skalierung der Y-Achse 737  
SmallIcon 486  
SMTP 578  
SmtMail 579  
SmtServer 579  
sndPlaySound 751  
SoapFormatter 691f.  
Sommerzeit 73, 100, 525  
Sonne 73  
Sort, ListView 338  
SortedList 801  
sortieren, ListView 334  
Sortieren von Objekten 64  
Sound, Ausgabe 751  
SpecialFolder 114, 117, 290, 440  
SpeedMeasures 710  
SplashScreen 235  
SqlDataAdapter 610  
SQL-Server  
  Anmeldung 585  
  erreichbare 584  
SQL-Server 583  
Stack 780  
Standardwerte 390  
Standardzeit 99  
Startbildschirm 234  
Startoptionen 807  
Steuerelement 302  
  für Zeitbereiche 358  
Steuerelemente  
  auf einem TreeView 342  
  kreisförmig 353  
  nicht rechteckig 351  
Stile, Windows XP 282  
StopSound 752  
StreamReader 473  
String.Format 37  
StringAlignment 186  
String-Arrays, konstante 71  
StringConverter 398  
StringDictionary 802  
StringFormat 134  
Stromstärke 700  
Struktur 779  
Stylesheet, XML 674  
Subclassing 293, 315  
Suche, binär 67  
SupportsTransparentBackColor 351

SurfaceMeasures 708  
switches, Konfigurationsdatei 755  
System.ComponentModel.Component 302  
System.DateTime 73  
system.diagnostics, Konfigurationsdatei 758  
System.Environment 439  
System.Reflection 442  
SystemMonitor 769  
Systemprozesse 546, 549

## T

T/Y-Diagramm 741  
T/Y-Plot 727  
Tabellen 721  
Tabellenfelder 590  
Taktfrequenz, Prozessor 576  
Temperatur 700  
TemperatureMeasures 713  
Temperaturwerte 713  
Text, kreisförmig anordnen 128  
TextBox  
  Anzahl Zeilen ermitteln 375  
  Eingabe abfangen 284  
  erste sichtbare Zeile 377  
  erstes Zeichen einer Zeile 377  
  Koordinate eines Zeichens 379  
  mehrzeilig 373  
  scrollen 380  
  Tabulatorpositionen 381  
  Zeichen an Koordinate 378  
  Zeilen ermitteln 373  
  Zeilenindex ermitteln 376  
Texte umbrechen 47  
Text-Eigenschaft 305  
Textknoten 636  
TextureBrush 135  
TextWriterTraceListener 757, 759  
this 781  
Thread 193, 235, 317  
  Haupt-, eines Prozesses 562  
ThreadException 124  
ThrowAPIError 774  
Thumbnail 193  
Thumbnails 210  
ThumbNailShow 196  
TimeEditorControl 402f.  
TimeMeasures 712  
TimeSpan 73  
Titelleiste 229  
ToDateTime 575  
Today 74  
ToHfont 287  
Toolbox, Texte speichern 805

ToString, DateTime 75  
TotalDays 74  
TotalHours 74  
TotalMinutes 74  
TotalVisibleMemorySize 575  
ToUniversalTime 101  
Trace.WriteLine 755  
Trace-Ausgaben  
    in Datei 759  
    in Eventlog 761  
    in TextBox umleiten 641, 757  
Trace-Level 756  
TraceListener 757, 759, 761  
TraceSwitch 756  
TraceToTextBox 757  
TranslateClip 134  
TranslateTransform 145  
Transparenz 136, 140, 184, 212, 306, 313  
Transparenzeffekte 232  
TreeView 313, 320, 330, 650  
    mit CheckBoxen 341  
Trojaner 567  
Typ, einer Datei 331, 486  
TypeConverter 395, 398  
Typisierten Liste 415  
Typsicherheit, Maßeinheiten 699

## U

Überblenden  
    Diagonal 177  
    durch Mosaik 181  
    Elliptisch 179  
    horizontal und vertikal 171  
    mit Transparenz 183  
Überblendvorgang 166, 186  
Übergabeparameter 544  
Überladen, Methoden 780  
Übernehmen 243  
Überwachung, Leistungs- 563  
Uhrzeit 73  
    im ISO-Format 99  
UICulture 434  
UIType-Editor 417  
UITypeEditor 363, 400, 403, 407, 652  
UITypeEditorEditStyle 403, 409  
Umbenennen, von Dateien 490  
Umrechnen, Physikalisch Technischer  
    Größen 716  
Unboxing 779  
UniformResourceLocator 275  
uniqueidentifier 611  
UnknownAttribute 691

UnknownElement 691  
UnknownNode 691  
UserControl 302, 304  
UserMouse 199  
UseShellExecute 543f.  
UTC 99

## V

Validated 250  
Validating 250, 252  
Validierung, von Benutzereingaben 249  
Verbs 421  
Vererbung 784  
Vergleichen von Objekten 64  
Vergrößern, Bildausschnitt 162  
Verknüpfung, Projektdatei 809  
Verlauf 128  
Verschieben, von Dateien 490  
Version 616  
Verzeichnis  
    erstellen 445  
    Existenz prüfen 443  
    Größe 450  
    kopieren 448  
    löschen 446  
    überwachen 475  
    umbenennen 446  
    verschieben 446  
Verzeichnisauswahl 319  
Verzeichnispfad, kürzen 286  
Verzeichnisse, System 439  
Verzerrungen, mit Warp 142  
Viewer  
    für Bilder 263  
    für formatierten Text 261  
    für HTML – Texte 266  
    für Text 259  
VisibleClipBounds 188  
Vorkommastellen, signifikante 45

## W

W3C 621, 657  
WaitForStatus 557, 559  
Warp 142  
Wave-Datei 753  
Wbem 819  
Web 657  
WebBrowser – Steuerelement 266  
WebBrowser-Steuerelement 662  
WebRequest 263  
Web-Service 501, 510  
Weihnachtstag 87  
Wertebereich 44

Wertetyp 779  
Win32\_Bios 518  
Win32\_ComputerSystem 526  
Win32\_Computersystem 527  
Win32\_NetworkAdapter 507  
Win32\_Product 539  
Win32\_Share 509  
WinDiff.exe 462  
WINDIR 440  
Windows, Version 573  
Windows XP 282  
Windows-Nachrichten 313  
Winterzeit 100, 525  
WM\_KEYDOWN 285  
WM\_PAINT 345  
WMI 515  
    Klassenerstellung in .NET 820  
    Namensräume 515  
    VS Plug-In 532  
WMI-Erweiterung 532  
WndProc 345  
WndProc-Methode 314f.  
Wochentag 76  
WParam 285  
WQL-Abfrage 509  
WQL-Abfrage 517  
WqlObjectQuery 520  
WriteBase64 629  
WriteEndDocument 621  
WriteEndElement 621  
WriteEntry 763  
WriteLineIf 756  
WriteStartDocument 621  
WriteStartElement 621  
WSDL 510

## X

XML 472, 621  
    Bilder speichern 629  
    Binärdaten lesen 632  
    Encoding 621  
    Namensraum 643  
    Prüfen 668  
    Rekursion 633  
    Schema 668  
    Serialisieren 685, 692  
    Stylesheet 674  
    Suche mit XPathNavigator 646  
    Suchen mit XPath 638  
    Transformationen 674  
    Validieren 668  
XmlArray, XmlSerializer 689

XmlAttribute, XmlSerializer 688  
XmlAttributeAttribute 685  
XmlDataDocument 681  
XmlDocument 633, 636  
XML-Editor 809  
XmlElement 636, 688  
XmlEnum, XmlSerializer 688  
XmlIgnore 685  
    XmlSerializer 687  
XmlInclude, XmlSerializer 690  
XmlNamespaceManager 645, 649  
XmlNodeList 683  
XmlResolver 675  
XmlRoot, XmlSerializer 687  
XmlSerializer 111, 685  
    unbekannte Knoten 691  
XmlText, XmlSerializer 689  
XmlTextReader 625, 633, 671  
XmlTextWriter 621  
XmlValidatingReader 671  
XOR-Drawing 163  
XPath 638, 646, 683  
    Testprogramm 639  
XPathDocument 646  
XPathExpression 649  
XPathNavigator 646, 661  
XPathNavigator.Select 649  
XPathNodeIterator 648  
XSD 668  
xsd.exe 684  
XSDErrorHandler 671  
XSL 674  
XSLT  
    Parameter übergeben 677  
    zusätzliche Funktionen 678  
XsltArgumentList 677  
XSLT-Datei 674  
XslTransform 675, 677

## Z

Zahlenwerte formatieren 35  
Zeichenkette, RTF 512  
Zeit 73, 700  
Zeitangaben 712  
Zeiten, mit hoher Auflösung messen 771  
Zeitspanne 74  
Zeitverschiebung 99  
Zeitzone 73  
Zoom, Bildausschnitt 162  
Zugriffsmodifizierer 786  
Zwischenablage 277



## 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