

Programmierung mit der .NET-Klassenbibliothek

Die Reihe Programmer's Choice

Von Profis für Profis

Folgende Titel sind bereits erschienen:

Bjarne Stroustrup

Die C++-Programmiersprache

1072 Seiten, ISBN 3-8273-1660-X

Elmar Warken

Kylix – Delphi für Linux

1018 Seiten, ISBN 3-8273-1686-3

Don Box, Aaron Skonnard, John Lam

Essential XML

320 Seiten, ISBN 3-8273-1769-X

Elmar Warken

Delphi 6

1334 Seiten, ISBN 3-8273-1773-8

Bruno Schienmann

Kontinuierliches Anforderungsmanagement

392 Seiten, ISBN 3-8273-1787-8

Damian Conway

Objektorientiertes Programmieren mit Perl

632 Seiten, ISBN 3-8273-1812-2

Ken Arnold, James Gosling, David Holmes

Die Programmiersprache Java

628 Seiten, ISBN 3-8273-1821-1

Kent Beck, Martin Fowler

Extreme Programming planen

152 Seiten, ISBN 3-8273-1832-7

Jens Hartwig

PostgreSQL – professionell und praxisnah

456 Seiten, ISBN 3-8273-1860-2

Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides

Entwurfsmuster

480 Seiten, ISBN 3-8273-1862-9

Heinz-Gerd Raymans

MySQL im Einsatz

618 Seiten, ISBN 3-8273-1887-4

Dusan Petkovic, Markus Brüderl

Java in Datenbanksystemen

424 Seiten, ISBN 3-8273-1889-0

Joshua Bloch

Effektiv Java programmieren

250 Seiten, ISBN 3-8273-1933-1

Holger Schwichtenberg, Frank Eller

Programmierung mit der .NET-Klassenbibliothek

Zugriff auf das Windows-Betriebssystem
mit Visual Basic .NET und C#



An imprint of Pearson Education

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

Die Deutsche Bibliothek – CIP-Einheitsaufnahme

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

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

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

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

Umwelthinweis:

Dieses Produkt wurde auf chlorfrei gebleichtem Papier gedruckt.
Die Einschrumpffolie – zum Schutz vor Verschmutzung – ist aus umweltverträglichem
und recyclingfähigem PE-Material.

5 4 3 2 1

05 04 03 02

ISBN 3-8273-1905-6

© 2002 by Addison-Wesley Verlag,
ein Imprint der Pearson Education Deutschland GmbH
Martin-Kollar-Straße 10–12, D-81829 München/Germany
Alle Rechte vorbehalten
Einbandgestaltung: Christine Rechl, München
Titelbild: Trollius ledebourii, Trollblume, Goldranunkel. © Karl Blossfeldt Archiv –
Ann und Jürgen Wilde, Zülpich/VG Bild-Kunst Bonn, 2002.
Lektorat: Sylvia Hasselbach, shasselbach@pearson.de
Korrektorat: Astrid Schürmann, Düsseldorf
Herstellung: Monika Weiher, mweiher@pearson.de
CD-Mastering: Gregor Kopietz, gkopietz@pearson.de
Satz: reemers publishing services gmbh, Krefeld, www.reemers.de
Druck und Verarbeitung: Bercker, Kevelaer
Printed in Germany

Inhalt

	Vorwort	17
	Was dieses Buch erreichen will	17
	Für wen dieses Buch geeignet ist	18
	Was Sie nicht in diesem Buch finden werden	19
	Wie Sie dieses Buch lesen sollten	19
	Website zum Buch	20
I	Die .NET-Klassenbibliothek	21
1.1	Einführung	21
1.1.1	Funktionsumfang	22
1.1.2	Implementierung	23
1.1.3	Grundkonzept der FCL	23
1.1.4	Hinweise zu den Vorab-Versionen	25
1.1.5	Nutzbarkeit der FCL von Unmanaged Clients	26
1.1.6	Verfügbarkeit	27
1.2	Typen in .NET	27
1.2.1	Klassen	28
1.2.2	Schnittstelle (Interface)	31
1.2.3	Strukturen (Werteklassen)	32
1.2.4	Aufzählungstypen (Enumerationen)	33
1.2.5	Typnamen und Namespaces	34
1.2.6	Typhierarchien	36
1.3	Verwendung von FCL-Typen	38
1.3.1	Assembly-Referenz	38
1.3.2	Importieren von Namespaces	38
1.3.3	Instanziierung	40
1.3.4	Vererbung von Klassen	40
1.3.5	Sichtbarkeit und Zugriff	41
1.4	Überblick über die FCL-Namespaces	42
1.5	Implementierung der FCL	47
1.6	Original-Dokumentation der FCL	54
1.6.1	Position der FCL-Dokumentation	54
1.6.2	Aufbau der FCL-Dokumentation	55
1.6.3	Weitere Dokumentation	57

1.7	Werkzeuge	57
1.7.1	Windows Class Viewer (WinCV.exe)	57
1.7.2	Objektbrowser	58
1.7.3	Class Browser	60
1.7.4	Reflector	60
1.7.5	Intermediation Language Disassembler	61
1.8	Standardisierung	62
2	Visual Basic .NET (VB 7.0)	65
2.1	Änderungen von VB 6.0 zu VB.NET (VB 7.0)	66
2.1.1	Integration in das .NET Framework	67
2.1.2	Verbesserte Objektorientierung	68
2.1.3	Veränderung bei den Sprachkonstrukten	69
2.1.4	Neue Konstrukte	71
2.1.5	Bewertung	72
2.2	Entwicklung von VB.NET-Programmen mit dem Kommandozeilen-Compiler	73
2.2.1	Erstellung einer Hello-World-Anwendung	73
2.2.2	Entwicklung einer DOS-Anwendung	75
2.2.3	Nutzung von FCL-Klassen	76
2.3	Nutzung der Visual Studio .NET-Entwicklungsumgebung	79
2.3.1	Visual Studio .NET-Überblick	79
2.3.2	Projektmappen und Projekte	83
2.3.3	Ausgaben in der VS.NET-Entwicklungsumgebung	89
2.3.4	Erstellung einer Anwendung in VS.NET	90
3	C# vs. Visual Basic .NET	95
3.1	Grundlagen	95
3.1.1	C# ist case-sensitive	96
3.1.2	Anweisungen und Zeilen	97
3.1.3	Kommentare	98
3.1.4	Codeblöcke	101
3.1.5	Option-Statements	102
3.1.6	Schlüsselwörter	102
3.1.7	Operatoren	105
3.2	Deklarationen	109
3.2.1	Klassen und Module	109
3.2.2	Variablen und Felder	110
3.2.3	Methoden	111
3.2.4	Eigenschaften	112
3.2.5	Konstruktoren und Destruktoren	114
3.3	Programmiertechniken	117
3.3.1	Schleifen	117
3.3.2	Verzweigungen	122
3.3.3	Casting	126
3.3.4	With-Anweisung	127
3.3.5	Arrays	128
3.4	Unterschiede in der Entwicklungsumgebung	130

4	Namespace System	133
4.1	Klasse System.Object	133
4.1.1	GetType()	134
4.1.2	ToString()	135
4.1.3	GetHashCode()	136
4.1.4	Equals()	136
4.1.5	ReferenceEquals()	137
4.1.6	Beispiel zur Anwendung der Klasse System.Object	137
4.1.7	MemberwiseClone()	140
4.2	Klassen für Basis-Datentypen	144
4.2.1	Übersicht	144
4.2.2	System.Void	146
4.2.3	Operationen der Basisdatentyp-Klassen	146
4.2.4	System.String	148
4.3	Weitere Datentypen	156
4.3.1	System.Guid	157
4.3.2	System.Uri und System.UriBuilder	159
4.3.3	System.Version	163
4.4	Klasse System.Console	164
4.5	Klasse System.BitConverter	167
4.6	Klasse System.Convert	169
4.7	Klasse System.Math	172
4.8	Klasse System.Random	172
4.9	Klasse System.GC	174
4.10	Klasse System.Environment	179
4.11	Weitere Klassen im Namespace System	185
5	Namespace System.Collections	187
5.1	Klassen in System.Collections	187
5.1.1	Die Klasse ArrayList	189
5.1.2	Die Klassen Stack und Queue	197
5.1.3	Die Klasse BitArray	201
5.1.4	Die Klasse Hashtable	204
5.2	Eigene Collections erzeugen	207
5.3	Spezialisierte Collections	220
6	Namespace System.IO	223
6.1	Zugriff auf die Struktur des Dateisystems	223
6.1.1	Objektmodell	224
6.1.2	Verzeichnisse auflisten	225
6.1.3	Verzeichnisse und Dateien suchen	226
6.1.4	Verzeichnisse anlegen und löschen	229
6.1.5	Dateien anlegen und löschen	232
6.1.6	Kopieren und Verschieben	233
6.1.7	Informationen über das Dateisystem	234
6.1.8	Arbeit mit Pfadangaben	235
6.2	Zugriff auf Dateiinhalte	237

6.2.1	Readers, Writers und Streams	237
6.2.2	Textdatei schreiben	239
6.2.3	Textdatei lesen	241
6.2.4	Binärdatei schreiben	242
6.2.5	Binärdatei lesen	243
6.3	Dateisystem überwachen	244
7	Namespaces System.Text und System.Text.RegularExpressions	251
7.1	Die Klasse StringBuilder	251
7.1.1	Eigenschaften und Methoden	252
7.1.2	Erzeugen eines Zufalls-Strings	253
7.1.3	Mehrfaches Einfügen von Strings	255
7.2	Reguläre Ausdrücke	257
7.2.1	Übersicht	257
7.2.2	Suchausdrücke (Patterns)	260
7.2.3	Die Klasse Regex	261
8	Namespaces System.Globalization und System.Resources	271
8.1	Lokalisierung	271
8.1.1	Ressourcendateien	272
8.1.2	Fehler vermeiden	272
8.2	Kalenderklassen	274
8.2.1	Umrechnen von Datumsangaben	277
8.3	Lokalisierung von Programmen	279
8.3.1	Die Klasse CultureInfo	280
8.3.2	Lokalisierung mit Visual Studio .NET	287
8.4	Zugriff auf Ressourcendateien	293
8.4.1	Zusammenhänge	293
8.4.2	Die Klasse ResourceManager	295
8.4.3	Eine Anwendung mit lokalisierten Strings	295
9	Der Namespace System.Xml	299
9.1	XmlReader/XmlWriter	300
9.1.1	XML-Daten aus Dateien lesen	302
9.1.2	XML validieren	311
9.1.3	XML-Daten schreiben	318
9.2	XmlDocument	324
9.2.1	Auslesen der Daten	325
9.2.2	Daten in XmlDocument ändern bzw. hinzufügen	332
9.3	XPath	341
9.3.1	XPathNodeIterator	343
9.3.2	XPathNavigator	343
9.4	XslTransform	345

10	Namespace System.Data (ADO.NET)	349
10.1	Architektur	350
10.2	Objektmodell	352
10.3	Hinweis zu den Beispielen	356
10.4	Daten-Verbindungen	357
10.5	Daten lesen mit dem OleDbDataReader	360
10.6	DataSets	361
10.6.1	DataTables	362
10.6.2	Datenadapter	365
10.6.3	Lesezugriff auf Daten	366
10.7	Daten ändern mit DataSets	373
10.7.1	Befehle generieren mit dem CommandBuilder	374
10.7.2	Datensätze ändern	376
10.7.3	Datensätze anfügen	378
10.7.4	Datensätze löschen	380
10.8	Daten ändern mit OleDbCommand	382
10.9	Verknüpfung von Tabellen in einem DataSet	384
10.10	Dynamische Tabellen im Speicher	388
10.11	ADO.NET und XML	390
10.11.1	Ausgabe eines DataSets als XML	390
10.11.2	XmlDataDocument	393
10.11.3	XML in DataSet einlesen	396
10.12	Weitere Möglichkeiten von ADO.NET	399
11	Namespaces System.Runtime.Serialization und System.Xml.Serialization	401
11.1	Serialisierer in .NET	402
11.2	Binärer Serialisierer	403
11.2.1	Serialisieren in Binärdatei	404
11.2.2	Serialisieren in Byte-Folge oder Datenbank	405
11.3	SOAP-Serialisierer	409
11.4	XML-Serialisierer	410
11.4.1	XML-Serialisierung	411
11.4.2	XML-Deserialisierung	413
11.4.3	Anwendungsbeispiel	414
11.5	Serialisierung eines Objektmodells	416
11.6	Änderungen an serialisierten Objekten	421
11.7	Benutzerdefinierte (De-)Serialisierung	422
11.7.1	Benutzerdefinierte Serialisierung für BinaryFormatter und SOAPFormatter	422
11.7.2	Benutzerdefinierte Serialisierung für den XmlSerializer	426
12	Namespace System.ServiceProcess	429
12.1	Architektur von Windows-Diensten	429
12.2	Die Klasse ServiceBase	431
12.2.1	Eigenschaften und Methoden	431
12.2.2	Die Dienst-Hauptmethode	433

12.3	Die Klassen ServiceProcessInstaller und ServiceInstaller	434
12.3.1	Beispiel: Ein einfacher Dienst	436
12.3.2	Das Setup-Projekt für den Dienst	442
12.4	Die Klasse ServiceController	444
12.4.1	Eigenschaften und Methoden	444
12.4.2	Dienste auflisten	446
13	Namespace Microsoft.Win32	449
13.1	Die Klasse SystemEvents	449
13.1.1	Standardereignisse	449
13.2	Zugriff auf die Registry	453
13.2.1	Die Klasse Registry	454
13.2.2	Die Klasse RegistryKey	455
14	Namespace System.Diagnostics	469
14.1	Arbeit mit den Ereignisprotokollen	469
14.1.1	Überblick über die Ereignisprotokoll-Klassen	470
14.1.2	Auslesen eines Ereignisprotokolls	470
14.1.3	Liste aller Ereignisprotokolle	472
14.1.4	Anlegen eines neuen Ereignisprotokolls	473
14.1.5	Löschen eines Ereignisprotokolls	474
14.1.6	Schreiben in das Ereignisprotokoll	475
14.1.7	Überwachung eines Ereignisprotokolls	476
14.2	Arbeit mit Prozessen	477
14.2.1	Überblick über die FCL-Klassen	478
14.2.2	Liste der Prozesse	480
14.2.3	Liste der Module	483
14.2.4	Liste der Threads	485
14.2.5	Prozesse starten	486
14.2.6	Prozesse beenden	487
14.2.7	Nicht reagierende Prozesse beenden	489
14.3	Informationen über Programmdateien	489
14.4	Arbeit mit Leistungsindikatoren	490
14.4.1	Grundlagen	491
14.4.2	Überblick über die FCL-Klassen	492
14.4.3	Lesen eines Leistungsindikators	493
14.4.4	Anlegen und Löschen eines Leistungsindikators	494
14.4.5	Schreiben von Werten in einen Leistungsindikator	496
15	Namespace System.DirectoryServices	501
15.1	Architektur	501
15.2	Objektmodell	503
15.2.1	Allgemeine Klassen	503
15.2.2	Klassen für die Ausführung von Suchanfragen	505
15.2.3	Vergleich zwischen .NET-ADSI und COM-ADSI	506

15.3	Überblick über die Programmiermechanismen	506
15.3.1	Objektidentifikation und Bindung	506
15.3.2	Zugriff auf Attribute und Methoden	508
15.3.3	Zugriff auf Container-Objekte	514
15.3.4	Instanzenverwaltung	515
15.4	Active Directory-Programmierung	518
15.4.1	Grundlagen zur Active Directory-Programmierung	518
15.4.2	Testumgebung	523
15.4.3	Ermittlung der Partitionen	524
15.4.4	Informationen über einen Benutzer	525
15.4.5	Benutzer ändern	528
15.4.6	Organisationseinheit anlegen	529
15.4.7	Organisationseinheit löschen	530
15.4.8	Benutzer anlegen	531
15.4.9	Benutzerkennwort setzen	532
15.4.10	Benutzer löschen	533
15.4.11	Benutzer umbenennen	534
15.4.12	Benutzer verschieben	535
15.4.13	Gruppe anlegen	536
15.4.14	Benutzer einer Gruppe hinzufügen	537
15.4.15	Benutzer aus Gruppe entfernen	537
15.4.16	Gruppe löschen	538
15.4.17	Containerinhalt auflisten	539
15.4.18	Computerkonto anlegen	540
15.4.19	Computerkonto löschen	541
15.4.20	Suche im Active Directory	541
15.5	Verzeichnisdienstprogrammierung mit dem WinNT-Provider	545
15.5.1	Grundlagen zum NT4-Verzeichnisdienst	545
15.5.2	Benutzerliste ausgeben	548
15.5.3	Benutzer verwalten	549
15.5.4	Dienste verwalten	550
15.5.5	Dateisystem-Dienst verwalten	552
16	Namespace System.Threading	557
16.1	Funktionsweise von Threads	557
16.2	Threads starten und stoppen	558
16.2.1	Thread und ThreadStart	558
16.2.2	Anhalten und Fortsetzen von Threads	562
16.3	Thread-Prioritäten setzen	572
16.4	Synchronisation	574
16.4.1	Wozu synchronisieren?	574
16.4.2	Die Klasse Monitor	575
16.4.3	Die Anweisung SyncLock	581
16.5	Threadpools	582

17	Namespace System.Windows.Forms	595
17.1	Grundlagen	595
17.1.1	Die Quelldatei des Programms	595
17.1.2	Start des Programms	597
17.1.3	Anordnen von Steuerelementen	598
17.2	Arten von Steuerelementen	599
17.2.1	Controls	599
17.2.2	Nicht-visuelle Steuerelemente	600
17.2.3	Dialoge	600
17.2.4	Übersicht	600
17.3	Datenbindung	602
17.4	Die Klasse Form	605
17.4.1	Eigenschaften	606
17.4.2	Formulare dynamisch erzeugen	609
17.4.3	Zugriff auf Komponenten eines Formulars	611
17.4.4	Zugriff auf Komponenten des Hauptformulars	612
17.4.5	MDI-Applikationen	616
17.5	Arbeiten mit den Steuerelementen	625
17.5.1	Dynamische Erzeugung von Steuerelementen	625
17.5.2	Eingaben kontrollieren	627
17.5.3	Drag&Drop	629
18	Der Namespace System.Drawing	637
18.1	Die Klasse Graphics	638
18.2	Hilfsklassen	640
18.3	Füllwerkzeuge	642
18.3.1	Die Klasse SolidBrush	642
18.3.2	Die Klasse LinearGradientBrush	643
18.3.3	Die Klasse PathGradientBrush	648
18.3.4	Die Klasse TextureBrush	652
18.4	Bitmaps	653
18.4.1	Die Klasse Pen	653
18.4.2	Einfache Pens verwenden	654
18.4.3	Zeichenstift mit Farbverlauf	658
18.5	Arbeiten mit Grafiken	662
18.5.1	Die Klasse Image	662
18.5.2	Die Klasse Bitmap	668
19	Namespace System.Security.Cryptography	679
19.1	Verschlüsselungstechniken	680
19.1.1	Private Schlüssel	680
19.1.2	Öffentliche Schlüssel	681
19.1.3	Hashcodes	681
19.1.4	Digitale Signatur	682
19.2	Hash-Algorithmen	682

19.3	Texte verschlüsseln	686
19.3.1	Asymmetrische Verschlüsselung	687
19.3.2	Symmetrische Verschlüsselung	694
19.4	Verschlüsselung in der Realität	701
19.4.1	Alice und Bob	701
19.4.2	Die Funktionalität	703
20	Namespace System.Reflection	713
20.1	Das Reflection-API	713
20.2	Die Klasse Type	714
20.2.1	Basisinformationen ermitteln	715
20.2.2	Detailinformationen ermitteln	720
20.3	Die Klassen Assembly und Module	728
20.3.1	Informationen über Assemblies ermitteln	728
20.3.2	Namespaces ermitteln	733
20.4	Code on the fly mit Reflection.Emit	741
20.4.1	Erzeugen einer Assembly	743
20.4.2	Die Klasse TypeBuilder	745
20.4.3	Die Klasse ILGenerator	746
20.4.4	Hello World mit Reflection.Emit	747
21	Namespaces System.Net und System.Net.Sockets	751
21.1	Der Namespace System.Net	751
21.1.1	Die Klasse WebClient	752
21.1.2	Die Klasse WebRequest	755
21.1.3	Die Klasse WebResponse	756
21.2	Der Namespace System.Net.Sockets	762
21.2.1	Die Klasse NetworkStream	763
21.2.2	Die Klasse TcpClient	763
21.3	Hilfsklassen	772
22	Namespace System.Management	775
22.1	Windows Management Instrumentation (WMI)	775
22.2	Programmieren mit dem Meta-Objektmodell	776
22.2.1	Überblick über das Objektmodell	776
22.2.2	Objektbindung	778
22.2.3	Zugriff auf Attribute und Methoden	780
22.2.4	Lesezugriff	781
22.2.5	Schreibzugriff	783
22.2.6	Methodenaufrufe	784
22.2.7	Informationen über WMI-Objekte	785
22.2.8	Auflisten von Objektmengen	788
22.3	Programmieren mit Wrapper-Klassen	789
22.3.1	Erzeugung einer Wrapper-Klasse	789
22.3.2	Verwendung der erzeugten Klasse	791
22.3.3	Verwendung der erzeugten Objektmengen	792

22.4	WQL-Abfragen	794
22.4.1	Datenabfragen	796
22.4.2	Ereignisabfragen	799
22.5	WMI im Visual Studio .NET Server Explorer	802
22.6	Weitere Beispiele	808
22.6.1	Abmelden, Herunterfahren und Neustart	808
22.6.2	Netzwerkverbindungen	810
22.6.3	Speichernutzung	811
22.6.4	ASP.NET-Leistungsdaten	812

A .NET-Glossar 815

B Active Directory Service Interface (ADSI) 853

B.1	Verzeichnisdienste	854
B.1.1	Aufbau eines Verzeichnisbaums	855
B.1.2	Verzeichnisdienstschemata	857
B.1.3	Objektidentifikation	858
B.1.4	Meta-Directories	859
B.1.5	Lightweight Directory Access Protocol (LDAP)	859
B.2	ADSI-COM-Komponente	863
B.2.1	Architektur	863
B.2.2	Installation	865
B.2.3	Verfügbare ADSI-Provider	865
B.2.4	ADSI Software Development Kit	866
B.2.5	Meta-Objektmodell	867
B.2.6	Überblick über die ADSI-COM-Standardschnittstellen	870
B.3	ADSI-Werkzeuge	876
B.3.1	Active Directory Service Browser (ADB)	876
B.3.2	ADSI Explorer	877
B.3.3	ADSI Edit	878
B.3.4	ADSI Schema Browser	880

C Windows Management Instrumentation (WMI) 881

C.1	Überblick über die Managed Objects	882
C.2	Installation und Konfiguration	883
C.3	Architektur	884
C.4	Programmierung	885
C.5	Konzepte	886
C.5.1	Schema	887
C.5.2	Namespaces	888
C.5.3	Lokalisierung	889
C.5.4	Klassen und Objekte	890
C.5.5	Managed Object Format (MOF)	893
C.5.6	Sicherheitsfunktionen	895
C.5.7	Ereignisse	895
C.5.8	WMI Query Language (WQL)	898

C.6	Werkzeuge	898
C.6.1	WMI-Steuerung	899
C.6.2	WMI Object Browser	899
C.6.3	WMI CIM Studio	902
C.6.4	WMI Event Registration Tool	905
C.6.5	WMI Event Viewer	907
C.6.6	VBIInstance	908
C.6.7	WMI-Testprogramm	908
C.6.8	MOF Compiler	909
D	Hinweise zum Buch	911
D.1	Sprachliche Konventionen	911
D.2	Hinweise zur Formatierung des Textes	912
D.3	Hinweise zu den Code-Beispielen	913
D.4	Grafische Notation in den Objektdiagrammen	917
D.4.1	Knoten (geometrische Formen)	918
D.4.2	Kanten (Verbindungslinien)	919
E	CD-ROM und Website	923
E.1	Der Inhalt der CD-ROM	923
E.2	Die Website zu diesem Buch	923
F	Abkürzungsverzeichnis	925
G	Literaturverzeichnis	933
G.1	Gedruckte Literatur	933
G.2	Quellen im Internet	934
G.2.1	Links	934
G.2.2	Requests for Comment (RFCs)	936
G.2.3	Newsgroups zu .NET	937
G.2.4	Newsgroups zu speziellen COM-Komponenten	937
	Index	939

Vorwort

Zum Microsoft .NET Framework gehört eine sehr umfangreiche Klassenbibliothek, die *.NET Framework Class Library (FCL)*, alias *Basisklassenbibliothek* (engl. *Base Class Library*, kurz *BCL*). Diese Klassenbibliothek ist universell in allen .NET-fähigen Sprachen einsetzbar.

Viele Klassen

Umfangreich – das heißt 2246 Klassen, 260 Schnittstellen, 468 Aufzählungstypen und 96 Strukturen. Dabei haben wir nur die öffentlich zugänglichen Typen gezählt. Die Framework Class Library Version 1.0 enthält intern weitere 3997 Typen.

Was dieses Buch erreichen will

Ein Blick auf die Originaldokumentation von Microsoft lässt einen schon fast verzweifeln, so groß sind die Mengen an Information, die es hier bezüglich der .NET-Klassenbibliothek zu bewältigen gilt. Man könnte es mit einiger Berechtigung einen Dschungel aus Klassen nennen. Mit diesem Buch wollen wir Ihnen einen Weg durch diesen Dschungel schlagen und Ihnen zeigen, wie Sie die umfangreichen Möglichkeiten des .NET Frameworks nutzen können. Wir haben die wichtigsten Klassen herausgesucht und liefern Ihnen dazu nicht nur Beispiele, sondern auch Objektdiagramme und wichtige Hintergrundinfos.

Zielsetzung

Zielsetzung dieses Buchs ist es, Ihnen den Einsatz der .NET-Klassenbibliothek zum Zugriff auf Windows zu demonstrieren. Alle Beispiele in diesem Buch sind nach dem Keep-It-Simple-Prinzip erstellt worden: Wir haben einfache, aber dennoch ausdrucksstarke Beispiele geschaffen, die jeweils eine einzelne, überschaubare Aufgabe erledigen. Viele Beispiele bestehen bewusst nur aus einer einzigen Unterroutine und liefern nur Text-Ausgaben. Wir möchten verhindern, dass Sie mühsam durch mehrseitigen Code blättern müssen, im Gegenteil, Sie sollen die Verwendung einer

Klasse bzw. einer Menge von Klassen auf den ersten Blick verstehen können. Wenn sinnvoll, liefern wir Ihnen als Beispiel aber auch Windows-Anwendungen mit grafischer Benutzeroberfläche.

Für wen dieses Buch geeignet ist

Zielgruppe Die primäre Zielgruppe dieses Buchs sind Softwareentwickler, die Windows- oder Webanwendungen mit dem .NET Framework entwickeln wollen. Dabei kommt es nicht darauf an, mit welcher Sprache Sie arbeiten. Zwar wird im Buch durchgehend Visual Basic .NET in den Beispielen verwendet, die vorgestellten Klassen sind aber mit jeder .NET-Sprache nutzbar.

Dass wir nicht jedes Beispiel in mehreren Sprachen nebeneinander gestellt haben, hat natürlich seinen Grund. In diesem Fall einen ganz pragmatischen: Platzmangel. Ebenso haben wir darauf verzichtet, zwischen den Kapiteln oder zwischen den Beispielen die Programmiersprache zu wechseln. Das stiftet nur Verwirrung und Sie könnten auch sicher sein, dass Sie das gesuchte Beispiel garantiert nicht in der Sprache finden würden, in der Sie es suchen. Wir haben uns für Visual Basic .NET entschieden, weil die meisten Programmierer diese Sprache (gut) kennen. Da wir auch C# als sehr wichtig erachten, stellen wir beide Sprachen in einem Kapitel gegenüber, wodurch es sehr schnell möglich ist, die angegebenen Beispiele zu portieren.

Auch Windows-Administratoren können (etwas Programmierkenntnisse vorausgesetzt) von diesem Buch profitieren. Schließlich enthält die Framework Class Library unzählige Möglichkeiten, den administrativen Alltag zu vereinfachen: Benutzer anlegen, Dienste verwalten, Prozesse überwachen, Ändern im Dateisystem, Zugriff auf die Windows Management Instrumentation usw. Das bisher COM-basierte Windows Scripting wird durch verschiedene Techniken im .NET Framework abgelöst werden: Statt VBScript sieht Microsoft nun Visual Basic .NET als Sprache zur Entwicklung von Automatisierungslösungen vor. Und an die Stelle von COM-Komponenten wie ADSI, WMI, FSO usw. tritt die Framework Class Library. Damit haben Administratoren dann aber wesentlich mehr Möglichkeiten als mit vielen COM-Komponenten, die oftmals nur unvollständige Wrapper für das Win32-API waren.

Was Sie nicht in diesem Buch finden werden

Die Framework Class Library ist ein umfangreiches Thema und doch selbst nur Teil eines viel umfangreicheren Themas, des .NET Framework, das wieder nur ein Teil der .NET-Strategie von Microsoft ist. Wir möchten auch klarstellen, was Sie auf den folgenden ca. 850 Seiten nicht finden werden:

Abgrenzung

- ▶ Informationen über die .NET-Strategie. Das meiste davon ist sowieso »Marketing« und hat in einem technischen Buch nichts zu suchen. Hierzu sei [WES02] empfohlen.
- ▶ Eine detaillierte Einführung in das .NET Framework. Wir setzen voraus, dass Sie sich schon einmal mit .NET beschäftigt haben. Hierzu sei ebenfalls auf [WES02] verwiesen. Sie müssen aber noch kein Experte sein: Das .NET-Glossar in Anhang A dieses Buchs hilft Ihnen mit einer ausführlichen Erläuterung, bestehende Wissenslücken zu schließen.
- ▶ Jede Klasse aus der Framework Class Library. Bei einer »vollständigen« Beschreibung der 2541 Klassen auf 850 Seiten wäre das ja höchstens ein Drittel einer Seite pro Klasse gewesen. Wir haben uns für mehr Tiefe bei den wichtigen Klassen entschieden. Sie finden eine Referenz aller Klassen im .NET Framework SDK, das auf der CD-ROM zu diesem Buch mitgeliefert wird.
- ▶ Webprogrammierung mit ASP.NET. Zu diesem Themengebiet gibt es viele andere Bücher, zum Beispiel [SCH02a] und [SCH02b].
- ▶ Die Namespaces `System.EnterpriseServices` (COM+-Programmierung), `System.Messaging` (MSMQ-Programmierung), `System.Runtime.Remoting` und `System.Security`. Beschränkter Platz und konzeptionelle Erwägungen haben uns diese Themen ausklammern lassen.
- ▶ JScript.NET, JSharp.NET (J#) und .NET-Sprachen, die nicht aus dem Hause Microsoft stammen.

Wie Sie dieses Buch lesen sollten

Dieses Buch ist mehr ein Nachschlagewerk denn ein Buch mit durchgehendem Handlungsstrang. Die Kapitel 1 bis 3 bieten Ihnen eine Einführung in die Framework Class Library und die

Lese-Anleitung

Nutzung mit den Sprachen Visual Basic .NET und C#. Diese Kapitel sollten Sie in der gegebenen Reihenfolge lesen.

Der Rest des Buchs widmet sich den Namespaces der FCL, wobei meistens ein Kapitel genau einen Namespace behandelt und in sich abgeschlossen ist. Es gibt natürlich in den Beispielen einige Rückgriffe, die Sie aber auch bei Bedarf nachschlagen können. So steht es Ihnen frei, die Kapitel 4 bis 22 gemäß Ihrem individuellen Wissensdurst zu lesen. Die von uns gewählte Reihenfolge gewährleistet, dass Sie nicht viel hin und her springen und auch nicht gleich mit den kompliziertesten Klassen anfangen müssen.

Der Anhang liefert Ihnen neben einem Glossar zum .NET Framework auch zwei Grundlagenkapitel zum Active Directory Service Interface (ADSI) und zur Windows Management Instrumentation (WMI). Diese Hintergrundinfos sind eine Ergänzung zu den Kapiteln 15 (`System.DirectoryServices`) und 22 (`System.Management`) für diejenigen, die sich bisher nicht mit ADSI und WMI beschäftigt haben. Im Anhang finden Sie auch Hinweise zu den in diesem Buch verwendeten sprachlichen und grafischen Notationen und dem Inhalt der bei diesem Buch mitgelieferten CD-ROM.

Website zum Buch

Wenn Sie Fragen und Anregungen haben oder Sie einmal einen Fehler in diesem Buch finden (hoffentlich nicht ...) oder wenn Sie uns loben wollen J, dann besuchen Sie uns doch auf <http://www.dotnet-komponenten.de>. Hier finden Sie auch Aktualisierungen des Programmcodes aus diesem Buch.

Viel Spaß und Erfolg mit dem Buch wünschen Ihnen

Frank Eller



Holger Schwichtenberg



München/Essen, im Mai 2002

I Die .NET-Klassenbibliothek

Dieses Kapitel bietet Ihnen eine Einführung in den Funktionsumfang und die Nutzung der .NET-Klassenbibliothek. Es ist aber keine Einführung in das komplette .NET Framework; dies würde den Umfang des Buches sprengen. Anhang A bietet Ihnen jedoch ein Glossar zu den wichtigsten Begriffen rund um .NET.

I.1 Einführung

Zum Microsoft .NET Framework gehört eine umfangreiche Klassenbibliothek – in der Version 1.0 mit 2541 öffentlichen Klassen. Diese Klassenbibliothek heißt .NET Framework-Klassenbibliothek (.NET Framework Class Library, kurz: *FCL*) oder auch *Basisklassenbibliothek* (*Base Class Library*, kurz *BCL*). Teile der FCL sind im Rahmen der Common Language Infrastructure (CLI) standardisiert. Hinweise zum CLI-Standard finden Sie in Kapitel 1.8.

FCL / BCL

Welche Klassen alle zur FCL gehören, wird von verschiedenen Quellen unterschiedlich bewertet. In diesem Buch werden als FCL diejenigen Klassen bezeichnet, die zum Wurzel-Namespace `System` gehören, deren voll qualifizierter Name also mit `System.*` beginnt. In der .NET Framework SDK-Dokumentation werden in der Klassenreferenz auch zahlreiche Klassen des Wurzel-Namespace `Microsoft.*` dokumentiert.



Einen Namespace, der nicht zu `System.*` gehört, werden Sie in diesem Buch finden: `Microsoft.Win32`. Hier findet man u.a. Klassen für Betriebssystemereignisse und die Registry. Diese Systembausteine wurden aus `System.*` verbannt, weil sie zu Windows-spezifisch sind. Dennoch: In einem Buch über Windows-Programmierung dürfen die Klassen für den Zugriff auf diese Systembausteine nicht fehlen. Außerdem ist dieser Namespace in den Assemblies `mscorlib.dll` und `system.dll` implementiert, die zweifelsfrei zur Framework Class Library gehören.



1.1.1 Funktionsumfang

Windows-Funktionen

Die FCL umfasst unzählige Funktionen zum Zugriff auf das Windows-Betriebssystem, zum Beispiel folgende Bausteine:

- ▶ Benutzeroberfläche (Fenster und Steuerelemente)
- ▶ Dateisystem
- ▶ Systeminformationen
- ▶ COM+-Transaktionsverwaltung
- ▶ TCP/IP-Funktionen
- ▶ Registry
- ▶ Windows-Dienste
- ▶ Active Directory
- ▶ Internet Information Services (IIS)
- ▶ Ereignisprotokolle
- ▶ Leistungsindikatoren
- ▶ Datenbanken und XML (ADO.NET)

CLR-Funktionen

Daneben enthält die FCL aber auch Funktionen zum Zugriff auf die *Common Language Runtime (CLR)*, die Laufzeitumgebung von .NET. Beispiele für diese Gruppe von Funktionen sind:

- ▶ die elementaren Datentypen des .NET Frameworks (Byte, Integer, String, etc)
- ▶ die .NET-Compiler für VB.NET, C# und JScript.NET
- ▶ .NET-Remoting (Fernzugriff auf .NET-Komponenten)
- ▶ Garbage Collector (Aufräumen des Speichers)
- ▶ Reflection (Auslesen von Typinformationen aus Assemblies)
- ▶ Dynamische Erzeugung von Assemblies in der Microsoft Intermediation Language (MSIL)



Einen detaillierteren Überblick über die Funktionen der FCL erhalten Sie in Kapitel 1.4.

1.1.2 Implementierung

Die .NET-Klassenbibliothek ist in Form von .NET-Komponenten (Assemblies) implementiert. Mehr dazu erfahren Sie in Kapitel 1.5.

1.1.3 Grundkonzept der FCL

Die Basisklassenbibliothek ist eine einheitliche Klassenbibliothek für alle .NET-fähigen Programmiersprachen. Die FCL macht es möglich, dass die meisten Programmieraufgaben in allen .NET-Sprachen fast identisch gelöst werden können – die Unterschiede zwischen den Sprachen reduzieren sich auf die verschiedenen Schlüsselwörter für Sprachkonstrukte wie Deklarationen, Bedingungen und Schleifen. Aufgaben wie das Schreiben in eine Datei oder das Senden einer HTTP-Anfrage können in allen Sprachen mit den gleichen Klassen gelöst werden.

Motivation

Die FCL ist der Ansatz zur Vereinheitlichung unterschiedlicher Funktions- bzw. Klassenbibliotheken von Microsoft. Bisher wurden neben C-DLLs (z.B. dem Win32-API) und COM-Komponenten (z.B. ActiveX Data Objects) auch sprachspezifische Klassenbibliotheken (z.B. die Microsoft Foundation Class (MFC) und die Active Template Library (ATL)) angeboten. Die Funktionen dieser verschiedenen Bibliotheken haben sich dabei zum Teil überschritten, wobei es in jeder Bibliothek eine andere Implementierung für sehr ähnliche Funktionen gab. Das Grundkonzept der einzelnen Bibliotheken war sehr unterschiedlich: Das Win32-API ist funktionsorientiert, COM-Komponenten sind vom Grundsatz her objektorientiert, wobei man im Standardfall (*IUnknown*) nur mit Schnittstellenzeigern arbeitet, während der *IDispatch*-Mechanismus in COM das Objekt als Ganzes in den Vordergrund stellt.

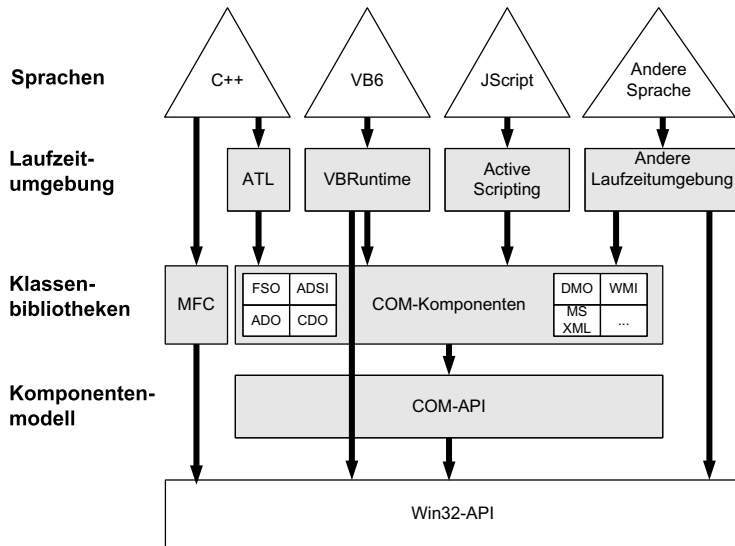
Vereinheitlichung

Sowohl der Entscheidungsprozess zwischen den verschiedenen Bibliotheksalternativen als auch die Einarbeitung in eine Vielzahl unterschiedlicher Konzepte führt zu einem unnötigen Mehraufwand.

Zudem waren gar nicht alle Bibliotheken von jeder Sprache aus nutzbar. Viele Funktionen des Win32-API, aber auch einige COM-Komponenten waren von Visual Basic und Skriptsprachen aus nicht nutzbar. So mussten viele Komponenten erstellt werden, deren einziger Zweck es war, einen Wrapper für nicht zugängliche Bibliotheken zu schaffen.



Die FCL ist die Realisierung einer Klassenbibliothek mit einem durchgängigen Konzept, die von allen Sprachen vollständig und einheitlich genutzt werden kann.



© Holger@Schwichtenberg.de 2001

Abbildung 1.1: Laufzeitumgebungen und Klassenbibliotheken vor .NET

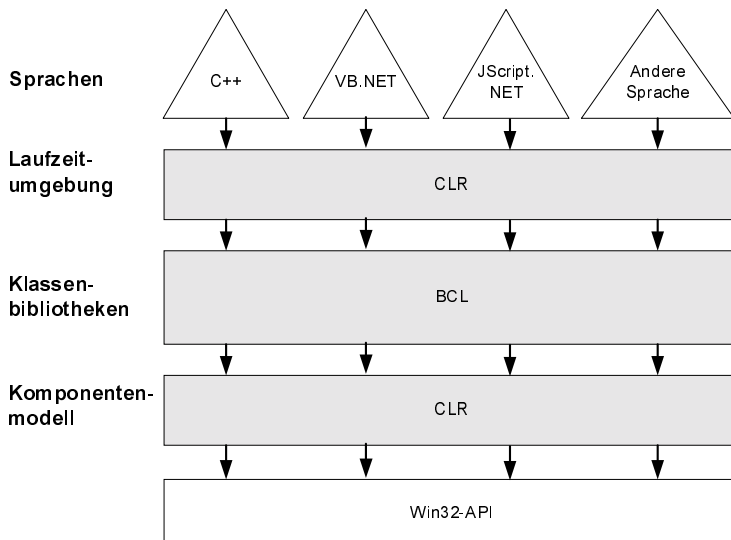


Abbildung 1.2: Laufzeitumgebungen und Klassenbibliotheken in .NET

Kapselung des Win32-API

In Zukunft soll die FCL das gesamte Win32-Application Programming Interface (Win32-API) in FCL-Klassen kapseln. Das trifft auf die Version 1.0 des .NET Frameworks noch nicht zu. Es bleibt zu hoffen, dass zukünftig wirklich alle API-Funktionen auch in der FCL zur Verfügung stehen werden.

Kapselung des WIN32-API

Abwärtskompatibilität

Zur Sicherung der Abwärtskompatibilität unterstützt die CLR auch weiterhin den Aufruf von COM-Komponenten über den so genannten Runtime Callable Wrapper (RCW) und von klassischen Win32-API-Funktionen über einen Mechanismus mit Namen Platform Invocation Service (PInvoke oder P/Invoke abgekürzt). Die FCL unterstützt aber bereits heute viele wichtige Win32-API-Funktionen.



1.1.4 Hinweise zu den Vorab-Versionen

Um die Akzeptanz des .NET Frameworks zu fördern, hat Microsoft der Öffentlichkeit sehr viele Vorab-Versionen zugänglich gemacht. Eine Technical Preview wurde auf der Professional Developers Conference (PDC) im Juli 2000 herausgegeben. Bis zur Verfügbarkeit der Endfassung am 14. Januar 2002 hatte Microsoft eine Beta1, eine Beta2 und einen Release Candidate breit gestreut. Im Internet findet man auch nach Erscheinen der Endfassung noch Werkzeuge und Beispielprogramme, die auf einer der Vorab-Versionen beruhen.

Die nachstehende Tabelle zeigt die bis zum Redaktionsschluss im Umlauf befindlichen Beta-Versionen des .NET Frameworks.

Produktstadium	Versionsnummer
Beta1	1.0.2204
Beta2	1.0.2914
Release Candidate 1	1.0.3328
Endfassung	1.0.3705

Tabelle 1.1: Verfügbare .NET Framework-Versionen

Änderungen zwischen Beta1 und Beta2

Microsoft hat insbesondere zwischen der Beta1 und der Beta2 erhebliche Änderungen an der Klassenbibliothek vorgenommen (vgl. Erläuterungen zur Standardisierung, Kapitel 1.7). Die Namen zahlreicher Namespaces und Klassen sowie der Klassenmitglieder (Attribute, Methoden, Ereignisse) haben sich geändert. Außerdem sind Klassen hinzugekommen. Alex Lowe hat Listen [LOW01] aller Änderungen erstellt: Demnach gibt es 36772 neue Namen, 54950 Namen wurden entfernt und 60302 Klassen und Klassenmitglieder wurden geändert. Selbst erforschen und abtippen musste er die Änderungen nicht: Alle .NET-Komponenten sind selbstbeschreibend. Über das so genannte Reflection-API (System.Reflection, siehe Kapitel 20) kann jedermann die Metadaten einer Komponente auslesen. Eine Routine zu schreiben, die die Typen zweier .NET-Komponenten vergleicht, ist keine große Herausforderung.

Die weitreichenden Änderungen sind der Hauptgrund, warum mit der Beta1 entwickelte .NET-Anwendungen mit späteren Versionen nicht mehr kompilierbar sind. Da sich nicht nur die Namen, sondern zum Teil auch die Objekthierarchien geändert haben, ist es mit einem einfachen Suchen&Ersetzen der Typnamen nicht getan: Oft ist eine komplette Neukodierung notwendig. Allerdings hatte Microsoft auch ausdrücklich darauf hingewiesen, dass die Beta2 weitreichende Änderungen umfassen wird.



Auf der CD-ROM zu diesem Buch finden Sie im Verzeichnis die komplette Liste der Änderungen von Beta1 zu Beta2 [CD:/Weitere Informationen/Änderungen zwischen Beta1 und Beta2] sowie der Änderungen zwischen Beta2 und der Endfassung [CD:/Weitere Informationen/Änderungen seit Beta2] in Form von HTML-Dateien.

1.1.5 Nutzbarkeit der FCL von Unmanaged Clients

COM Callable Wrapper (CCW)

Einige FCL-Klassen können auch von Nicht-.NET-Programmen aus (so genannten *Unmanaged Clients*) via COM genutzt werden. Für die Interoperabilität zwischen COM und .NET stellt die CLR einen so genannten *COM Callable Wrapper* (CCW) zur Verfügung. Damit FCL-Klassen als COM-Komponenten nutzbar sind, müssen die entsprechenden FCL-Assemblies aber zunächst mit dem Werkzeug *regasm.exe* als COM-Klassen in die Registry eingetragen werden. Dieses Tool wird zusammen mit dem .NET-Kern installiert.

regasm.exe erzeugt eine Typbibliothek und daraus dann die notwendigen Registry-Einträge.

```
regasm AssemblyName.dll /tlb:AssemblyName.tlb
```

Einige FCL-Assemblies werden schon bei der Installation des .NET Frameworks als COM-Komponenten verfügbar registriert (z.B. *System.dll*).

Regasm.exe registriert nur solche Typen, die mit dem Meta-Attribut `<ComVisible(true)>` belegt sind.



1.1.6 Verfügbarkeit

Nicht alle FCL-Klassen werden von allen Betriebssystemen unterstützt. So unterstützen NT 4.0, Windows 98/Windows ME und Windows XP Home nicht die Klassen des Namespaces *System.Web*, da diese Plattformen ASP.NET nicht betreiben können. Die Namespaces *System.ServiceProcess* und *System.Diagnostics* werden nicht auf Windows 98/ME unterstützt, da diese Betriebssysteme keine NT-Dienste und keine Leistungsindikatoren (Performance Counter) kennen. Gleiches gilt für die Dateisystemüberwachungs-Klasse aus *System.IO*.

Betriebssysteme

Das .NET Framework – damit auch die FCL – läuft nicht auf Windows 95.



1.2 Typen in .NET

Dieses Kapitel behandelt ausgewählte Details des Common Type Systems (CTS), die zum Verständnis der FCL wichtig sind.

Typen

Ein Typ ist in der CLR eines der folgenden vier Konzepte:

- ▶ eine Klasse (*Class*) oder
- ▶ eine Schnittstelle (*Interface*) oder
- ▶ eine Struktur (*Structure*) oder
- ▶ ein Aufzählungstyp (*Enumeration*).

Neben der Vorstellung dieser vier Konzepte werden in diesem Kapitel auch die Namensgebung und die Hierarchisierung von Typen in .NET behandelt.



1.2.1 Klassen

Eine Klasse (VB-Schlüsselwort `Class`) kann folgende öffentlichen Members enthalten:

- Attribute** ► Attribute
- Attribute sind Daten-Mitglieder einer Klasse. Es gibt in .NET zwei Arten von Attributen: Felder (engl. Fields) und Eigenschaften (engl. Properties).
- Zur der in diesem Buch verwendeten Definition des Begriffs »Attribut« schauen Sie bitte im Anhang A bei den Begriffen »Attribut« und »Meta-Attribut« nach.
- Field** Fields sind Attribute, bei denen der Nutzer des Objekts direkt einen bestimmten Speicherplatz innerhalb des Objekts beschreibt. Er kann alle dem Datentyp des Feldes entsprechenden Werte in das Feld schreiben, ohne dass eine weitere Prüfung erfolgt.
- Property** Im Gegensatz dazu wird bei einer Property bei jedem Schreib- und Lesezugriff Programmcode ausgeführt, bei dem z.B. Prüfungen durchgeführt werden können. So kann der Gültigkeitsbereich auf beliebige Werte beschränkt werden. Zu einer Property gehört eine Getter-Methode (`Get()`) und/oder eine Setter-Methode (`Set()`). Wie und ob überhaupt die übergebenen Werte verarbeitet werden, ist in diesen Methoden zu implementieren. In der Regel gehört zu einem Property ein `private`, also nur innerhalb des Objekts zugängliches Feld.
- Indexer** Aus der Sicht des Nutzers der Klasse gibt es fast keinen Unterschied zwischen Fields und Properties. Eine Property kann jedoch Parameter besitzen und wird dann *Indexer* (*indiziertes Attribut*) genannt. Der Lesezugriff sieht dann aus wie der Aufruf einer Methode:
- ```
element = collection.item(0)
```
- Beim Schreibzugriff fällt allerdings schon auf, dass es sich nicht um eine Methode handelt, denn einer solchen könnte kein Wert zugewiesen werden:
- ```
collection.item(2) = "Guten Tag"
```
- Ein Indexer kann als ein Array von Attributen betrachtet werden.

► Methoden	Methode
Methoden sind in Klassen implementierte Unterprogramme. Eine Klasse enthält in der Regel eine Implementierung zu den Methoden. Eine Klasse kann aber auch einzelne <i>abstrakte Methoden</i> (auch <i>virtuelle Methoden</i> genannt) besitzen oder aber komplett abstrakt (auch <i>rein virtuelle Klasse</i> genannt) sein. Eine abstrakte Klasse kann nicht instanziiert werden.	
► Ereignisse	Ereignis
Nutzer eines Objekts können ihr Interesse an von der Klasse definierten Ereignissen bekunden, indem sie dem Objekt einen Zeiger auf ein Unterprogramm übergeben, das das Objekt beim Eintritt bestimmter Bedingungen aufruft.	
► Konstruktoren	Konstruktor
Ein Konstruktor ist eine Unterroutine, die bei der Instanziierung der Klasse aufgerufen wird. Konstruktoren können Parameter haben, die bei der Instanziierung anzugeben sind.	
► Destruktor	Destruktor
Jede Klasse kann einen Destruktor mit dem Namen <code>Finalize()</code> besitzen, der aufgerufen wird, bevor der Garbage Collector die Instanz aus dem Speicher entfernt.	
► Konstanten	Konstante
Eine Klasse kann öffentliche Konstanten enthalten, die wie Attribute behandelt werden – jedoch mit dem Unterschied, dass auf diese Mitglieder nur lesend zugegriffen werden kann.	
► Aufzählungstypen (Konstantenlisten, Enumerationen)	Enumeration
Ein Aufzählungstyp ist eine Liste von symbolischen Konstanten. Jede einzelne symbolische Konstante der Liste repräsentiert einen Zahlwert. Zeichenketten oder andere Datentypen sind nicht erlaubt.	
► Eingebettete Typen	Eingebettete Typen
Eine Klasse kann andere Klassendefinitionen enthalten, sogenannte eingebettete Typen (engl. Nested Type). Diese eingebettete Typen werden über den Namen des übergeordneten adressiert, sodass bei einem voll qualifizierten Typname nicht unterschieden werden kann, ob es sich um bei vorgelagerten Namen um eine Namespace oder einen übergeordneten Typ handelt. Der Unterschied besteht darin, dass ein Namespace im Gegensatz zu einem übergeordneten Typ auf keinen Fall Mitglieder enthält und nicht instanziiert werden kann.	



Ein Modul (Schlüsselwort *Module*) in der Sprache Visual Basic .NET ist eine .NET-Klasse, bei der automatisch alle Mitglieder die Zugriffsart *Shared* haben.

Vererbung

Einfachvererbung

.NET unterstützt Einfachvererbung. Jede Klasse kann von höchstens einer anderen Klasse erben. Wird keine Vererbung explizit definiert, erbt die Klasse implizit von *System.Object*. Aus diesem Grund ist die Klasse *System.Object* die Wurzel jeder Vererbungshierarchie und alle Klassen in .NET besitzen die von *System.Object* definierten Mitglieder. Mehrfachvererbung wird nicht unterstützt.

```
Public Class KlassenName
    Inherits OberKlassenName

    ' --- Field-Attribute
    Public PublicAttribut As Byte
    Friend FriendAttribut As Byte
    Private PrivateAttribut As Byte
    Shadows ShadowAttribut As Byte
    Shared SharedAttribut As Byte
    ' --- Property-Attribute
    Public Property PropertyAttribut() As Byte
        Get
        End Get
        Set(ByVal Value As Byte)
        End Set
    End Property
    ' --- Methoden
    Public Sub Methode()
    End Sub
    Public Function Methode_mit_Rueckgabewert() _
        As Byte
    End Function
    ' --- Events
    Event Ereignis()
    ' --- Konstanten
    Public Const Konstante = 10
    Public Enum KonstantenListe
        k1 = 1
        k2 = 2
        k3 = 3
    End Enum
End Class
```

Listing 1.1: Beispiel für eine Klasse in VB.NET

Schnittstellen

Eine Klasse kann optional eine oder mehrere Schnittstellen implementieren.

1.2.2 Schnittstelle (Interface)

Eine Schnittstelle (VB-Schlüsselwort `Interface`) ist eine Beschreibung von Attributen, Methoden und Ereignissen. Der Unterschied zu einer Klasse besteht darin, dass die Methoden keine Implementierung enthalten. Eine Schnittstelle entspricht einer abstrakten Klasse.

Mitglieder ohne Implementierung

Eine Schnittstelle repräsentiert die Tür zu der Funktionalität eines Objekts und ist eine Untermenge seiner Funktionalität. Eine Schnittstelle ist als Vertrag zwischen einem Objekt und seinem Benutzer zu interpretieren, durch den das Objekt zusichert, auf einen bestimmten Satz von Nachrichten (also Methodenaufrufen) reagieren zu können.



Eine Schnittstelle kann nicht instanziiert, sondern nur im Rahmen einer Klasse verwendet werden. Dort muss die Schnittstelle implementiert werden, genauer gesagt, dort müssen **alle** Attribute und Ereignisse deklariert und alle Methoden deklariert und implementiert werden.

Eine Schnittstelle kann von genau einer anderen Schnittstelle erben. Eine Klasse kann eine oder mehrere Schnittstellen implementieren. Eine .NET-Klasse ist im Gegensatz zu einer COM-Klasse nicht verpflichtet, eine Schnittstelle zu implementieren.

Schnittstellen können nur als Attribute nur in Form von Properties, nicht aber Fields enthalten. Schnittstellen können keine Konstanten oder öffentlichen Aufzählungstypen enthalten.

```
Public Interface SchnittstellenName
    ' --- Property-Attribute
    Property PropertyAttribut() As Byte
    ' --- Methoden
    Sub Methode()
    Function Methode_mit_Rueckgabewert() As Byte
    ' --- Events
    Event Ereignis()
End Interface
```

Listing 1.2: Beispiel für eine Schnittstelle in VB.NET

1.2.3 Strukturen (Werteklassen)

Structure Eine Struktur (VB-Schlüsselwort: `Structure`, in C#: `struct`) hat große Ähnlichkeit mit einer Klasse. Unterschiede zwischen einer Klasse und einer Struktur sind:

- ▶ Eine Struktur erbt nicht direkt von `System.Object`, sondern automatisch von `System.ValueType`. `System.ValueType` erbt von `System.Object`.
- ▶ Die Instanz einer Struktur wird auf dem Stack gespeichert, während die Instanz einer Klasse auf dem Heap gespeichert wird.
- ▶ Strukturen können nicht explizit erben, d.h. eine Struktur erbt immer nur von `System.ValueType`.
- ▶ Von Strukturen kann nicht geerbt werden.
- ▶ Es kann keinen benutzerdefinierten parameterlosen Konstruktor geben, da dieser schon implizit definiert ist.
- ▶ Alle Mitglieder einer Struktur sind im Standard öffentlich (`Public`).
- ▶ Eine Struktur muss nicht (mit dem `new`-Schlüsselwort) instanziiert werden. Zur Erzeugung einer Instanz einer Struktur reicht die Deklaration einer Variablen dieses Typs.
- ▶ Strukturen können keine Ereigniskonsumenten sein (wohl aber Ereignisse aussenden).

```
Public Structure StrukturName
    ' --- Field-Attribute
    Public PublicAttribut As Byte
    Friend FriendAttribut As Byte
    Private PrivateAttribut As Byte
    Shadows ShadowAttribut As Byte
    Shared SharedAttribut As Byte
    ' --- Property-Attribute
    Public Property PropertyAttribut() As Byte
        Get
        End Get
        Set(ByVal Value As Byte)
        End Set
    End Property
    ' --- Methoden
    Public Sub Methode()
    End Sub
    Public Function Methode_mit_Rueckgabewert() _
```

```

        As Byte
    End Function
' --- Events
Event Ereignis()
' --- Konstanten
Public Const Konstante = 10
Public Enum KonstantenListe
    k1 = 1
    k2 = 2
    k3 = 3
End Enum
End Structure

```

Listing 1.3: Beispiel für eine Struktur in VB.NET

1.2.4 Aufzählungstypen (Enumerationen)

Ein Aufzählungstyp (Enumeration) ist eine Liste von Konstanten. Jede einzelne symbolische Konstante der Liste repräsentiert einen Ganzzahlwert. Zeichenketten oder andere Datentypen sind nicht erlaubt.

Konstantenliste

Ein Aufzählungstyp kann Teil einer Klasse sein oder eigenständig existieren.

```

Public Enum Farben
    rot
    gruen
    gelb
End Enum

```

Listing 1.4: Beispiel für eine Enumeration in VB.NET

Beim obigen Aufzählungstyp erhält die Konstante `rot` den Wert 0 und die folgenden Konstanten den jeweils um eins erhöhten Wert. Die Konstante `gelb` ist also in diesem Beispiel 2. Es ist auch möglich, explizit Werte festzulegen. Die Werte dürfen nur Ganzzahlen (Byte, Short, Integer oder Long) sein.

Explizite Werte

```

Namespace DE.ITVisions.FCL_Buch.Kapitel1
    Enum Farben As Long
        blau = 5
        gelb
        gruen = 6
    End Enum
End Namespace

```

Listing 1.5: Beispiel für eine Enumeration in VB.NET mit expliziten Werten

Die Konstante `gruen`, für die kein Wert explizit angegeben wurde, erhält automatisch den Wert 11.

Der voll qualifizierte Name einer Konstanten besteht aus folgenden Teilen:

```
Namespace.Unternamespace.Konstantenliste.Konstantenname
```

also z.B.

```
DE.ITVisions.FCL_Buch.Kapitel1.Farben.gruen
```

1.2.5 Typnamen und Namespaces

Namespace Typen werden in .NET nicht mehr durch GUIDs, sondern durch Zeichenketten eindeutig benannt. Diese Zeichenketten sind hierarchische Namen. Ein Name besteht aus dem relativen Namen des Typs und dem Namensraum (Namespace). Der relative Name muss nur innerhalb eines Namespaces eindeutig sein. Über alle Namespaces hinweg kann der Typname mehrfach vorkommen, denn der Namespace ermöglicht im Zweifel die Unterscheidung. Dies ist vergleichbar mit Dateien und Ordnern in einem Dateisystem.

Hierarchien Ein *Namespace* ist eine Gruppierung von Typen, die in Assemblies implementiert sind. Ein Namespace kann auch Unter-Namespaces enthalten, so dass eine Hierarchie entsteht. Ein Namespace kann einen Alias haben. Die Gruppierung, also die Auswahl der Typen, die zu einem Namespace gehören, sollte nach logischen oder funktionellen Prinzipien erfolgen. Im Gegensatz dazu sollte die Zusammenfassung von Typen zu einer Assembly gemäß den Bedürfnissen zur Verbreitung der Klassen (Deployment) erfolgen.

Im .NET Framework kann es beliebig viele Namespace-Hierarchien parallel geben. Es gibt keinen gemeinsamen Wurzel-Namespace. Die .NET Framework Class Library (FCL) besitzt zwei Wurzel-Namespaces, *System* und *Microsoft*.

Ein Namespace kann in beliebig vielen Assemblies implementiert werden, ebenso wie jede Assembly Typen zu beliebig vielen verschiedenen Namespaces beisteuern kann.



Ein Durchlaufen aller Namespaces auf einem System ist nicht ohne weiteres möglich, weil es kein globales Verzeichnis aller Namespaces gibt. Dies würde eine Registrierung von Komponenten voraussetzen und daher dem Gedanken des XCOPY-Deployment widersprechen. Möglich wäre aber die Suche nach .dll-/exe-Dateien im Dateisystem und eine Einzelprüfung dieser DLLs darauf, ob sie Typen enthalten.

Vergabe der Namespace-Namen

Da es keine zentrale Stelle gibt, die die Namespace-Namen vergibt, besteht natürlich grundsätzlich die Gefahr, dass es zu doppelten Typnamen kommt. Im Rahmen des CLI-Standards (siehe Kapitel 1.7) wurde deshalb folgende Notation festgelegt [CD:/Weitere Informationen/ECMA-Dokumente/Partition V Annexes.doc, Kapitel D.1.5]:

Firmenname.Technologiename

Beispiele:

Microsoft.Office

PowerSoft.PowerBuilder

Corel.CorelDraw

ITObjects.AMS

Es ist legitim, den Internet-Domännennamen in umgekehrter Reihenfolge zu verwenden, also z.B. COM.Microsoft.Office oder DE.ITObjects.AMS.

Ein Angreifer könnte eine DLL auch einfach durch eine gleichnamige DLL mit gleicher Versionsnummer und mit den gleichen implementierten Typen austauschen (diese Gefahr bestand auch in COM!). In .NET unterbindet ein auf einem Public Key-Verfahren basierender Strong Name für eine Assembly einen solchen Missbrauch. Zwar kann es eine DLL mit gleichem Namen, gleicher Versionsnummer und gleichen Typen geben, wenn sich die Assembly-Referenz aber auf einen Strong Name bezieht, wird .NET sich weigern, die falsche DLL zu laden.

Eindeutigkeit



**Schutz durch
Strong Names**

Vergabe der Typnamen

Auch für die Namensgebung von Typen in der .NET-Klassenbibliothek gibt es Regeln, die im CLI-Standard manifestiert sind [CD:/Weitere Informationen/ECMA-Dokumente/Partition V Annexes.doc, Kapitel D.1].

Die Namen für Klassen, Schnittstellen und Attribute sollen Substantive sein. Die Namen für Methoden und Ereignisse sollen Verben sein.

Für die Groß-/Kleinschreibung gilt grundsätzlich *PascalCasing*, d.h. ein Bezeichner beginnt grundsätzlich mit einem Großbuchstaben und jedes weitere Wort innerhalb des Bezeichners beginnt ebenfalls wieder mit einem Großbuchstaben. Ausnahmen gibt es lediglich für Abkürzungen, die nur aus zwei Buchstaben bestehen.

**Namespace und
Assembly**

Wortwahl

**Groß-/Klein-
schreibung**

Diese dürfen komplett in Großbuchstaben geschrieben sein (z.B. UI und IO). Alle anderen Abkürzungen werden entgegen ihrer normalen Schreibweise in Groß-/Kleinschreibung geschrieben (z.B. Xml, Xsd und W3c). Attribute, die geschützt (Schlüsselwort `Protected`) sind, und die Namen von Parametern sollen in *camelCasing* (Bezeichner beginnt mit einem Kleinbuchstaben, aber jedes weitere Wort innerhalb des Bezeichners beginnt mit einem Großbuchstaben) geschrieben werden.

1.2.6 Typhierarchien

Das .NET Framework unterstützt bezüglich Typen drei Hierarchiekonzepte:

Namespaces

1. Namespace-Hierarchie (Namensraumhierarchie)

Typen haben hierarchische Namen, was durch die Organisation von Typen in Namespaces erreicht wird. Die Namespace-Hierarchie ist ein Baum ohne Zyklen.



Durch die Voranstellung einer Internetdomain in umgekehrter Reihenfolge vor den Namen kann man sehr leicht weltweit eindeutige Namen erzeugen, z.B. DE.ITVisions.FCL-Buch.Farben.gruen

Vererbungshierarchie

2. Vererbungshierarchie

Klassen und Schnittstellen (nicht aber die anderen beiden Arten von Typen) können voneinander erben, allerdings ist immer höchstens eine explizite Vererbungsbeziehung möglich. Die Vererbungshierarchie ist daher ein Baum ohne Zyklen. Die Klasse `System.Object` ist dabei immer die oberste Klasse.

Objektmodelle

3. Objektmodell (Objekthierarchie, Containment-Hierarchie)

Instanzen von Typen können (müssen aber nicht!) zur Laufzeit in Beziehung stehen. Daraus ergibt sich eine *Objekthierarchie*, die *Objektmodell* genannt wird. Da die Objektbeziehungen beliebig sind, ergibt sich ein Graph, der möglicherweise auch Zyklen enthält.

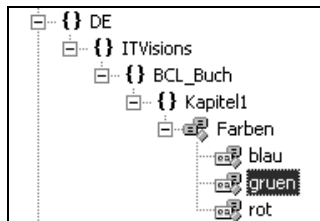
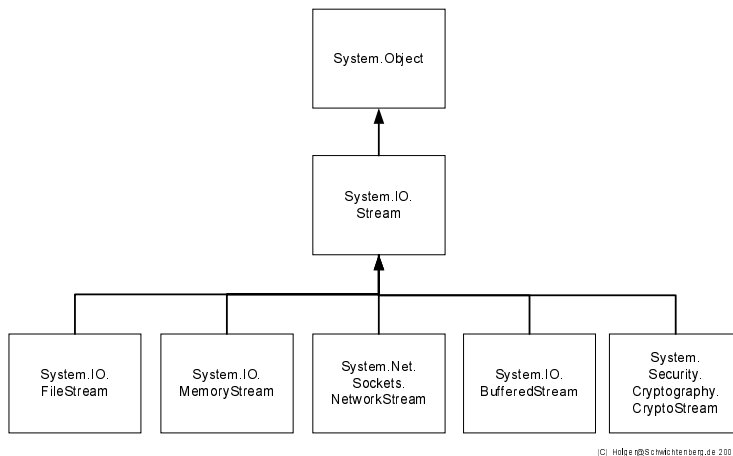


Abbildung 1.3: Beispiel für eine Namespace-Hierarchie



(C) HolgerSchneitzberg.de 2002

Abbildung 1.4: Beispiel für eine Vererbungshierarchie

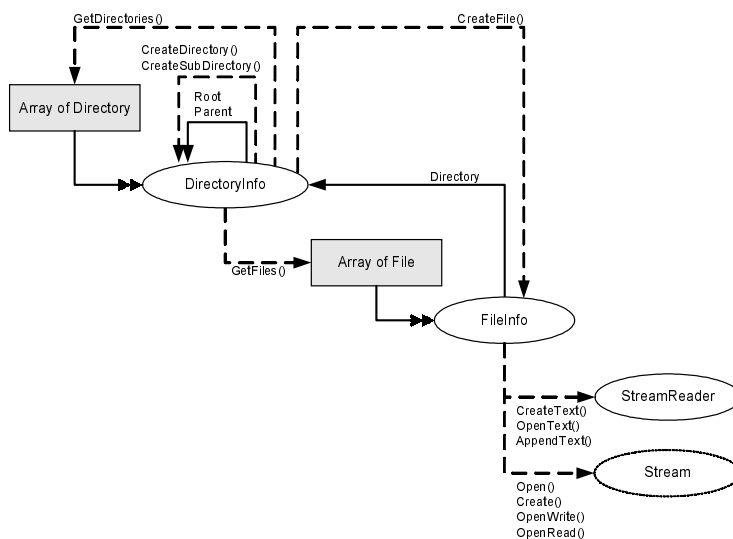


Abbildung 1.5: Beispiel für ein Objektmodell

In diesem Buch werden die Objektmodelle und – bei Bedarf – auch die Vererbungshierarchien grafisch dargestellt. Hinweise zur verwendeten Notation finden Sie in Anhang D.



I.3 Verwendung von FCL-Typen

Dieses Unterkapitel stellt die Möglichkeiten der Verwendung von FCL-Typen (insbesondere Klassen) vor. Die Ausführungen gelten auch analog für Strukturen.

I.3.1 Assembly-Referenz

Assembly-Referenz

Grundvoraussetzung für die Verwendung eines Typs ist, dass die Assembly, die diesen Typ implementiert, beim Kompilieren hinzugefügt wird. Dadurch entsteht eine Referenz von einer Assembly zu einer anderen. Die Entwicklungsumgebung Visual Studio .NET unterstützt die Referenzierung einer Assembly schon bei der Kodierung, so dass zahlreiche Eingabeunterstützungen zur Verfügung stehen.

I.3.2 Importieren von Namespaces

Namen bekannt machen

Um einen Typ verwenden zu können, muss im Normalfall der gesamte Name, inklusive Namespace, verwendet werden. Die .NET-Sprachen unterstützen jedoch das Einbinden eines Namespaces, so dass die Namen verkürzt im Quellcode verwendet werden können.

Ohne Namespace-Import	Mit Namespace-Import
<pre>If LogoFarbe = DE.ITVisions. FCL_Buch.Kapitel1.Farben. rot Then System.Console.WriteLine("SPD") ElseIf LogoFarbe = DE.ITVisi- ons.FCL_Buch.Kapitel1.Farben.schwarz Then System.Console.WriteLine("CDU") ElseIf LogoFarbe = DE.ITVisi- ons.FCL_Buch.Kapitel1.Farben.gruen Then System.Console.WriteLine("Die Grünen") ElseIf LogoFarbe = DE.ITVisi- ons.FCL_Buch.Kapitel1.Farben.gelb Then System.Console.WriteLine("FDP") End If</pre>	<pre>Imports DE.ITVisions.FCL_Buch.Kapitel1.Farben Imports System If LogoFarbe = rot Then Console.WriteLine("SPD") ElseIf LogoFarbe = schwarz Then Console.WriteLine("CDU") ElseIf LogoFarbe = gruen Then Console.WriteLine("Die Grünen") ElseIf LogoFarbe = gelb Then Console.WriteLine("FDP") End If</pre>

Tabelle I.2: Vorteile des Namespace-Imports

Wenn ein Typ verwendet wird, sucht der Compiler in allen eingebundenen Namespaces danach. Der Compiler beschwert sich, wenn er den Typ dort nirgendwo finden kann oder der Typ innerhalb der eingebundenen Namespaces doppelt vorkommt. In diesem Fall hilft entweder die Beseitigung der Doppeldeutigkeit durch Entfernen eines Namespaces oder aber die Verwendung des vollen Namens bei jedem Gebrauch im Quellcode.

Sprache	Konstrukt zur Einbindung von Namespaces
Visual Basic .NET	Imports System.Drawing
C#	using System.Drawing
JScript.NET	import System.Drawing

Tabelle 1.3: Einbindung von Namespaces

Der Import eines Namespaces erzeugt nicht automatisch eine Assembly-Referenz. Vielmehr kann ein Import nur erfolgreich sein, wenn eine Referenz besteht.



Direkter Import von Typen

In Visual Basic .NET ist es auch möglich, einen einzelnen Typ mit der Imports-Anweisung einzubinden. In diesem Fall stehen die Shared Members dieses Typs wie eingebaute Funktionen zur Verfügung.

Import des Namespaces	Direkter Import des Typs
Imports System Console.WriteLine("Hallo Holger!")	Imports System.Console WriteLine("Hallo Holger")

Tabelle 1.4: Direkter Import von Datentypen

Namespace-Aliase

Imports und using unterstützen auch die Vergabe eines Alias für einen Namespace, so dass bei gleichen Typnamen die Fingerkuppen des Entwicklers geschont bleiben.

**Anderen Namen
vergeben**

```
Imports VB6 = _
    microsoft.VisualBasic.Compatibility.VB6

Public Class namespaces
    Dim o As New VB6.LabelArray()
End Class
```

Listing 1.6: Beispiel für einen Namespace-Alias in VB.NET

I.3.3 Instanziierung

Eine FCL-Klasse kann instanziiert werden, wenn sie folgende Bedingungen erfüllt:

- ▶ Die Klasse ist nicht abstrakt und
- ▶ die Klasse besitzt mindestens einen öffentlichen Konstruktor.

Abstrakte Klassen Eine abstrakte Klasse ist mit dem Schlüsselwort `MustInherit` (VB.NET) bzw. `abstract` (C#) belegt.

Konstruktoren Die meisten Klassen besitzen einen öffentlichen Konstruktor, da sie einen parameterlosen Konstruktor von `System.Object` erben. Sofern dieser Konstruktor nicht mit einem privaten Konstruktor überschrieben wird, kann eine Instanziierung in der Form

```
Dim o As New Klassenname
```

erfolgen. Einige Klassen besitzen nur Konstruktoren mit Parametern. Dann müssen die Parameter bei der Instanziierung mit angegeben werden.

```
Dim o As New Klassenname = New Klassenname( _  
    param1, param2, ...)
```

Klassen ohne Konstruktor Manche Klassen in der FCL (z.B. `System.IO.File`) besitzen keinen öffentlichen Konstruktor. Diese Klassen können also nicht instanziiert werden. Sie können auf drei Weisen verwendet werden:

- ▶ Ein Objekt einer anderen Klasse liefert einen Zeiger auf ein Objekt der konstruktorlosen Klasse.
- ▶ Die konstruktorlose Klasse besitzt `Shared Members` (Statische Member), die direkt über den Klassennamen aufgerufen werden können.
- ▶ Einige dieser Klassen (z.B. `System.Drawing.Image` oder `System.Web.WebRequest`) besitzen eine `Shared Members-Methode`, die eine Instanz der Klasse erzeugt.

I.3.4 Vererbung von Klassen

NotInheritable (sealed) Von jeder .NET-Klasse (einschließlich aller FCL-Klassen) kann geerbt werden, sofern die Klasse nicht mit dem Schlüsselwort `NotInheritable` (VB.NET) bzw. `sealed` (C#) belegt ist.

1.3.5 Sichtbarkeit und Zugriff

Bei der Klassendeklaration können durch verschiedene Schlüsselwörter die Sichtbarkeit und die Zugriffsart eines Klassenmitglieds bestimmt werden. Für die Arbeit mit der FCL ist es wichtig zu verstehen, welche Auswirkungen die verschiedenen Eigenschaften von Klassenmitgliedern auf die Verwendung einer Klasse haben.

Ein als `Public` deklariertes Mitglied kann über eine Instanz verwendet werden. Ein `Private`-Klassenmitglied kann niemals von einem Objektnutzer, sondern immer nur von internen Routinen der Klasse selbst genutzt werden. Ein Klassenmitglied, das als geschützt (`Protected`) deklariert wurde, kann nicht über eine Instanz dieser Klasse, wohl aber von einer erbenden Klasse verwendet werden.

**Public, Private,
Protected**

Ein `Shared Member` (Schlüsselwort `Shared` in VB.NET bzw. `static` in C#) gehört nicht zu einer Instanz, sondern direkt zur Klasse und kann daher auch über den Klassennamen aufgerufen werden, ohne dass zuvor eine Instanz der Klasse erzeugt werden musste. Ein `Shared Member` kann aber auch von einer Instanz der jeweiligen Klasse aus aufgerufen werden. Eine Klasse kann nur aus `Shared Members` bestehen und agiert dann wie ein `Intrinsic Object`, da es nur eine Instanz gibt (genau genommen kann man mehrere Instanzen erzeugen, diese sind jedoch alle völlig identisch).

Shared (Static)

	Public	Private	Protected	Shared (static)
Aufruf über Instanz der Klasse	Ja	Nein	Nein	Ja
Aufruf über Klassennamen	Nein	Nein	Nein	Ja
Aufruf aus Instanz einer erbenden Klasse	Ja	Nein	Ja	Ja
Aufruf über Namen der Unterklasse	Nein	Nein	Nein	Ja

Tabelle 1.5: Aufrufmöglichkeiten bei verschiedenen Mitgliedsarten

Ein Modul (Schlüsselwort `Module`) in der Sprache Visual Basic .NET ist eine .NET-Klasse, bei der automatisch alle Mitglieder die Zugriffsart `Shared` haben.

I.4 Überblick über die FCL-Namespaces

- Namespaces
- Die FCL ist – wie in .NET üblich – hierarchisch in Namespaces gegliedert. Sie hat aber nicht eine, sondern zwei Wurzeln:
- `System` (hier sind die weitaus meisten Klassen untergeordnet) und
 - `Microsoft` (hier befinden sich im Wesentlichen nur Klassen für die .NET-Compiler, die Windows-Registry und Systemereignisse)

Einige Teile der FCL haben aber weiterhin einen eigenständigen, »umgangssprachlichen« Namen. Das beste Beispiel dafür ist *ADO.NET* (auch: *ADO+*). *ADO.NET* sind Klassen für den Zugriff auf Datenbanken als Nachfolger der COM-Komponente *ActiveX Data Objects (ADO)*. Ein weiteres Beispiel ist der Namespace *System.Web*, der auch als *Active Server Pages .NET (ASP.NET)* bekannt ist.

Die FCL im Vergleich zu verfügbaren COM-Komponenten

Die folgende Tabelle gibt einen Überblick über die wichtigsten Namespaces der FCL und setzt diese bezüglich ihrer Funktionalität in Beziehung zu bestehenden COM-Komponenten.

Namespace	Beschreibung	Beispielklassen	Ähnliche klassische COM-Komponente
Microsoft.CSharp	C#-Compiler	Compiler	RegCol
Microsoft.JScript	JScript.NET-Compiler	MathObject ScriptObject	
Microsoft.VisualBasic	Visual Basic .NET-Compiler	Collection Conversion	
Microsoft.VSA	Klassen für Visual Studio for Applications (VSA)	IVsaEngine IVsaSite	
Microsoft.Win32	Zugriff auf die Registry und Systemereignisse (z. B. Herunterfahren des Rechners, Stromsparmodusänderung)	Registry RegistryKey SystemEvents	

Tabelle I.6: Die wichtigsten Namespaces der FCL

Namespace	Beschreibung	Beispielklassen	Ähnliche klassische COM-Komponente
System	Elementare Datentypen, DOS-Fenster, Garbage Collector, Application Domain	Object Buffer Byte Char Array Int32 Exception GC Console String	Laufzeitumgebung der jeweiligen Sprache
System.Collections	Objektmengen	ArrayList BitArray Dictionary HashTable Queue SortedList Stack	Scripting Runtime Library
System.Data	Zugriff auf Datenquellen aller Art (ActiveX Data Objects.NET)	DataRelation DataRow DataSet DataTable DataSource	ADO
System.Configuration	Zur Verwaltung von AppDomains und zum Zugriff auf Assembly-Konfigurationsdaten und globale Konfigurationsdaten	ConfigManager BaseConfigItem AppDomain	
System.EnterpriseServices (vor Beta2: Microsoft.ComServices)	Zugriff auf COM+-Dienste	SecurityIdentity Transaction Attribute	COM+ Administration Objects
System.DirectoryService	Zugriff auf Verzeichnisdienste	DirectoryEntry DirectoryEntries PropertyCollection SearchResults	ADSI
System.Drawing	Funktionen des Windows Graphics Device Interface (GDI)	Bitmap Brush Cursor Font Image Pen Point Rectangle	

Tabelle 1.6: Die wichtigsten Namespaces der FCL (Fortsetzung)

Namespace	Beschreibung	Beispielklassen	Ähnliche klassische COM-Komponente
System. Globalization	Zugriff auf Länder-einstellungen (Kalender, Datums-format, Zahlenfor-mat etc.)	CultureInfo Calendar DaylightTime KoreanCalendar	WMI, WSHRuntime, STMAAdmin, u.a.
System. Diagnostics	Debugging, Tracing, Ereignisprotokoll, Performance-Counter	CounterSample EventLog EventLogEntry Debugger Process Trace	
System.IO	Dateisystemzugriff und Dateizugriff	Directory File FileStream MemoryStream Path StreamReader StreamWriter	
System. Management	Netz- und System-management mit WMI	ManagementObject EventQuery MethodData MethodSet	WMI
System. Messaging	Steuerung des Microsoft Message Queue Service (MSMQ)	Message MessageQueue XmlMessageFormatter	MSMQ-Objekte
System.Net	Zugriff auf Netzwerkprotokolle (TCP, UDP, HTTP, DNS, etc.)	HTTPWebRequest HTTPWebResponse DNS IPAddress Cookie TCPClient TCPListener	TypLib-Informationskomponente
System. Reflection	Zugriff auf Metadaten von .NET-Komponenten	Assembly ConstructorInfo FieldInfo MemberInfo MethodInfo Module ParameterInfo PropertyInfo EventInfo	
System. Runtime. InteropServices	COM-Interoperabilität	COMException RegistrationServices	

Tabelle 1.6: Die wichtigsten Namespaces der FCL (Fortsetzung)

Namespace	Beschreibung	Beispielklassen	Ähnliche klassische COM-Komponente
System. Runtime. Remoting	Entfernte Nutzung von entfernten Komponenten (Remoting)	ChannelServices RemotingServices IMessage IMessageSink	Remote Data Service (RDS)
System. Runtime. Serialization	Serialisierung von Objekten	Formatter Serialization Info	ADsSecurity
System. Security	Sicherheitseinstellungen für Komponenten und Objekte des Betriebssystems	Permissions Policy Principal Util Cryptography FileIOPermission X509Certificate DES SecurityElement	
System. ServiceProcess	Kontrolle über NT-Dienste	ServiceBase Service Controller ServiceInstaller	
System.Text	Zeichenkettenfunktionen, Textkodierung (z.B. UTF-7 und UTF-8)	Encoder Decoder StringBuilder	RegExp
System.Text. RegularExpressions	Reguläre Ausdrücke (Musterprüfung)	Capture Match MatchCollection	
System. Threading	Multi-Threading-Programmierung	Thread Mutex ThreadPool	
System. Timers	Zeitgesteuerte Ereignisse	Timer	Active Server Pages-Objekte
System.Web	Kommunikation zwischen Browser und Webserver (Objekte für ASP.NET)	HttpApplication HttpCookie HttpRequest HttpResponse	

Tabelle 1.6: Die wichtigsten Namespaces der FCL (Fortsetzung)

Namespace	Beschreibung	Beispielklassen	Ähnliche klassische COM-Komponente
System.Web.UI	Steuerelemente für die Gestaltung von Webseiten (serverseitige Verarbeitung in ASP.NET)	AdRotator BorderStyle DataGrid HyperLink ListBox Panel Radiobutton Table Page CheckBox Button	MSXML
System.Windows.Forms (vor Beta2 hieß dieser Namespace System.WinForms)	Steuerelemente für die Gestaltung von Windows-Desktop-Anwendungen	Button CheckBox DataGrid FileDialog Form ListBox MainMenu MonthCalendar NewFontDialog RichEdit ToolBarTreeView	
System.Xml	Zugriff auf das Document Object Model der Extensible Markup Language	XmlDocument XmlAttribute XmlElement	
System.XML.XPath	Einsatz der XPath-Sprache	XPathDocument XPathNavigator	
System.XML.Xsl	Ausführung von Extensible Stylesheet Language-Transformationen	XslTransform XsltException	
System.Xml.Serialization	Serialisierung von Objekten in XML-Daten	SoapAttributes XmlSerializer	

Tabelle 1.6: Die wichtigsten Namespaces der FCL (Fortsetzung)

Allgemeine vs. sprachspezifische Namespaces

Microsoft Visual
Basic .NET
Runtime

In Bezug auf den Zugriff auf Windows soll die FCL alle sprachspezifischen Bibliotheken ersetzen. Natürlich kann jede .NET-Sprache weiterhin ihre eigene Bibliothek besitzen. Im Fall von Visual Basic ist das auch so.

Die Datei *Microsoft.VisualBasic.dll* enthält eine Assembly mit Namen *Microsoft Visual Basic .NET Runtime*, die neben dem VB.NET-Compiler auch zahlreiche Klassen mit Methoden beinhaltet, die aus VB.NET-Anwendungen heraus nutzbar sind. Hier sind – aus Kompatibilitätsgründen – die aus VB 6.0 bekannten Funktionen bereitgestellt, z.B. `Trim()`. Die Methoden sind als *Shared Members* implementiert und stehen daher innerhalb von VB.NET wie nicht-objektorientierte Funktionen zur Verfügung.

Die FCL bietet aber für diese Funktionen jeweils eine alternative Aufrufmöglichkeit. `Microsoft.VisualBasic.Strings.Trim()` entspricht `System.String.Trim()`. Ob Sie die eine oder andere Aufrufart verwenden, macht hinsichtlich der Performance keinen Unterschied.

Alternativen

Es ist (für neu entwickelten Code) empfehlenswert, den offiziellen Weg für die sprachneutralen Namespaces zu gehen, selbst wenn damit ein erhöhter Lernaufwand verbunden ist. Dafür ist anschließend der Aufwand für das Erlernen einer anderen Sprache bzw. die Portierung in eine andere Sprache wesentlich geringer.



1.5 Implementierung der FCL

Die FCL ist in Form mehrerer .NET-Komponenten (Assemblies) implementiert und läuft daher als *Managed Code*. Als Implementierungssprachen hat Microsoft C++ und C# verwendet. Die Assemblies sind *Shared Assemblies*, d.h. im *Global Assembly Cache* (GAC) gespeichert. Sie sind daher von allen .NET-Anwendungen aus zugänglich. Nicht jeder Namespace ist in einer eigenen Assembly; es gibt einige wenige Assemblies, die den Großteil der FCL-Typen enthalten. Andersherum gibt es auch Namespaces, die auf mehrere Assemblies aufgeteilt sind (z.B. `System`, `System.IO` und `System.Xml`).

FCL-Assemblies

Liste der FCL-Assemblies

Die folgende Tabelle gibt einen Überblick über die wichtigsten Assemblies der FCL. In der Tabelle sind alle Assemblies berücksichtigt, die nach der Installation von Visual Studio .NET im .NET Framework-Verzeichnis (`%Windows%\Microsoft.NET\Framework\v1.0.3705`) liegen und aus diesen Assemblies alle Typen, die mit dem Top-Level-Namespace `System` oder `Microsoft` beginnen.

Legende zu der Tabelle:

Spalte A: Anzahl der enthaltenen Typen (öffentliche + nicht öffentliche)

Spalte B: Anzahl der öffentlichen Typen

Spalte C: Anzahl der öffentlichen Klassen

Spalte D: Anzahl der öffentlichen Schnittstellen

Spalte E: Anzahl der öffentlichen Aufzählungstypen (Enumerationen)

Spalte F: Anzahl der öffentlichen Strukturen

Name der Assembly	Assembly-Datei	A	B	C	D	E	F
CustomMarshalers	CustomMarshalers.dll	22	4	4	0	0	0
ISymWrapper	ISymWrapper.dll	8	8	8	0	0	0
mscorlib	mscorlib.dll	1335	863	566	124	108	64
System.Configuration. Install	System.Configuration. Install.dll	22	14	12	1	1	0
System.Data	System.Data.dll	427	160	106	15	26	13
System.Design	System.Design.dll	454	102	89	10	3	0
System.Directory Services	System.Directory Services.dll	61	21	16	0	5	0
System	System.dll	746	451	349	51	48	3
System.Drawing.Design	System.Drawing. Design.dll	29	9	9	0	0	0
System.Drawing	System.Drawing.dll	246	168	98	3	59	8
System.Enterprise Services	System.Enterprise Services.dll	226	81	55	10	14	2
System.Management	System.Management.dll	230	66	55	2	9	0
System.Messaging	System.Messaging.dll	85	46	28	1	17	0
System.Runtime. Remoting	System.Runtime. Remoting.dll	186	26	25	0	1	0
System.Runtime.Seriali zation.Formatters.Soa p	System.Runtime.Seriali zation.Formatters. Soap.dll	49	1	1	0	0	0
System.Security	System.Security.dll	40	21	21	0	0	0
System.ServiceProcess	System.Service Process.dll	39	18	11	0	7	0
System.Web	System.Web.dll	674	319	252	20	45	2
System.Web.Regular Expressions	System.Web.Regular Expressions.dll	45	15	15	0	0	0
System.Web.Services	System.Web. Services.dll	270	143	133	0	10	0

Tabelle 1.7: Typen in der FCL, ausgewertet nach Assemblies

Name der Assembly	Assembly-Datei	A	B	C	D	E	F
System. Windows.Forms	System. Windows.Forms.dll	897	330	217	17	92	4
System.Xml	System.XML.dll	716	205	176	6	23	0
Summe		6807	3071	2246	260	468	96

Tabelle 1.7: Typen in der FCL, ausgewertet nach Assemblies (Fortsetzung)

Wenn Sie nachrechnen, werden Sie in der Assembly *mscorlib.dll* eine kleine Differenz feststellen: 568 Klassen, 124 Schnittstellen, 109 Aufzählungstypen und 64 Strukturen ergeben zusammen 865 öffentliche Typen statt der in der Tabelle genannten 866. Des Rätsels Lösung liegt im Typ *System.Enum*, der Oberklasse aller Aufzählungstypen. Diese öffentliche Klasse ist in den Metadaten nicht als solche markiert.

Die *mscorlib* hieß in Beta1 des .NET Frameworks noch *ComRuntimeLibrary*, was auf die ursprünglichen Pläne von Microsoft, eine einheitliche Laufzeitumgebung für COM zu schaffen, zurückzuführen ist.



Aufteilung der FCL-Namespaces auf die FCL-Assemblies

Die folgende Tabelle zeigt, in welchen Assemblies die FCL-Namespaces implementiert sind. Es gibt mehrere Namespaces, deren Implementierung sich über mehrere Assemblies erstreckt.

Die Liste ist alphabetisch nach den Namespaces sortiert. Berücksichtigt wurden nur Typen, die öffentlich und nicht als »*NotPublic*« deklariert waren.

Namespace	Assembly-Datei	Anzahl Typen
System	mscorlib.dll	136
System	System.dll	5
System.CodeDom	System.dll	77
System.CodeDom.Compiler	System.dll	17
System.Collections	mscorlib.dll	21
System.Collections.Specialized	System.dll	9
System.ComponentModel	System.dll	119
System.ComponentModel.Design	System.Design.dll	10
System.ComponentModel.Design	System.dll	51

Tabelle 1.8: Zusammenhang zwischen Namespaces und Assemblies

Namespace	Assembly-Datei	Anzahl Typen
System.ComponentModel. Design.Serialization	System.Design.dll	3
System.ComponentModel. Design.Serialization	System.dll	13
System.Configuration	System.dll	11
System.Configuration.Assemblies	mscorlib.dll	3
System.Configuration.Install	System.Configuration. Install.dll	12
System.Data	System.Data.dll	87
System.Data.Common	System.Data.dll	12
System.Data.OleDb	System.Data.dll	22
System.Data.SqlClient	System.Data.dll	20
System.Data.SqlTypes	System.Data.dll	18
System.Diagnostics	mscorlib.dll	7
System.Diagnostics	System.Configuration. Install.dll	2
System.Diagnostics	System.dll	53
System.Diagnostics.Design	System.Design.dll	4
System.Diagnostics.SymbolStore	ISymWrapper.dll	8
System.Diagnostics.SymbolStore	mscorlib.dll	14
System.DirectoryServices	System.DirectoryServices.dll	21
System.Drawing	System.Drawing.dll	47
System.Drawing.Design	System.Drawing.Design.dll	9
System.Drawing.Design	System.Drawing.dll	17
System.Drawing.Drawing2D	System.Drawing.dll	36
System.Drawing.Imaging	System.Drawing.dll	33
System.Drawing.Printing	System.Drawing.dll	29
System.Drawing.Text	System.Drawing.dll	6
System.EnterpriseServices	System.EnterpriseServices.dll	56
System.EnterpriseServices. CompensatingResourceManager	System.EnterpriseServices.dll	9
System.EnterpriseServices.Internal	System.EnterpriseServices.dll	16
System.Globalization	mscorlib.dll	26
System.IO	mscorlib.dll	29
System.IO	System.dll	12

Tabelle 1.8: Zusammenhang zwischen Namespaces und Assemblies (Fortsetzung)

Namespace	Assembly-Datei	Anzahl Typen
System.IO.IsolatedStorage	mscorlib.dll	6
System.Management	System.Management.dll	54
System.Management Instrumentation	System.Management.dll	12
System.Messaging	System.Messaging.dll	44
System.Messaging.Design	System.Design.dll	1
System.Messaging.Design	System.Messaging.dll	2
System.Net	System.dll	46
System.Net.Sockets	System.dll	17
System.Reflection	mscorlib.dll	63
System.Reflection.Emit	mscorlib.dll	34
System.Resources	mscorlib.dll	9
System.Resources	System.Windows.Forms.dll	4
System.Runtime.CompilerServices	mscorlib.dll	20
System.Runtime.InteropServices	mscorlib.dll	120
System.Runtime.InteropServices Services.CustomMarshalers	CustomMarshalers.dll	4
System.Runtime.InteropServices Services.Expando	mscorlib.dll	1
System.Runtime.Remoting	mscorlib.dll	19
System.Runtime.Remoting. Activation	mscorlib.dll	5
System.Runtime.Remoting. Channels	mscorlib.dll	28
System.Runtime.Remoting. Channels	System.Runtime.Remoting.dll	9
System.Runtime.Remoting. Channels.Http	System.Runtime.Remoting.dll	5
System.Runtime.Remoting. Channels.Tcp	System.Runtime.Remoting.dll	3
System.Runtime.Remoting. Contexts	mscorlib.dll	15
System.Runtime.Remoting.Lifetime	mscorlib.dll	5
System.Runtime.Remoting. Messaging	mscorlib.dll	24

Tabelle 1.8: Zusammenhang zwischen Namespaces und Assemblies (Fortsetzung)

Namespace	Assembly-Datei	Anzahl Typen
System.Runtime.Remoting.Metadata	mscorlib.dll	7
System.Runtime.Remoting.Metadata.W3cXsd2001	mscorlib.dll	32
System.Runtime.Remoting.MetadataServices	System.Runtime.Remoting.dll	7
System.Runtime.Remoting.Proxies	mscorlib.dll	2
System.Runtime.Remoting.Services	mscorlib.dll	3
System.Runtime.Remoting.Services	System.Runtime.Remoting.dll	2
System.Runtime.Serialization	mscorlib.dll	20
System.Runtime.Serialization.Formatters	mscorlib.dll	9
System.Runtime.Serialization.Formatters.Binary	mscorlib.dll	1
System.Runtime.Serialization.Formatters.Soap	System.Runtime.Serialization.Formatters.Soap.dll	1
System.Security	mscorlib.dll	18
System.Security.Cryptography	mscorlib.dll	65
System.Security.Cryptography.X509Certificates	mscorlib.dll	1
System.Security.Cryptography.X509Certificates	System.dll	1
System.Security.Cryptography.Xml	System.Security.dll	21
System.Security.Permissions	mscorlib.dll	46
System.Security.Permissions	System.dll	2
System.Security.Policy	mscorlib.dll	28
System.Security.Principal	mscorlib.dll	10
System.ServiceProcess	System.ServiceProcess.dll	16
System.ServiceProcess.Design	System.Design.dll	1
System.ServiceProcess.Design	System.ServiceProcess.dll	2
System.Text	mscorlib.dll	8
System.Text.RegularExpressions	System.dll	12
System.Threading	mscorlib.dll	28
System.Threading	System.dll	2
System.Timers	System.dll	4
System.Web	System.Web.dll	41

Tabelle 1.8: Zusammenhang zwischen Namespaces und Assemblies (Fortsetzung)

Namespace	Assembly-Datei	Anzahl Typen
System.Web.Caching	System.Web.dll	5
System.Web.Configuration	System.Web.dll	5
System.Web.Handlers	System.Web.dll	1
System.Web.Hosting	System.Web.dll	6
System.Web.Mail	System.Web.dll	6
System.Web.RegularExpressions	System.Web.RegularExpressions.dll	15
System.Web.Security	System.Web.dll	18
System.Web.Services	System.Web.Services.dll	4
System.Web.Services.Configuration	System.Web.Services.dll	3
System.Web.Services.Description	System.Web.Services.dll	69
System.Web.Services.Discovery	System.Web.Services.dll	21
System.Web.Services.Protocols	System.Web.Services.dll	46
System.Web.SessionState	System.Web.dll	7
System.Web.UI	System.Web.dll	72
System.Web.UI.Design	System.Design.dll	39
System.Web.UI.Design.WebControls	System.Design.dll	27
System.Web.UI.HtmlControls	System.Web.dll	22
System.Web.UI.WebControls	System.Web.dll	132
System.Web.Util	System.Web.dll	4
System.Windows.Forms	System.Windows.Forms.dll	313
System.Windows.Forms.ComponentModel.Com2Interop	System.Windows.Forms.dll	3
System.Windows.Forms.Design	System.Design.dll	17
System.Windows.Forms.Design	System.Windows.Forms.dll	7
System.Windows.Forms.PropertyGridInternal	System.Windows.Forms.dll	3
System.Xml	System.Data.dll	1
System.Xml	System.XML.dll	52
System.Xml.Schema	System.XML.dll	73
System.Xml.Serialization	System.XML.dll	61
System.Xml.XPath	System.XML.dll	12
System.Xml.Xsl	System.XML.dll	7
Summe		3071

Tabelle 1.8: Zusammenhang zwischen Namespaces und Assemblies (Fortsetzung)

I.6 Original-Dokumentation der FCL

SDK Zur Framework Class Library (FCL) hat Microsoft eine ausführliche Dokumentation erstellt, die Teil des .NET Framework SDKs ist. Das .NET Framework SDK kann kostenlos auf der Microsoft-MSDN-Download-Website [MSDC] heruntergeladen und lokal installiert werden. Die FCL-Dokumentation liegt in Form einer kompilierten HTML-Hilfe-Datei (Dateinamen *csref.HxS* und *csref.HxI*) vor und ist ca. 44 MB groß. Diese Hilfe-Datei wird auch mit Visual Studio .NET ausgeliefert.

MSDN Library Die FCL-Dokumentation ist auch Teil der Microsoft MSDN Library. Die komplette MSDN Library kann von Microsoft auf CD-ROM bzw. DVD abonniert oder kostenlos im Internet unter [MSLIB] eingesehen werden.



Zusammen mit der deutschen Version von Visual Studio .NET liefert Microsoft eine komplett ins Deutsche übersetzte Version der .NET Framework-Dokumentation aus.

I.6.1 Position der FCL-Dokumentation

Die folgende Abbildung zeigt die Position der FCL-Dokumentation im Rahmen der Visual Studio .NET-Dokumentation.

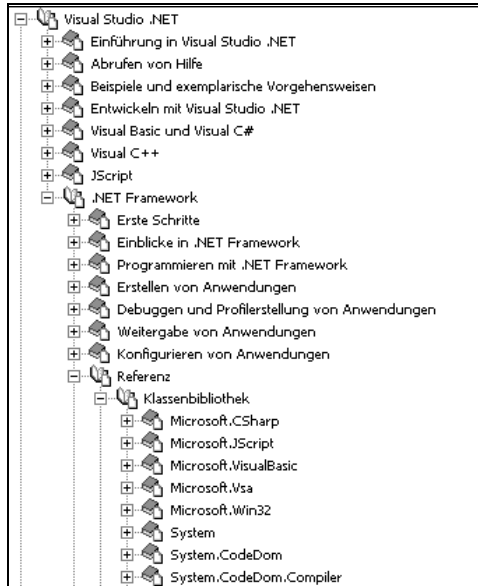


Abbildung I.6: Position der FCL-Dokumentation

1.6.2 Aufbau der FCL-Dokumentation

Unterhalb des Astes »Klassenbibliothek« (»Class Library«) ist die Dokumentation alphabetisch nach Namespaces und innerhalb der Namespaces alphabetisch nach Klassennamen sortiert. Gegenüber früheren Dokumentationen hat Microsoft viel gelernt und bietet nun einen einheitlichen Aufbau der Hilfe-Dokumente.

Informationsdokument

Zu jedem Typ gibt es ein zentrales Informationsdokument (»About«-Dokument), das neben einer Einführung in den Zweck des Typs folgende Elemente enthält:

»About«-
Dokument

- ▶ Deklaration des Typs
- ▶ Vererbungshierarchie des Typs, wenn der Typ eine Klasse oder Schnittstelle ist
- ▶ Namespace, zu dem der Typ gehört
- ▶ Assembly, in der der Typ realisiert ist
- ▶ Betriebssysteme, auf denen der Typ verfügbar ist

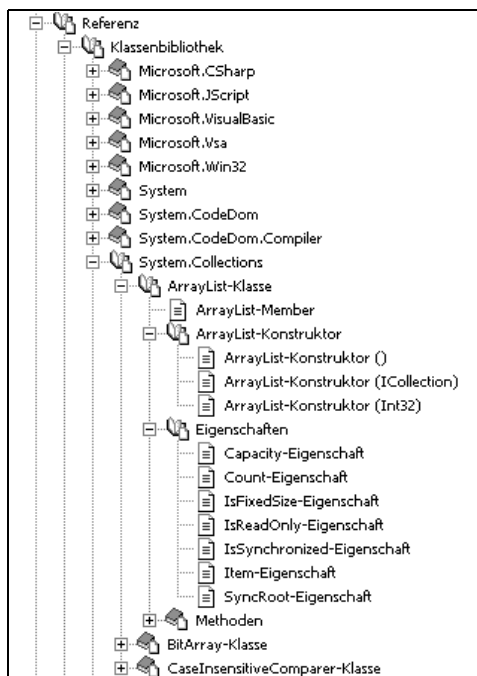


Abbildung 1.7: Aufbau der Dokumentation der Klasse System.Collections.ArrayList

»Alle Member«-Dokument

»All Members«-Dokument

Die FCL-Dokumentation bietet zu jedem Typ eine tabellarische Liste aller Mitglieder mit Kurzinformationen (»Alle Member«-Dokument) an.

Die Mitgliedsart wird im »Alle Member«-Dokument durch Icons angezeigt. Ein Attribut ist durch eine Hand mit einer Karte markiert, eine Methode durch eine fliegende Schachtel. Ein statisches Mitglied ist mit einem »S« gekennzeichnet, ein `Protected`-Mitglied durch einen Schlüssel.

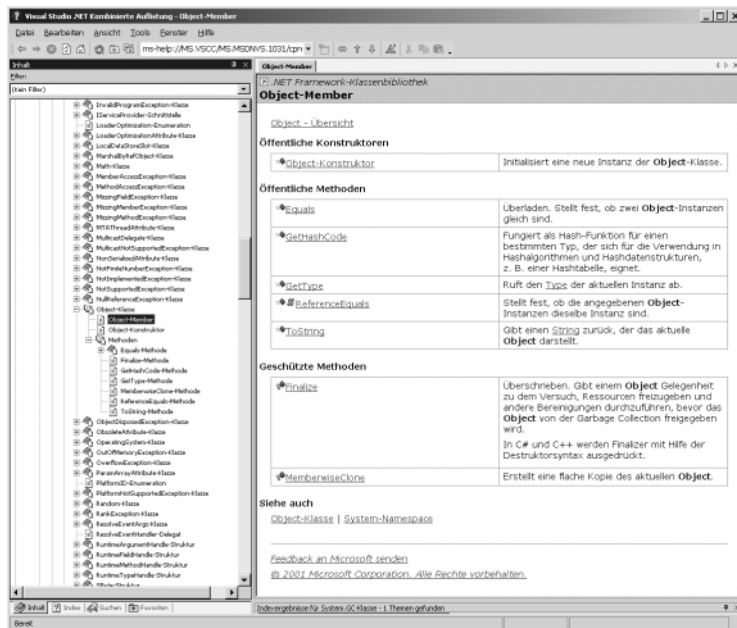


Abbildung 1.8: Die Klasse `System.Object` in der FCL-Dokumentation

Detaildokumente

Entweder über die »Alle Member«-Tabelle oder direkt über die Gliederungsansicht kann man die Detaildokumente anspringen. Zu jedem Klassenmitglied gibt es ein Detaildokument. Sofern eine Methode überladen ist, gibt es ein Gesamtdokument zu diesem Methodenamen und ein Detaildokument zu jeder einzelnen überladenen Methode.

Leider finden sich zu vielen Mitgliedern in den Detaildokumenten keine Code-Beispiele. Manchmal gibt es passende Code-Beispiele aber im »About«-Dokument oder in den Detaildokumenten zu den Konstruk-
toren.



1.6.3 Weitere Dokumentation

Allgemeine Erläuterungen und Anwendungsbeispiele zu FCL-Klassen finden sich auch an anderen Stellen in der .NET Framework-Dokumentation, z.B.(Tabelle nicht auf level)

Verweise

Deutsche Version	Englische Version
Visual Studio .NET/ Visual Basic und Visual C#/Program- mierung mit Komponenten .NET Framework/ Schnelleinstieg .NET Framework/ Programmieren mit dem .NET Frame- work	Visual Studio .NET/ Visual Basic and Visual C#/Program- ming with Components .NET Framework/ Quickstarts .NET Framework/ Programming with the .NET Frame- work

1.7 Werkzeuge

Um die unzähligen Klassen der FCL besser überblicken und erfor-
schen zu können, gibt es neben der Framework SDK-Dokumen-
tation auch ein paar hilfreiche Werkzeuge.

1.7.1 Windows Class Viewer (WinCV.exe)

Der Windows Class Viewer hilft bei der Suche nach einer geeig-
neten Klassen, indem er alle FCL-Klassen sucht, in denen ein
bestimmtes Wort vorkommt. Zu einer ausgewählten Klasse zeigt
er dann die Deklaration in C# an. Der WinCV ist Teil des .NET
Framework SDK und damit kostenlos.

WinCV

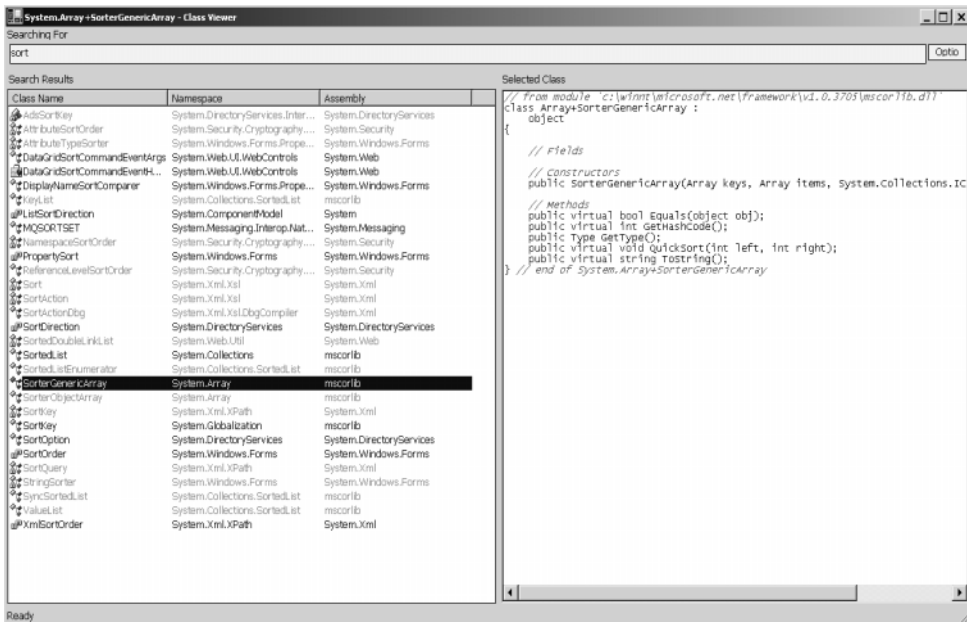


Abbildung 1.9: Suche nach dem Begriff »Sort« im Windows Class Viewer

1.7.2 Objektbrowser

Weiterentwick- lung des Objekt- katalogs

Der Objektbrowser ist ein Fenster von Visual Studio .NET. Er ist die Weiterentwicklung des Objektkatalogs aus Visual Basic 6.0 bzw. System InterDev 6.0. Optisch basiert der Objektbrowser stark auf dem Objektkatalog in Visual InterDev. Der Objektbrowser zeigt, gegliedert nach Komponenten, die hierarchisch die Namespace und die dort enthaltenen Typen an. Zu jedem Typ können die Member aufgelistet werden – in der Regel ist auch ein kurzer Hilfetext verfügbar. Auch die Vererbung und die implementierten Schnittstellen sind erkennbar. Die Taste **[F1]** führt zu den passenden Seiten in der Framework SDK-Dokumentation. Die Suchfunktion läuft nicht nur über die Klassen, sondern auch die Membernamen.

Der Objektkatalog kann nicht nur .NET-Komponenten, sondern auch COM-Komponenten anzeigen. Im Objektbrowser werden nicht automatisch alle auf dem System verfügbaren Komponenten angezeigt. Damit eine Komponente hier betrachtet werden kann, muss erst ein Verweis (Ordner VERWEISE) im Projekt vorhanden sein.

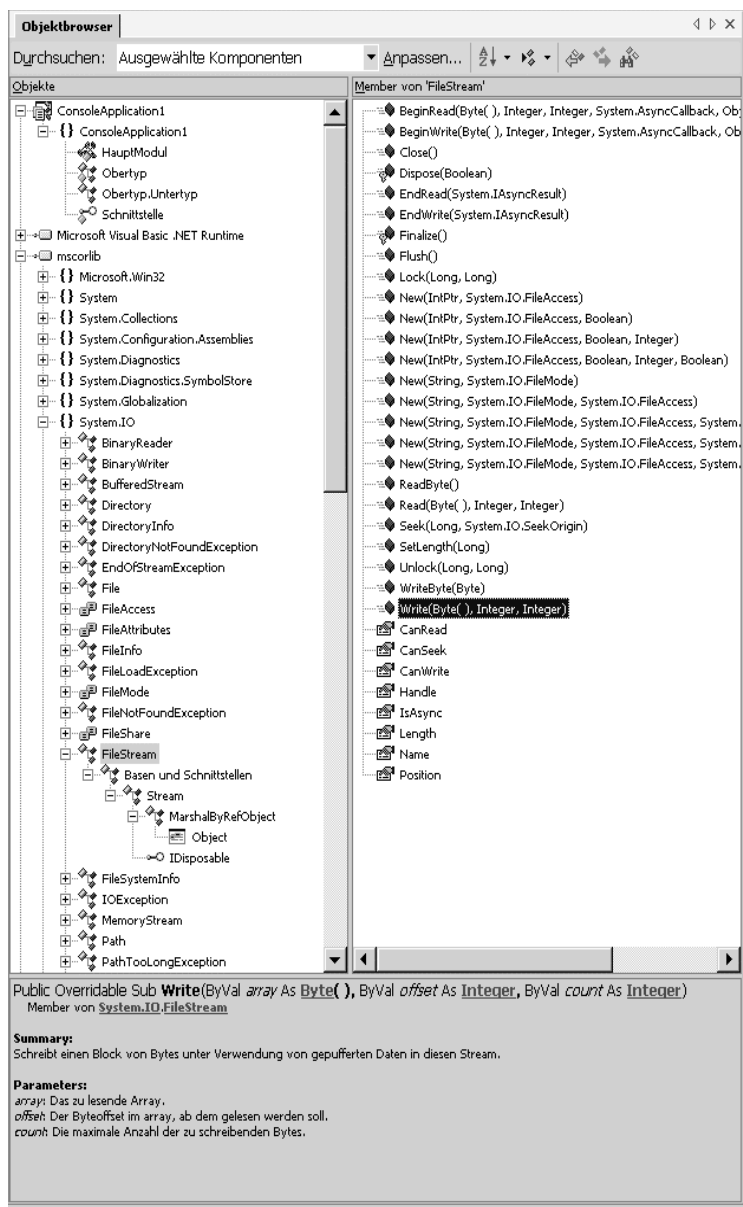


Abbildung 1.10: Der Objektbrowser in Visual Studio.NET zeigt die Methode Write() in der Klasse System.IO.FileStream.

Der Objektbrowser kann nicht getrennt von Visual Studio .NET gestartet werden.



1.7.3 Class Browser

Der Class Browser ist eine mit dem Framework SDK mitgelieferte Beispiel-Anwendung (*Samples/Quickstart/aspplus/ClassBrowser*), die einen ähnlichen Zweck erfüllt wie der Objektbrowser von Visual Studio.NET. Allerdings ist der Class Browser eine ASP.NET-Webanwendung, die über den Browser aufgerufen werden kann, sofern die Webanwendung dem lokal Webserver oder einem entfernt Webserver bereitgestellt wurde.

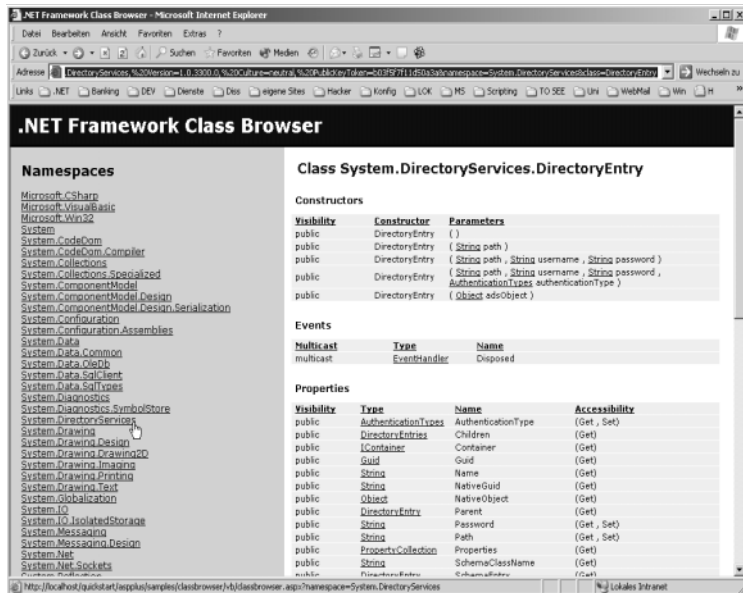


Abbildung 1.11: Die Klasse `System.DirectoryServices.DirectoryEntry` im Class Browser

Im Gegensatz zum Objektbrowser müssen keinen Verweise gesetzt werden; die wichtigsten FCL-Assemblies werden automatisch in einer Liste angezeigt. Die Liste ist über die `web.config`-Datei der ASP.NET-Anwendung erweiterbar.

1.7.4 Reflector

Der von Lutz Roeder entwickelte .NET Reflector ist ein Objektbrowser, der dem in Visual Studio .NET enthaltenen Objektbrowser in der Funktionalität sehr nahe kommt. Im Gegensatz zu seinem Vorbild ist der Reflector jedoch eine eigenständige Windows-Anwendung.

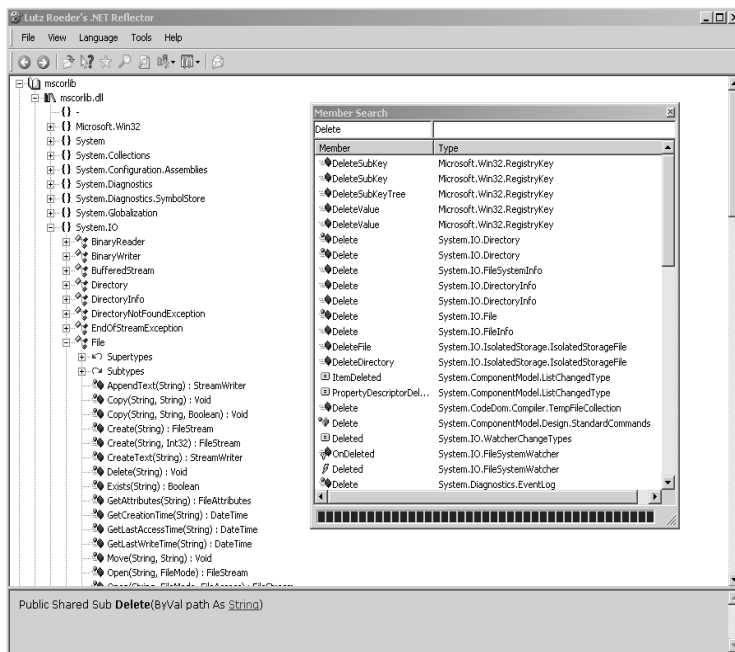


Abbildung 1.12: Die Klasse `System.IO.File` im Reflector

1.7.5 Intermediation Language Disassembler

Das Werkzeug **ILDASM** (*Intermediation Language Disassembler*) dient dazu, die Metadaten einer Assembly oder eines Moduls zu betrachten (engl. *Metadata Inspection*). Es ist vergleichbar mit der Funktion des OLE/COM-Viewers für COM-Komponenten. ILDASM ist Teil des .NET Framework SDK.

ildasm.exe

Der ILDASM bedient sich des .NET Reflection-API. Darüber hinaus kann der ILDASM – wie der Name schon besagt – auch den kompletten IL-Code einer Assembly disassemblieren.

ILDasm kann auch dazu verwendet werden, eine MSIL-Quelltextdatei zu erzeugen. MSIL-Quelltextdateien können mit dem Microsoft IL Assembler (*ilasm.exe*) in MSIL-Dateien übersetzt werden.

ildasm.exe

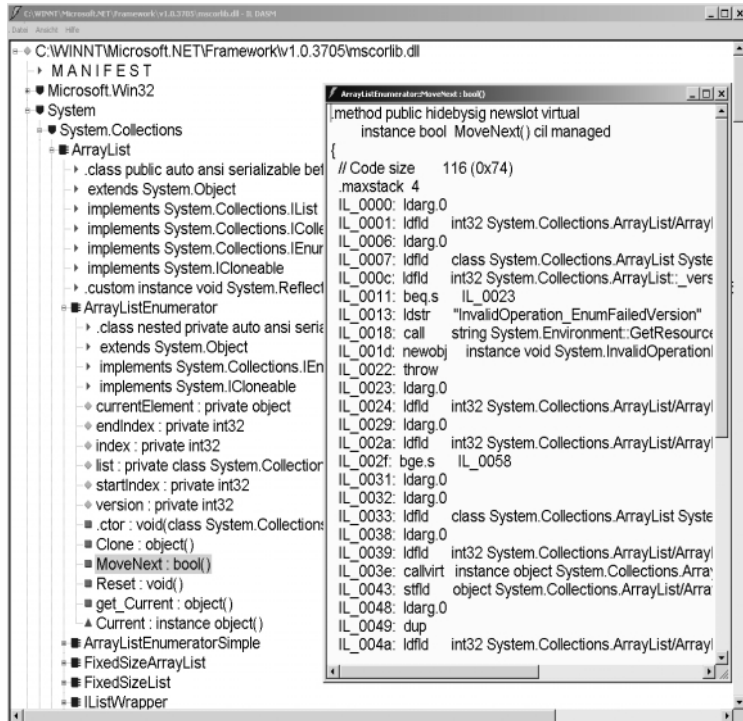


Abbildung 1.13: Die Assembly mscorlib.dll im IL Diassembler

1.8 Standardisierung

CLI Base Class Library

Viele Teile der FCL sind Bestandteile des ECMA-Standards 335 (Common Language Infrastructure, kurz: CLI). Microsoft hat ausgewählte Teile des .NET Frameworks bei der ECMA standardisieren lassen [CLI02a]. Dazu gehören auch zahlreiche Namespaces der Framework Class Library, die im Rahmen der CLI als *CLI Class Library* bezeichnet wird.

Die Mitsprache anderer Unternehmen in diesem Standardisierungsprozess war der Hauptgrund für die sehr große Zahl von Änderungen zwischen der Beta1-Version und der Beta2-Version der FCL.

Shared Source CLI

Zum Zeitpunkt der Drucklegung dieses Buchs gibt es Implementierungen der CLI und der CLI Class Library für Windows XP und FreeBSD. Zu diesen Implementierungen ist der Source Code für

jedermann zugänglich im Rahmen des Shared Source CLI. Sie finden den Source Code in der Beta2-Version (Stand 26.03.2002) auf der CD-ROM zu diesem Buch [CD:/install/CLI/Source/].

Ebenfalls frei zugänglich sind die Standardisierungsdokumente, die die Mitglieder und das Verhalten jeder einzelnen Klasse der CLI Class Library beschreiben [CD:/Weitere Informationen/ECMA Dokumente/TG3_CLI]. Auf der Microsoft-Homepage gibt es einen Klassen-Browser für die CLI Class Library [CLI02b].

Der Quellcode und die ECMA-Standardisierungsdokumente können Ihnen in Zweifelsfällen eine wichtige Hilfe bei der Suche nach einer geeigneten Funktionalität in der FCL bzw. bei der Suche nach Fehlern sein.



Bibliotheken und Profile

Die CLI Class Library setzt sich aus sieben einzelnen Bibliotheken zusammen, die zu zwei Profilen zusammengefasst sind (siehe Abbildung). Der Begriff Profile ist so zu verstehen, dass eine CLI-Implementierung nicht alle CLI-Bibliotheken implementieren muss, sondern dass die Umsetzung eines Profils ausreicht. Die Umsetzung eines Profils bedeutet allerdings die Implementierung aller zu dem Profil gehörenden Bibliotheken. Lediglich die Implementierung nicht zu einem Profil gehörender Bibliotheken ist jeder CLI-Implementierung freigestellt.

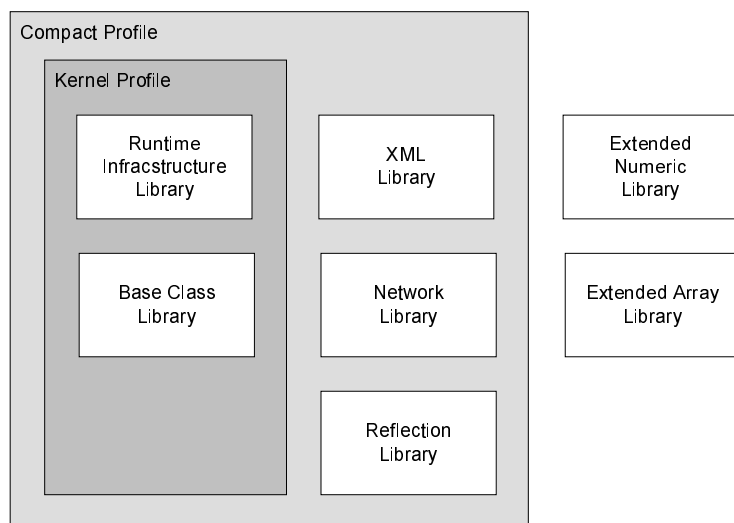


Abbildung 1.14: Bibliotheken und Profile in der CLI Class Library

Bibliothek	Inhalt
Runtime Infrastructure Library	Basisdienst
Base Class Library	Dateizugriff, benutzerdefinierte Attribute (Meta-Attribute), Stringmanipulation, Streams, Formatierung, Collections, etc.
Network Library	TCP/IP, HTTP
Reflection Library	Zugriff auf Metadaten
XML-Library	XML, XSD, XPath, XSLT, XML-Serialisierung
Extended Numerics Library	Fließkomma-Datentypen (<code>System.Single</code> , <code>System.Double</code> , <code>System.Decimal</code>)
Extended Array Library	Mehrdimensionale Arrays und Arrays mit einer Untergrenze >0

Tabelle 1.9: Inhalt der CLI-Bibliotheken

2 Visual Basic .NET (VB 7.0)

Visual Basic .NET (kurz: VB.NET) ist die Nachfolgeversion der Visual Basic 6.0-Vollversion und trägt intern die Versionsnummer 7.0. VB.NET folgt der Common Language Specification (CLS) und ist daher eine mit dem .NET Framework kompatible Sprache. Der aktuellen Marketing-Strategie folgend heißt die neue Version also Visual Basic .NET.

VB 6.0-Nachfolger

Visual Basic .NET wurde gegenüber seiner Vorgängerversion erheblich aufgewertet und ist nun eine echte objektorientierte Sprache mit Vererbung. Die Unterschiede zur Sprache C# sind eher syntaktischer Natur; hinsichtlich der Möglichkeiten gibt es nur marginale Unterschiede. Der COM- und .NET-Guru Don Box sagte auf der TechEd 2001 treffend: »Visual Basic .NET bedeutet, dass man sich nicht länger schämen muss, ein VB-Entwickler zu sein!«

Im Zuge von VB.NET hat Microsoft die drei Geschwister Visual Basic 6.0, Visual Basic for Applications (VBA) und VBScript zu einer Sprache fusioniert. Leider ist es keine Fusion unter Gleichen, denn während Visual Basic 6.0 alle seine zentralen Sprach-Features in das neue Visual Basic .NET übernehmen durfte (abgeschnitten wurden lediglich einige Zöpfe), fallen die in VBScript seit der Version 5.1 vorhandenen Funktionen zur Interpretierung von zur Laufzeit generiertem Programmcode (`Eval` und `Execute`) der Fusion zum Opfer. Andrew Clinick [CLI01] nährt die Hoffnung, dass diese Befehle in späteren Versionen von Visual Basic .NET wieder unterstützt werden (siehe Abbildung 2.1).

Fusion

Compiler

Der VB.NET-Compiler ist in der .NET-Laufzeitumgebung enthalten. Der Compiler kann über ein Kommandozeilenwerkzeug (`vbc.exe`) oder per Programmcode aufgerufen werden. Die Anschaffung der Entwicklungsumgebung Visual Studio ist also nicht zwin-

VB.NET-Compiler

gend nötig, um VB.NET zu programmieren. VB.NET-Programme können mit jedem beliebigen Texteditor geschrieben werden. Visual Studio .NET macht die Entwicklung von VB.NET-Anwendungen jedoch wesentlich komfortabler.

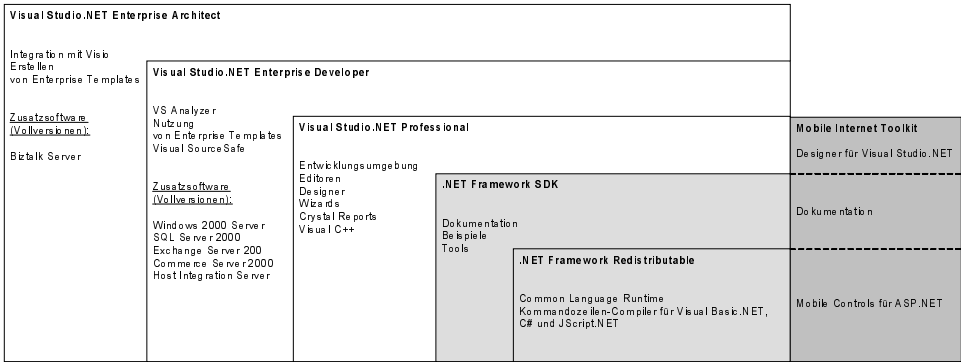


Abbildung 2.1: Umfang der Dialekte und Versionen von Visual Basic

Der VB.NET-Compiler erzeugt immer Managed Code. Er bietet keine Option zur Erzeugung von Unmanaged Code (Classic Code). Allerdings können .NET-Komponenten unter bestimmten Voraussetzungen von COM aus angesprochen werden.

2.1 Änderungen von VB 6.0 zu VB.NET (VB 7.0)

Grundlegende Änderungen

Von Visual Basic 5.0 zur Version 6.0 war es nur ein kleiner Schritt. Zwischen der aktuellen Version und dem Nachfolger 7.0 (VB.NET) hat Microsoft einen großen Wassergraben hinterlassen, der einen kräftigen Sprung mit guten Sprunggelenken erfordert.

Die Veränderungen in Visual Basic lassen sich in vier Gebiete einteilen:

- ▶ VB.NET ist in das .NET Framework integriert.
- ▶ Durch mehr objektorientierte Konzepte steigt Visual Basic von einer *objektbasierten* zu einer *objektorientierten* Sprache auf.
- ▶ Microsoft hat zahlreiche alte Sprachkonstrukte und Besonderheiten beerdigt.
- ▶ VB.NET bietet einige neue Sprachkonstrukte.

Inkompatibilität zu VB 6.0

Microsoft hat bei VB.NET ganz bewusst auf die Kompatibilität mit der Vorgängerversion VB 6.0 verzichtet. Inkompatibilitäten wurden aus zwei Gründen hingenommen:

- Alle Features aus VB 6.0, die signifikante Verwirrung bei den Entwicklern stifteten, wurden entfernt.
- Alle Features aus VB 6.0, die nicht kompatibel mit dem .NET Framework waren, wurden geändert oder entfernt.

Dabei wurde zwar versucht, die Inkompatibilität zwischen VB 6.0 und VB.NET so zu gestalten, dass ein automatisiertes Upgrade möglich ist. Dies war jedoch keine Pflichtanforderung. Es gibt einen Upgrade-Wizard von VB 6.0 nach VB.NET, der jedoch nicht alle Änderungen vollziehen kann.

2.1.1 Integration in das .NET Framework

Alle CLS-Sprachen (z. B. Visual C++ 7.0, C#, Visual Basic .NET, Eiffel, SmallTalk, Pascal) erhalten Zugriff auf eine einzige, gemeinsame Klassenbibliothek, die Common Language Runtime (CLR). Damit werden Aufrufe einer bestimmten (Betriebssystem-)Funktion in allen Sprachen – mit Ausnahme der Syntax – gleich sein. Visual Basic .NET verabschiedet sich daher auch von vielen eingebauten Funktionen: An ihre Stelle tritt der Aufruf von Methoden der in Namespaces organisierten Klassen der Framework Class Library (FCL). So wird aus *DoEvents* das Konstrukt *System.Windows.Forms.Application.DoEvents*. Das Ergebnis ist eine klare Trennung von Sprache und API.

Zur FCL gehören auch visuelle Komponenten, zur Gestaltung von Desktop-Anwendungen (*Windows Forms* oder *WinForms*) sowie von auf HTML basierenden Webanwendungen (*WebForms*). WinForms lösen die VB-Formulare ab.

Bisher unterlagen Visual Basic-Entwickler bei der systemnahen Programmierung einigen Beschränkungen. Viele Funktionen der in C++ entwickelten Routinen des klassischen Win32-APIs und einige COM-Komponenten waren für Visual Basic unerreichbar. Diese Einschränkungen sind mit .NET weitestgehend verschwunden, indem das Win32-API und das COM-API in FCL-Klassen gekapselt wurden. Damit stoßen VB-Entwickler in bislang nur umständlich

**Zugriff auf
Betriebssystem-
funktionen**

oder gar nicht erreichbare Gebiete wie z.B. Thread-Steuerung vor. In der ersten Version der Framework Class Library stehen aber noch nicht alle Funktionen des Win32-API in der FCL zur Verfügung.

2.1.2 Verbesserte Objektorientierung

Vererbung Das Konzept von Klassen und Objekten ist in Visual Basic bereits seit langem enthalten. Zum Gütesiegel »objektorientierte Sprache« fehlten der Sprache aber die Möglichkeit der Implementierungsvererbung und verschiedene Formen des Polymorphismus. In Visual Basic 6.0 wurde es mit dem Schlüsselwort `Implements` erstmals möglich, Schnittstellen zu vererben: Man konnte eine Klasse somit zwingen, die gleichen Methoden wie eine andere Klasse anzubieten. Dabei konnte man zwei zueinander polymorphe Objekte aus verschiedenen Klassen erzeugen; die Implementierung musste man jedoch »zu Fuß« übernehmen. Diese Arbeit wird einem nun durch das neue Schlüsselwort `Inherits` abgenommen. Wie in anderen objektorientierten Sprachen auch hat ein Entwickler natürlich weiterhin die Option, eine Methode trotzdem zu reimplementieren. Anders als in C++, aber wie in Java ist die Mehrfachvererbung, also die gleichzeitige Ableitung einer Klasse von mehreren anderen Klassen, nicht möglich. Visual Basic ermöglicht allerdings die Vererbung von mit WinForms gestalteten Bildschirmmasken (*Visual Inheritance*).

Klassendefinition Jede Klassendefinition benötigte in Visual Basic 6.0 eine eigene Datei, ein so genanntes Klassenmodul. VB.NET übernimmt das bereits in VBScript ab Version 5.0 realisierte Konzept, innerhalb einer Datei mehrere Klassen mit dem Schlüsselwort `Class` zu definieren. Konstruktormethoden, die bei der Instanziierung einer Klasse automatisch aufgerufen werden, können nun auch Parameter empfangen (*parametrisierte Konstruktoren*). Die aus C++ und Java bekannten Schlüsselwörter `Public`, `Private` und `Protected` steuern in der Klassendefinition die Vererbung einzelner Attribute und Methoden. Auch .NET-Komponenten können – analog zu COM-Komponenten – mehrere Schnittstellen haben. In Visual Basic 6.0 dienten noch abstrakte Basisklassen der Definition von Schnittstellen. VB.NET führt mit dem Schlüsselwort `Interface` eine wesentlich elegantere Möglichkeit ein.

Overloading Auch das Überladen einzelner Funktionen (*Overloading*) wird ermöglicht: Mehrere Funktionsdefinitionen dürfen den gleichen Funktionsnamen haben, sofern sie sich in Anzahl und/oder Daten-

typ der Parameter unterscheiden. VB entscheidet bei einem Funktionsaufruf anhand der Typsignatur, welche der gleichnamigen Implementierungen aufzurufen ist.

2.1.3 Veränderung bei den Sprachkonstrukten

Eine einerseits begrüßenswerte, aber andererseits auch kritische Änderung in VB.NET ist die Beseitigung von Inkonsistenzen und Redundanzen im Sprachumfang von Visual Basic. Dazu gehören insbesondere folgende Punkte:

Komponenten

- ▶ Die Datentypen sind der CLS angeglichen. So umfasst z.B. der Datentyp `Integer` in Zukunft 32 statt 16 Bit. Dafür vergrößert sich `Long` auf 64 Bit. Außerdem gibt es neue Typen, z.B. `Char`, `Short` und `Decimal`. Alle VB-Datentypen haben Entsprechungen im Namespace `System` der FCL.
- ▶ Die Abschaffung des viel kritisierten Schlüsselworts `Variant` bedeutet dagegen keineswegs, dass VB.NET jetzt keinen universellen Datentyp mehr bereitstellt: Der Datentyp `System.Object` übernimmt die Rolle von `Variant`.
- ▶ Alle Variablen sind Objekte. Auch elementare Datentypen haben Methoden (vgl. die Sprachen `SmallTalk` oder `JavaScript`).
- ▶ In VB.NET werden die Zustände `Empty` und `Null` für Variablen nicht mehr unterstützt. An deren Stelle tritt das Schlüsselwort `Nothing`.
- ▶ In VB.NET ist es möglich, einer Variablen schon bei der Deklaration einen Wert zuzuweisen.
- ▶ In VB.NET ist die Deklaration von Variablen im Standard Pflicht (kann aber zur Option gemacht werden). Die Verpflichtung zur Deklaration mit `Option Explicit Off` kann in VB.NET deaktiviert werden.
- ▶ VB.NET bietet eine Compiler-Option, die eine strikte Typprüfung erzwingt. Mit der Option `Strict` müssen alle Typkonversionen explizit ausgeführt werden. Implizite Typkonversionen führen zu einem Fehler. Wenn Sie die neue Option `Strict` nutzen, müssen alle Variablen zwingend typisiert werden (mit `As...`).
- ▶ In VB.NET wird zwischen einer Zeichenkette und einem einzelnen Zeichen (Datentyp `Char`) unterschieden. Um ein Literal vom Typ `Char` zu erzeugen, muss dem schließenden Anführungszeichen ein kleines `c` folgen, z.B. `"A"c`.

Datentypen und Variablen

- ▶ Arrays mit fester Größe sind abgeschafft. Alle Arrays sind nun dynamisch. Die Anzahl der Dimensionen muss bei der Deklaration festgelegt werden; mit `ReDim` kann nur noch die Größe der Dimensionen geändert werden.
- ▶ Arrays beginnen immer bei 0. Die Möglichkeit, ein Array bei einer anderen Untergrenze beginnen zu lassen, wurde abgeschafft.
- ▶ Eine Menge von Werten kann in geschweiften Klammern definiert und einem Array zugewiesen werden.
- ▶ Bei Variablendeklarationen können nun mehrere Variablen eines Typs in einer Zeile ohne Wiederholung des Datentyp-Schlüsselwortes deklariert werden.

```
Dim a,b,c as Integer
```

- ▶ Eine innerhalb eines Blocks (Schleife oder Bedingung) definierte Variable ist nur innerhalb der Schleife gültig, nicht in der ganzen Prozedur.
- ▶ Nur noch parametrisierte Attribute können Standardmitglieder einer Klasse werden.
- ▶ Die Definition von Properties wurde geändert:

```
Property Month( ) As Integer
    Get
        ...
    End Get
    Set(ByVal Value As Integer)
        ...
    End Set
End Property
```

Unterroutinen

- ▶ Alle Parameter werden im Standard mit ihrem Wert übergeben (*Call by Value*). Bisher war die Übergabe von Zeigern auf die Parameter üblich (*Call by Reference*). Dieses Verhalten wich von dem ab, was »herrschende Meinung« unter den Programmiersprachenentwicklern war.
- ▶ Bei einem Unterroutinenaufruf muss der Entwickler die Parameter nun immer in runde Klammern setzen. Bisher musste er zwischen Funktionen mit Rückgabewert und Prozeduraufrufen ohne Klammern unterscheiden.
- ▶ In VB.NET muss ein optionaler Parameter einen Standardwert haben.

- ▶ Die Definition von Klassen in VB.NET entspricht der Definition von Klassen in VBScript 5.x mit `Class...End Class`. **Klassen**
- ▶ Strukturen (`Structure...End Structure`) können nicht nur Daten, sondern auch Code enthalten und sind daher eine abgespeckte Variante einer Klasse. Dieses Konstrukt ersetzt `Type...End Type`.
- ▶ Es ist nicht mehr erlaubt, ein Standardattribut in einer Klasse festzulegen, das automatisch verwendet wird, wenn ohne nähere Angabe eines Attributnamens Bezug auf eine Objektvariable genommen wird.
- ▶ In VB.NET gibt es echte Konstruktoren, so dass bereits bei der Instanziierung Parameter übergeben werden können.
- ▶ In VB.NET erfolgt die Instanziierung bei einem `New`-Operator sofort und nicht erst bei der ersten Verwendung des Objekts.
- ▶ Verschiedene Schlüsselwörter (z.B. `Set`, `Let`, `Type`, `Variant`, `GoSub`, `Null`, `Exit`, `Wend`, `Option Base`) wurden gestrichen oder durch andere Schlüsselwörter ersetzt. **Weitere Änderungen**
- ▶ Die `While...Wend`-Schleife heißt in VB.NET `While...End While`.
- ▶ In VB.NET sind viele ehemals eingebaute Funktionen nun Teil der Framework Class Library. Aus Gründen der Kompatibilität gibt es jedoch die Komponente *Microsoft Visual Basic .NET Runtime* (*Microsoft.VisualBasic.dll*). Diese Assembly enthält viele aus VB6 bekannte Funktionsnamen, die als Abbildung auf Methoden in FCL-Klassen implementiert sind.

2.1.4 Neue Konstrukte

Daneben gibt es aber auch einige völlig neue Konstrukte in VB.NET, die bislang weder in VB6 noch in einem anderen Visual Basic-Dialekt vorkamen:

Neue Konstrukte

- ▶ In VB.NET können beliebig viele Module eine `.vb`-Datei bilden. Dazu dient das neue Konstrukt `Module...End Module`.
- ▶ VB.NET unterstützt daher nun – wie andere Sprachen auch schon – Exceptions mit dem Konstrukt `Try...Catch...Finally`.
- ▶ `Throw` ExceptionKlasse erzeugt eine Exception.
- ▶ Mit `SyncLock...End SyncLock` wird ein Programmabschnitt markiert, der vor dem gleichzeitigen Aufruf durch mehrere Threads geschützt sein soll.

- ▶ Durch die neue Compiler-Option `Option Strict` wird die Typisierung von Variablen und Konstanten erzwungen. Außerdem wird dadurch die implizite Typkonvertierung eingeschränkt.
- ▶ Es gibt neue Operatoren, die die Zuweisung mit einer Operation verknüpfen (z.B. `+=`, `-=`, `*=`).
- ▶ VB.NET bietet zwei neue logische Operatoren (`AndAlso` und `OrElse`) zur Short-Circuit-Auswertung. Dadurch wird die Auswertung eines Ausdrucks abgebrochen, sobald das Ergebnis feststeht.
- ▶ In VB.NET kann zur Rückgabe von Werten aus Funktionen das neue Schlüsselwort `Return(wert)` verwendet werden.
- ▶ VB.NET unterstützt das Overloading von Prozeduren und Funktionen mit dem Schlüsselwort `Overloads`. Das bedeutet, dass es mehrere Unterrouinen gleichen Namens, aber mit unterschiedlicher Signatur geben kann.
- ▶ Schnittstellen können in VB.NET direkt, ohne Umweg über abstrakte Klassen, definiert werden. Es gibt dazu ein neues Schlüsselwort `Interface`, das ähnlich wie `Class` einen Block (`Interface...End Interface`) bildet.
- ▶ Zur Bindung einer Ereignisbehandlungsroutine an ein Ereignis bietet VB.NET die Konstrukte `AddHandler`, `RemoveHandler` und `Handles`.

2.1.5 Bewertung

Vorbereitung auf VB.NET

Die in diesem Kapitel genannten Veränderungen sind nur ein Ausschnitt und stecken dennoch bereits die enorme Größe des Unterschieds zwischen Visual Basic 6.0 und Visual Basic .NET ab. Microsoft trennt sich auf Kosten der Abwärtskompatibilität von Altlasten. Dass es jemals Tools gelingen wird, alle VB 6.0-Codes automatisch ohne Fehler in VB.NET-Code zu überführen, ist zweifelhaft. Diejenigen VB-Entwickler werden die besten Karten haben, die sauberen VB-Code entwickeln und auf »Abkürzungen« wie implizite Verwendung der Standardattribute (Default Properties) verzichten.

Psychologische Schranken

Auch die Fusion zwischen VBScript, VBA und VB wirft einen Schatten: Es war schon schwierig genug, vielen programmierunerfahrenen Systemadministratoren VBScript nahe zu bringen. Es

wird schwierig, diesem Personenkreis in Zukunft zu vermitteln, dass sie jetzt »richtig« objektorientiert programmieren müssen.

Laut Andrew Clinick [BAK01] war die Zusammenführung der drei Dialekte aber der Wunsch der Microsoft-Kunden. Microsoft gibt aber auch etwas Hoffnung mit der Aussage, Visual Basic 6.0, VBA und VBScript werden noch einige Zeit parallel zu VB.NET unterstützt werden.

2.2 Entwicklung von VB.NET-Programmen mit dem Kommandozeilen-Compiler

Zur Entwicklung einer Anwendung mit Visual Basic .NET reicht die Installation des .NET Framework-SDKs. Es enthält den Visual Basic Kommandozeilen-Compiler `vbc.exe`. Komfortabler geht es natürlich mit der Visual Studio .NET-Entwicklungsumgebung. Visual Studio .NET wird im nächsten Kapitel behandelt. Eine kostenlose Alternative dazu stellt der Freeware-Editor SharpDevelop dar [CD:/install/Werkzeuge/Editoren/sharpdevelop/], der – trotz seines Namens – auch mit VB.NET arbeitet. Dieses Kapitel möchte Ihnen einen Einstieg in den VB.NET-Compiler vermitteln.

Die Beispiele aus diesem Kapitel finden Sie auf der CD-ROM unter [CD:/code/Kapitel2_VB.NET/].

2.2.1 Erstellung einer Hello-World-Anwendung

Gehen Sie folgendermaßen vor, um Ihre erste einfache VB.NET-Anwendung (Assembly) zu erstellen:

**Hello World in
.NET**

1. Öffnen Sie einen Texteditor, z.B. den bei Windows mitgelieferten Editor *Notepad.exe*.
2. Importieren Sie den Namespace `Microsoft.VisualBasic`, da Ihnen sonst viele elementare Funktionen wie `MsgBox()` nicht zur Verfügung stehen.

```
Imports Microsoft.VisualBasic
```

3. Deklarieren Sie ein Modul mit `Module...End Module`.
4. Deklarieren Sie innerhalb des Moduls `Sub main()`.

5. Implementieren Sie `Sub main()`. Vergessen Sie am Ende nicht das `End Sub`. Eine einfache Implementierung könnte die Ausgabe einer Dialogbox mit der `MsgBox()`-Funktion sein.
6. Speichern Sie das Programm unter *hello.vb*.



Bitte beachten Sie bei der Verwendung von Konstanten im `MsgBox()`-Befehl, dass Sie den Namen der Konstantenliste vor der Konstante nennen müssen. Die Konstante ist zwar Bestandteil des importierten Namespaces `Microsoft.VisualBasic`, dennoch findet der Compiler die Konstante sonst nicht.

Code

Realisierung Das Programm sollte dann in etwa so aussehen:

```
' =====
' HelloWorld.vb
' (C) Holger@Schwichtenberg.de
' =====

Imports Microsoft.VisualBasic

Module ErstesModul
    Sub main()
        Dim Antwort As Integer
        antwort = MsgBox("Hello World!", _ MsgBoxStyle.OKOnly,
            "Meine erste .NET-Anwendung")
    End Sub
End Module
```

Listing 2.1: Eine einfache .NET-Windows-Anwendung
 [CD:/code /VB.NET/HelloWorld_Windows/Hello_Windows.vb]

Übersetzung

vbc.exe Übersetzen Sie das Programm an der Kommandozeile mit

```
vbc hello.vb
```

Der Compiler erzeugt dann eine Datei mit dem Namen `HELLO.EXE`.

Starten Sie das Programm nun mit

```
Hello
```

Standard ist DOS-Anwendung

Wenn Sie diese Anwendung nicht aus dem DOS-Fenster heraus, sondern per Klick auf das Icon im Explorer starten, werden Sie sehen, dass Windows ein neues DOS-Fenster öffnet, das sich schließt, sobald Sie das Dialog-Fenster schließen. Das liegt daran,

dass der Visual Basic .NET-Compiler im Standard eine DOS-Anwendung (Assembly-Typ: Console Application) erzeugt.

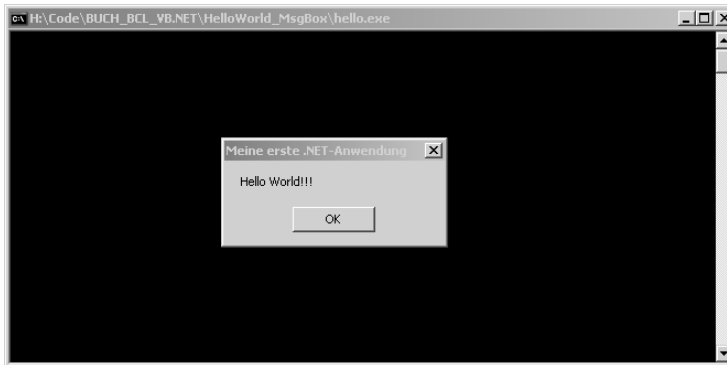


Abbildung 2.2: Ausgabe von »hello.exe«

Übersetzung als Windows-Anwendung

Um den VB.NET-Compiler zum Erstellen einer Windows-Anwendung ohne DOS-Fenster (Assembly-Typ: *Windows Application*) zu bewegen, kompilieren Sie bitte so:

```
vbc hello.vb /t:winexe
```

Das `/t` steht für das Wort »Target« (deutsch: Ziel).

2.2.2 Entwicklung einer DOS-Anwendung

Oft (gerade zu Testzwecken) ist aber auch eine DOS-Anwendung (alias Konsolenanwendung, engl. *Console Application*) ohne grafische Benutzerschnittstelle sinnvoll. Um eine Ausgabe in das DOS-Fenster zu senden, wird der Namespace `System` benötigt. Dort ist die Klasse `Console` mit der Methode `WriteLine()` implementiert.

**Ausgaben ins
DOS-Fenster**

```
Imports System

Module ErstesModul
    Sub main()
        Console.WriteLine _
            ("Hello World an der Kommandozeile!")
    End Sub
End Module
```

Listing 2.2: Eine einfache .NET-DOS-Anwendung

[CD:\code\DOTNET\VB.NET\HelloWorld_DOS\Hello_DOS.vb]

Die Übersetzung ist analog zur Windows-Anwendung. Der Zieltyp ist »exe«. Das ist aber der Standard, weshalb die Option `/t:exe` optional ist.

```
vbc hello_dos.vb /t:exe
```



Wenn Sie diese Anwendung mit der Option `/t:winexe` kompilieren, werden Sie beim Start der Anwendung gar nichts sehen, da bei diesem .NET-Anwendungstyp alle Ausgaben der Klasse `Console` unterdrückt werden.

2.2.3 Nutzung von FCL-Klassen

/reference Voraussetzung für die Nutzung einer Klasse aus einer anderen Assembly ist, dass diese Assembly bei der Übersetzung referenziert wird. Eine Referenz definiert man über den Parameter `/reference:`

```
vbc client.vb /reference:pfad\komponente.dll
```

Nach dem Setzen einer Referenz können alle öffentlichen Klassen der referenzierten Assembly verwendet werden – zunächst allerdings nur über ihren vollständigen Namen, inklusive Namespace.

XML-Klassen nutzen

Das folgende Listing zeigt am Beispiel der Nutzung von XML-Klassen aus der FCL, dass es zu sehr langen Ausdrücken bei Deklarationen und Instanziierung kommt. Ohne eine weitere Anweisung gilt das auch für die VB6-Funktionen wie `MsgBox()`.

```
' =====
' FCL-Nutzung1.vb
' (C) Holger@Schwichtenberg.de
' =====

Module FCLNutzung1
    Sub main()
        Const EINGABE = _
            "<Autor>" + _
            "<Vorname>Holger</Vorname>" + _
            "<Nachname>Schwichtenberg</Nachname>" + _
            "</Autor>"

        Dim d As System.Xml.XmlDocument
        d = New System.Xml.XmlDocument()
        d.LoadXml(EINGABE)
        Dim vorname As System.Xml.XmlNode
        Dim nachname As System.Xml.XmlNode
        vorname = d.SelectSingleNode("*/Vorname")
        nachname = d.SelectSingleNode("*/Nachname")
        Microsoft.VisualBasic.MsgBox( _
```



```

        "Name des Autors:" & vorname.InnerXml & _
        " " & nachname.innerXml,)
    End Sub
End Module

```

Listing 2.3: Beispiel zur Nutzung von Klassen aus System.Xml.dll
 [CD: ICode/Kapitel2_VB.NET/FCL_Nutzung/FCL-Nutzung1.vb]

Dieser Code kann nur übersetzt werden, wenn der Compiler eine Referenz auf die Assembly *System.Xml.dll* erhält, in der die Klassen *XmlDocument* und *XmlNode* implementiert sind. Da die *System.Xml.dll* eine Shared Assembly im Global Assembly Cache ist, ist die Angabe des Pfads zu dieser DLL nicht notwendig.

```
vbc FCL-Nutzung1.vb /reference:system.xml.dll
```

Ohne die Referenz würde sich der Compiler beschweren, dass die Typen *System.Xml.XmlDocument* und *System.Xml.XmlNode* nicht definiert sind.

Der VB.NET-Compiler bindet immer automatisch zwei Assemblies implizit hinzu: *mscorlib.dll* und *Microsoft.VisualBasic.dll*, wie der folgende Screenshot der Assembly *FCL-Nutzung1.exe* zeigt. Diese beiden Assemblies brauchen nicht explizit referenziert zu werden.

**Automatisch
gebundene
Assemblies**

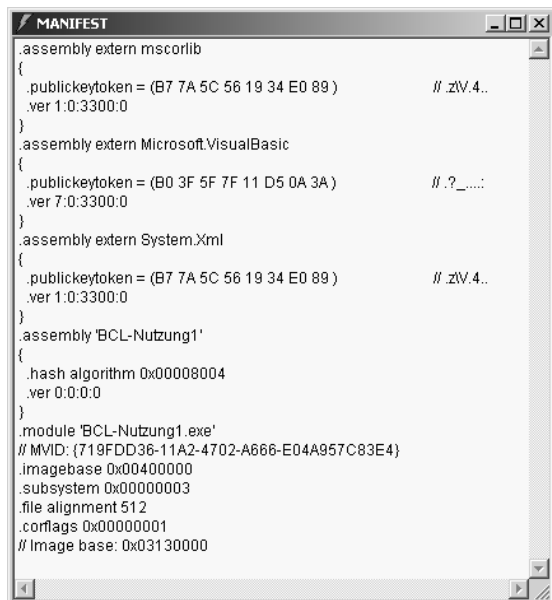


Abbildung 2.3: Ansicht des Manifests der Assembly »FCL-Nutzung1.exe«

Imports-Anweisung

Imports Die Nutzung von Typen aus einem anderen Namespace kann durch die *Imports*-Anweisung vereinfacht werden. Diese Anweisung bindet alle genannten Typen bzw. Typenmengen so ein, dass sie über ihren Namen angesprochen werden können. Dabei muss die Aneinanderreihung einer *Imports*-Anweisung und des im Code genannten Typnamens den Gesamtnamen ergeben.

Beispiel Zum Beispiel führt

```
Imports a.b
```

dazu, dass die Klasse

```
a.b.c.d.Klasse
```

verkürzt so angesprochen werden kann:

```
c.d.Klasse.
```

Beispiel

Das obige XML-Beispiel noch einmal mit *Imports*-Anweisungen

```
' =====
' FCL-Nutzung2.vb
' (C) Holger@Schwichtenberg.de
' =====

Imports Microsoft.VisualBasic
Imports System.Xml
Imports System

Module FCLNutzung2
  Sub main()
    Const EINGABE = _
      "<Autor>" + _
      "<Vorname>Holger</Vorname>" + _
      "<Nachname>Schwichtenberg</Nachname>" + _
      "</Autor>"
    Dim d As XmlDocument
    d = New XmlDocument()
    d.LoadXml(EINGABE)
    Dim vorname As XmlNode
    Dim nachname As XmlNode
    vorname = d.SelectSingleNode("*/Vorname")
    nachname = d.SelectSingleNode("*/Nachname")
    Dim Ausgabe = "Name des Autors: " & _
      vorname.InnerXml & " " & nachname.InnerXml
    MsgBox(Ausgabe)
```

```
        Console.WriteLine(Ausgabe)
    End Sub
End Module
```

Listing 2.4: Beispiel zur Nutzung von Klassen aus System.Xml.dll unter Verwendung der Imports-Anweisung
[CD: /Code/Kapitel2_VB.NET/FCL_Nutzung/FCL-Nutzung2.vb]

2.3 Nutzung der Visual Studio .NET-Entwicklungsumgebung

Natürlich ist es komfortabler, eine .NET-Anwendung mit Visual Studio .NET zu entwickeln als mit den Kommandozeilen-Compilern. Darauf setzt auch Microsoft, wenn die Kommandozeilen-Compiler verschenkt werden, die Entwicklungsumgebung aber weiterhin zusätzliche Lizenzgebühren kostet.

**Komfortable
Entwicklungs-
umgebung**

In Visual Studio .NET ist die Entwicklungsumgebung für alle Sprachen und Anwendungstypen (fast) gleich. Zusätzlich bietet die Entwicklungsumgebung auch Editoren für XML- und HTML-Dateien.

Visual Studio .NET heißt intern Microsoft Development Environment 2002 (interne Versionsnummer der deutschen Endversion: 7.0.9515.0).



2.3.1 Visual Studio .NET-Überblick

Microsoft bietet Visual Studio .NET in drei Versionen an: Professional Edition, Enterprise Developer Edition und Enterprise Architect Edition (in aufsteigendem Funktionsumfang).

Die **Enterprise Developer Edition** setzt sich von der Professional Edition durch zusätzliche Entwicklungstools (Visual SourceSafe, Application Center Test, Visual Studio Analyzer) und eine Entwicklungslizenz der .NET Enterprise Server (SQL Server 2000, Exchange Server 2000, Commerce Server 2000, Host Integration Server) ab.

**Enterprise
Developer Edition**

Im Rahmen der **Enterprise Architect Edition** gibt es zusätzlich noch einen BizTalk Server, Integration in Microsoft Visio sowie die Möglichkeit, Visual Studio .NET durch so genannte Enterprise Templates den eigenen Entwicklungsprojekten anzupassen. Eine abgespeckte Standard Edition gibt es für jede einzelne der drei Visual Studio-Sprachen. In den Professional- oder Enterprise-Editionen verkauft Microsoft die Sprachen aber nur als Paket.

**Enterprise
Architect Edition**

Die in der folgenden Abbildung grau hinterlegten Produkte sind kostenlos. Das Mobile Internet Toolkit ist ein kostenloses Add-On.

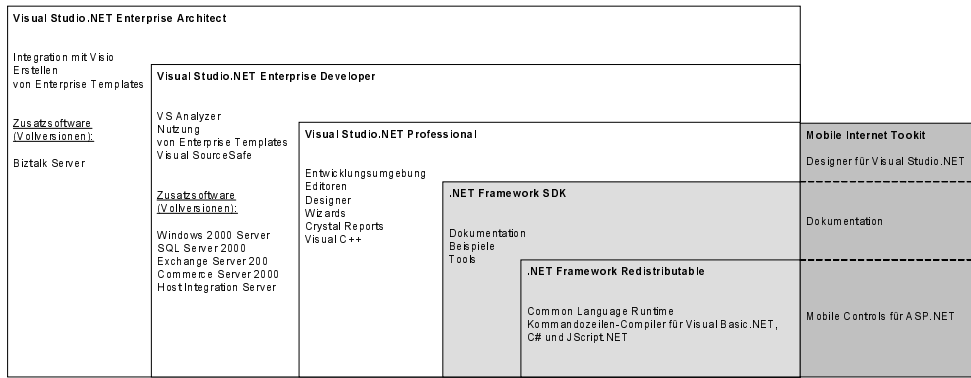


Abbildung 2.4: Überblick über die .NET-Entwicklerprodukte



Wenn Sie lieber deutsche Dokumentation lesen, dann hat der Erwerb einer deutschen Version von Visual Studio .NET für Sie einen Vorteil: Die komplette Framework SDK-Dokumentation ist dort ins Deutsche übersetzt.

Editoren

Für Visual Studio .NET benötigt ein Software-Entwickler einen großen Monitor, denn die zahlreichen Fenster der Entwicklungsumgebung fordern Raum. Code-Editoren liefert Microsoft zunächst nur für C# und Visual Basic .NET, während das Visual Studio .NET-Entwicklungsteam JScript.NET als dritte .NET-Sprache aus Kapazitätsgründen zunächst einmal vernachlässigt hat.

Weitere Sprachen

JSharp .NET (J#) ist als vier .NET-Sprache von Microsoft etwas später als Add-on ausgeliefert worden. Auf der CD-ROM zu diesem Buch finden Sie nicht nur J#, sondern auch noch weitere .NET-Sprachen [CD:/install/Sprachen/] (siehe Abbildung 2.5).

Die Unterstützung bei der Eingabe von Quellcode wurde in Visual Studio .NET gegenüber Visual Studio 6.0 weiter verbessert. Die im Folgenden genannten Änderungen sind nur eine Auswahl.



Wenn man mit C# und Visual Basic .NET arbeitet, merkt man schnell, dass die Code-Editoren bezüglich des Bedienungskomforts und der Eingabehilfen keineswegs identisch sind. Die Unterschiede liegen im Detail (siehe Kapitel 3).

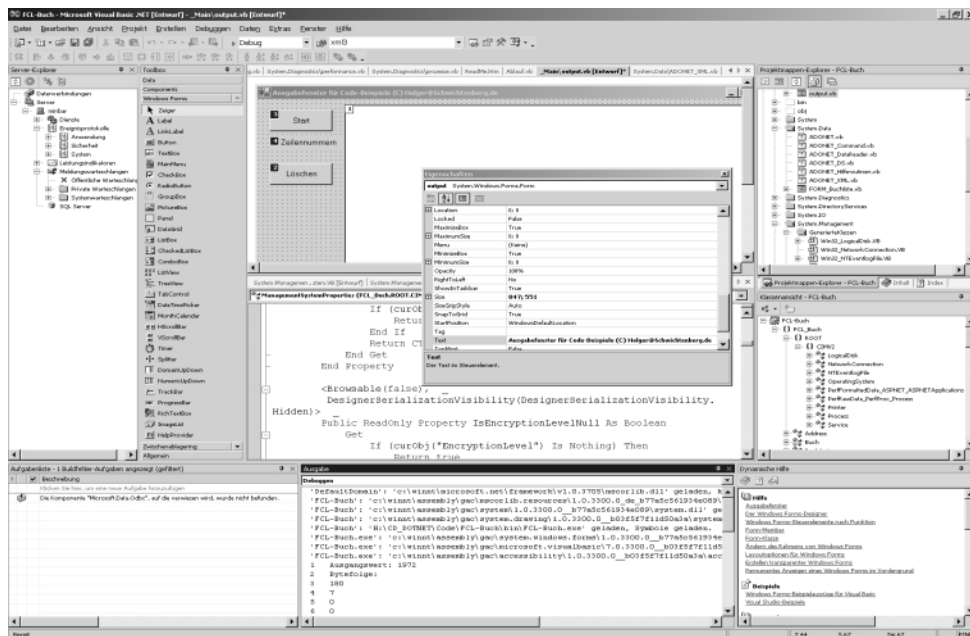


Abbildung 2.5: Fensterpracht in Visual Studio .NET

Aufklappbarer Code

In allen Code-Editoren kann der Entwickler die Implementierung von einzelnen Klassen und Unterroutinen durch die Zeichen **+** und **-** ähnlich wie bei den Verzeichnissen im Windows Explorer verbergen. Es bleibt dann nur der Klassen- bzw. Unterroutinen-Rumpf stehen mit einem Verweis, dass die Implementierung verborgen ist. Mit der Compiler-Direktive `#Region...#End Region` kann der Entwickler auch eigene zuklappbare Code-Abschnitte definieren.

Automatisches Vervollständigen und Einrücken

Im Visual Basic-Code-Editor arbeitet die Autovervollständigen-Funktion nun nicht nur für die Konstrukte `Sub` und `Function`, sondern ergänzt auch bei Eingabe anderer Sprachkonstrukte wie `For` und `Do While` automatisch das schließende Element (`Next`, `Loop`, etc).

Nach Blockkonstrukten, bei denen nach den Regeln zur übersichtlichen Programmierung ein vergrößerter Einzug folgen sollte (z. B. `Sub`, `For...Next`, `Do While...Loop`), wird der Blockinhalt automatisch eingerückt.

Region

Autovervollständigen

Einrücken

Hintergrundkompilierung

Hintergrund-kompilierung

Sehr nützlich ist die Hintergrundkompilierung: Schon während der Eingabe, jeweils nach Verlassen einer Code-Zeile, prüft die Entwicklungsumgebung auf Syntax- und Typfehler. Eine geschlängelte blaue Linie zeigt dem Entwickler sofort an, dass hier etwas nicht stimmt. Parallel dazu erzeugt Visual Studio .NET einen Eintrag im Fenster AUFGABENLISTE, in dem die Compiler auch die Kompilierungsfehler melden. Der Entwickler wird nicht mehr wie bei der automatischen Syntaxprüfung in Visual Basic 6.0 durch Dialogboxen gestört. Zudem bemerkt Visual Studio .NET wesentlich mehr Fehler, z.B. dass eine Variable oder eine verwendete Unter-routine undeklariert ist.

Aufgabenliste

Aufgabenliste

Die AUFGABENLISTE zeigt nicht nur Kompilierungsfehler an, sondern ermöglicht auch die automatische oder manuelle Aufnahme von Stichpunkten. Die manuelle Aufnahme erfolgt einfach über das Fenster selbst (»Klicken Sie hier, um eine neue Aufgabe hinzuzufügen«). Automatisch aufgenommen werden Kommentare, die bestimmte Schlüsselwörter enthalten. Vordefiniert ist das Schlüsselwort »TODO«. Ein Kommentar, der mit diesem Schlüsselwort beginnt, wird in die Aufgabenliste aufgenommen. Weitere Schlüsselwörter kann man über EXTRAS/OPTIONEN/UMGEBUNG definieren.

Debugging

Kein Edit-Resume

Auf ein lieb gewonnenes Feature müssen Visual Basic-Entwickler in Visual Basic .NET verzichten: Die EDIT/RESUME-Funktion beim Debuggen gibt es nicht mehr. Unter Visual Basic 6.0 konnte der Tester laufenden Programmcode, der innerhalb der Entwicklungsumgebung wegen eines Fehlers oder eines Haltepunkts abbrach, (in bestimmten Grenzen) verändern und dann weiterlaufen lassen. Visual Basic .NET erwartet nach einer Änderung am Quellcode immer die Neukompilierung und den Neustart des Programms. Auch die MEMBER AUFLISTEN-Funktion für Programmcode in Active Server Pages-Dateien ist nicht mehr vorhanden. Microsoft drängt damit die Entwickler dazu, die neuen Code-Behind-Dateien zu nutzen, die HTML-Code und Programm-Code voneinander trennen.

Designer

Nicht unerwähnt bleiben sollten auch die Funktionen, die das *Visual* im Namen Visual Studio ausmachen: die Designer, die die Erstellung von Software-Bausteinen per Maus ermöglichen. Visual Studio .NET bietet Designer für Windows-Fenster (Windows Forms), dynamische Webseiten (Webforms), Windows-Steuerelemente (Windows User Controls), Web-Steuerelemente (Web User Controls), Berichte (Crystal Reports), Bitmaps, Icons, Cursor, XML-Dokumente, XSD-Schema-Dokumente sowie fensterlose .NET-Komponenten (in Visual Studio .NET »Komponentenklasse« genannt).

**Visuelles
Programmieren**

Der letztgenannte Designer ist ein Container für verschiedene Systembausteine (z.B. Datenbankverbindungen, Ereignisprotokolle, Leistungsindikatoren, Dienste) aus der Werkzeugsammlung oder dem Server Explorer. Jeder per Drag&Drop auf den Designer abgelegte Systembaustein erscheint als ein Icon mit Namen und kann über das Eigenschaftsfenster konfiguriert werden. Aus dem Programmcode heraus kann der Entwickler den Systembaustein über den im Designer vergebenen Namen ansprechen, weil Visual Studio für die im Designer abgelegten Icons automatisch Programmcode generiert, der in einer Region mit Namen »Component Designer Generated Code« zusammengefasst ist.

**Fensterlose .NET-
Komponenten**

Bei der ersten Ankündigung von .NET im Juli 2000 sprach Microsoft von der »Orchestrierung« von Anwendungen mit Visual Studio .NET – davon ist man aber später immer mehr abgekommen, denn so weit, wie man es wohl einmal geplant hatte, reicht die visuelle Unterstützung des Programmierens in Visual Studio .NET 2002 dann eben doch nicht.



2.3.2 Projektmappen und Projekte

Ein Visual Studio .NET-Projekt dient dem Anlegen einer *Single-File-Assembly* in einer bestimmten Sprache. Die Sprache muss beim Anlegen eines Projekts festgelegt werden. Sie gilt für das ganze Projekt. Es können also nicht einzelne Teile eines VS.NET-Projekts in verschiedenen Sprachen geschrieben werden. VS.NET unterstützt in der Version 2002 (noch) nicht das Erstellen von *Multi-File-Assemblies*.

**Single-File-
vs. Multi-File-
Assemblies**

Außerdem ist beim Anlegen eines neuen Visual Studio .NET-Projekts zwischen verschiedenen .NET-Anwendungstypen (in diesem

**Anwendungs-
typen**

Zusammenhang Projekttypen genannt) zu wählen – darunter die Typen Windows-Anwendung (engl. Windows Application) und Konsolenanwendung (engl. Console Application). Zwischen diesen beiden .NET-Anwendungstypen kann aber auch später jederzeit im Eigenschaftsfenster des Projekts gewechselt werden. Anders als in der VB6-IDE wird für das neue Projekt automatisch ein eigenes Unterverzeichnis angelegt. In diesem Unterverzeichnis werden eine Projektdatei (Dateiextension *.vbproj*), eine Projektmappen-Datei (engl. Solution, Dateiextension: *.sln*) und ein Verzeichnis */bin* erzeugt, in das die kompilierten MSIL-Dateien abgelegt werden.

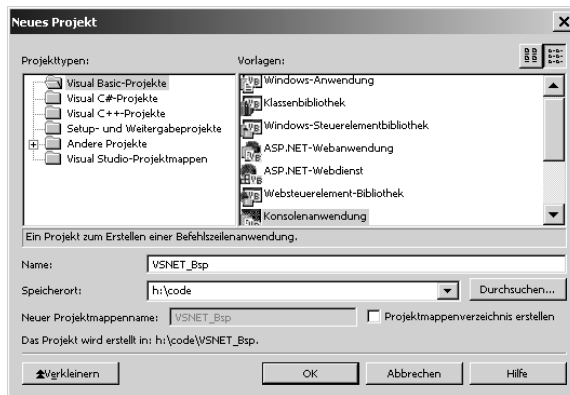


Abbildung 2.6: Anlegen eines neuen VB.NET-Projekts in VS.NET

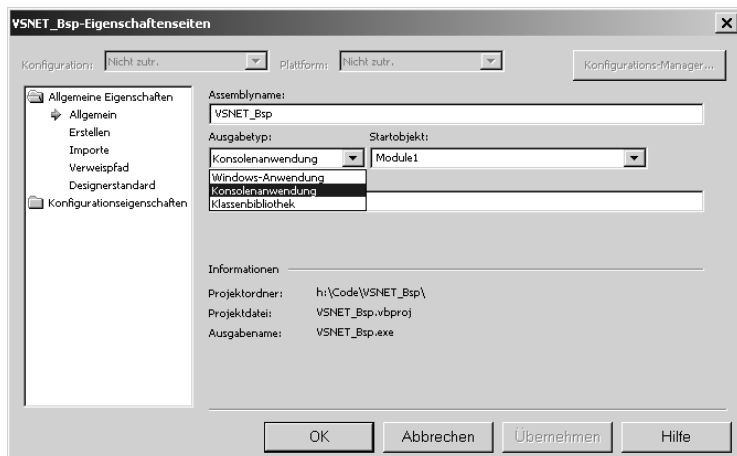


Abbildung 2.7: Änderung des Projekttyps im Eigenschaftsdialog eines Projekts

Projektelemente

Ein Projekt besteht aus Elementen verschiedener Typen. Elemente sind in zwei Kategorien gegliedert: Elemente für *Web-Projekte* und Elemente für *lokale Projekte*. Die in der folgenden Tabelle genannten Einzelemente sieht man, wenn man zu einem bereits geöffneten Projekt ein neues Element hinzufügt.

Projektelemente

Elementkategorie	Beispiele für Elemente dieser Kategorie
Elemente für beide Kategorien (Web-Projekte und lokale Projekte)	Klasse (Class, <i>.vb</i> oder <i>.cs</i>) Komponentenkasse (Component Class, <i>.vb</i> oder <i>.cs</i>) Frameset (<i>.htm</i>) HTML-Seite (<i>.htm</i>) Modul (<i>.vb</i> oder <i>.cs</i>) Stylesheet (<i>.css</i>) Text-File (<i>.txt</i>) XML-Datei (<i>.xml</i>) XSD-Schema (<i>.xsd</i>) XSLT-Datei (<i>.xslt</i>) DataSet (<i>.xsd</i>) JScript-Datei für WSH (<i>.js</i>) VBScript-Datei für WSH (<i>.vbs</i>) XML-strukturierte Skriptdatei (<i>.wsf</i>)
zusätzliche Elemente für Web-Projekte	Active Server Page (<i>.asp</i>) Webkonfigurationsdatei (Web Configuration File, <i>.config</i>) Web Form (<i>.aspx</i>) Webdienst (Web Service, <i>.asmx</i>) Web-Benutzersteuerelement (<i>.ascx</i>) Globale Anwendungsklasse (<i>global.asax</i>) Dynamische Discovery-Datei (<i>.vdisco</i>) Statische Discovery-Datei (<i>.disco</i>)
zusätzliche Elemente für lokale Projekte	Windows Form (<i>.vb</i> oder <i>.cs</i>) Benutzersteuerelement (<i>.vb</i> oder <i>.cs</i>) Windows Service (<i>.vb</i> oder <i>.cs</i>)

Elementkategorien

Tabelle 2.1: Übersicht über die Visual Studio .NET-Projektelemente

Die einzelnen Elemente eines Projekts können in Verbesserung der VB6-IDE innerhalb eines Projekts in eine beliebige Ordnerhierarchie sortiert werden, die im Projektmappen-Explorer (engl. Solution Explorer) auch so angezeigt wird.

Visual Studio .NET-Projektmappen

Projektmappe und Projekte

Visual Studio kennt als oberstes Gliederungselement eine Projektmappe (engl. *Solution*). Eine Projektmappe enthält ein Projekt oder mehrere Projekte. Sie kann auch in einem speziellen Ordner mit Namen *Projektmappen-Elemente* (engl. *Solution Items*) direkt projektübergreifende Elemente enthalten. Die verschiedenen verfügbaren Elemente sind in zwei Elementkategorien (Web-Projekte und lokale Projekte) geordnet. Es gibt zahlreiche Elemente, die in beiden Kategorien vorhanden sind.

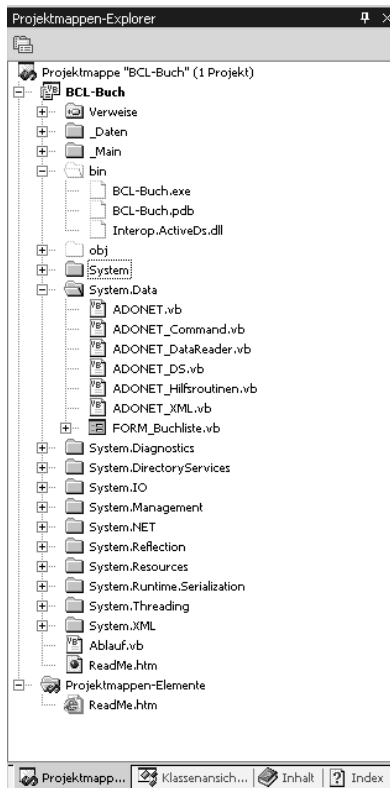


Abbildung 2.8: Ordnung in den Projektelementen im Projektmappen-Explorer

Verweise

Verweise

Um einen Typ nutzen zu können, benötigt der Visual Basic .NET-Compiler eine Referenz auf die Assembly, die diesen Typ implementiert. Diese Verweise sind nicht mehr wie in der VB6-IDE in einem Menü versteckt, sondern gut sichtbar in den Projektmappen-

pen-Explorer integriert (Ordner *Verweise*). Ein Projekt kann Verweise zu FCL-Namespaces, anderen .NET-Komponenten, COM-Komponenten oder anderen Visual Studio .NET-Projekten haben.

Für jeden Projekttyp ist eine bestimmte Menge von Verweisen auf verschiedene FCL-Namespaces automatisch vordefiniert. Über den Kontextmenüpunkt **VERWEIS HINZUFÜGEN** im Ordner *Verweise* können Verweise hinzugefügt werden.

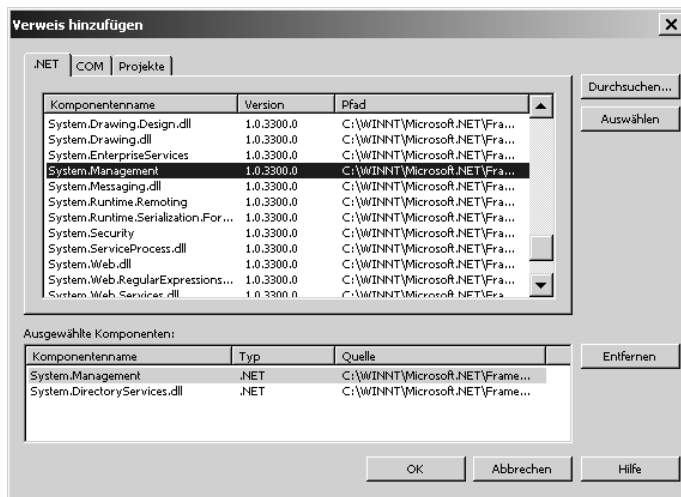


Abbildung 2.9: Hinzufügen eines Verweises auf eine .NET-Komponente

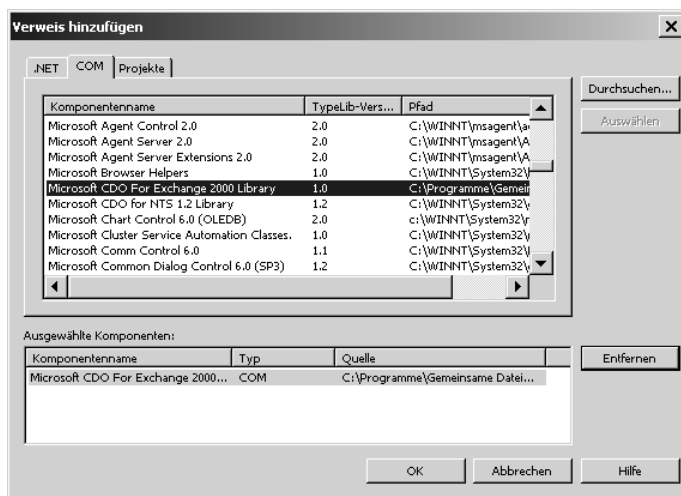


Abbildung 2.10: Hinzufügen eines Verweises auf eine COM-Komponente

Verweiseigenschaften

Referenzeigenschaften

Bei jedem Verweis kommt man durch Doppelklick zu einem Eigenschaftsfenster, das den Pfad zur Assembly, die Versionsnummer und den Strong Name anzeigt. Einzig die Eigenschaft **LOKALE KOPIE** kann verändert werden. Durch Auswahl von **JA** wird die Assembly in das `/bin`-Verzeichnis des lokalen Projekts kopiert.

Durch das Kontextmenü eines Verweises kann der Verweis entfernt werden.

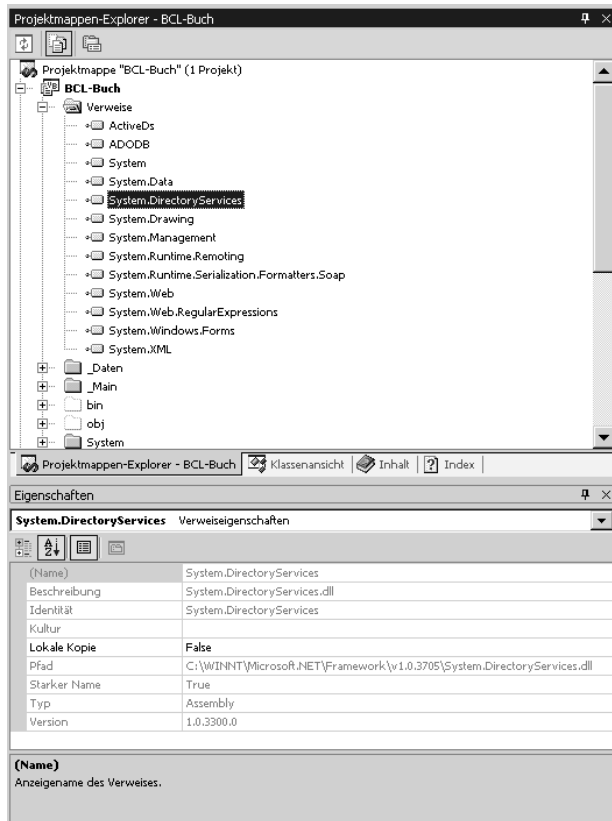


Abbildung 2.11: Verweise im Projektmappe-Explorer

Automatischer Import

Visual Studio .NET ermöglicht es für VB.NET-Projekte, in den Projekteigenschaften `Imports`-Anweisungen festzulegen, die für alle Quellcodedateien in dem Projekt gelten. Den entsprechenden Dia-

log finden Sie durch einen Klick auf den PROJEKTOPTIONEN-Button im Eigenschaftsfenster oder über das Kontextmenü des Projekts (Auswahl EIGENSCHAFTEN). In diesem Dialog können Sie nun die zu importierenden Namespaces festlegen.

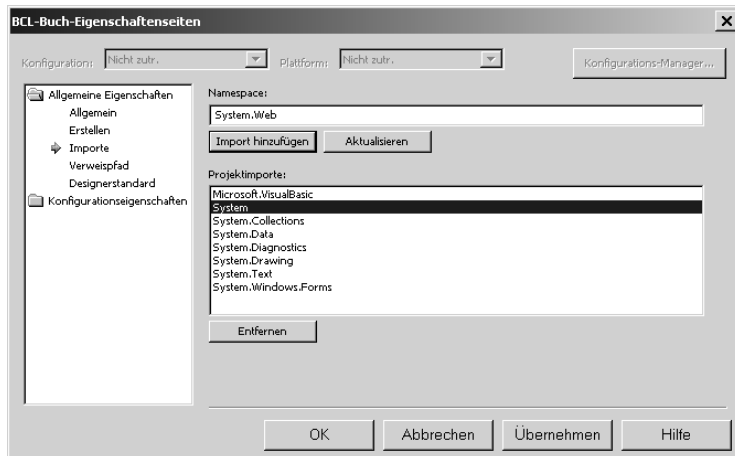


Abbildung 2.12: Die Eigenschaftsseiten des Projekts mit den eingebundenen Namespaces

Wenn Sie in diesem Fenster einen Namespace zu der Liste hinzufügen, gilt dieser standardmäßig für das gesamte Projekt, d.h. auch bei neuen Dateien wird er automatisch eingebunden.

Dieses Feature ist leider nicht für C# verfügbar. Dort müssen Sie die Namespaces manuell in jede Quelldatei einbinden.



2.3.3 Ausgaben in der VS.NET-Entwicklungsumgebung

Das aus der VB6-IDE bekannte Direktfenster ist nun auf zwei Fenster aufgeteilt:

- ▶ Das AUSGABEFENSTER liefert die Ausgaben des Compilers und Debuggers. Ein Programm kann mit `Debug.Write()` sowie `Debug.WriteLine()` Ausgaben in dieses Fenster machen. Der Entwickler kann aber selbst nichts darin eintippen.
- ▶ Das BEFEHLSFENSTER erlaubt die Eingabe von Befehlen an die Entwicklungsumgebung, z.B. die Ausführung von SUCHEN/ ERSETZEN.

Ausgabe

Befehle

**Debug.Write()
statt
Debug.Print()**

Debug ist kein eigenes Intrinsic Object von VB mehr, sondern Teil der FCL. Die Klasse heißt `System.Diagnostics.Debug`. Die Methode `WriteLine()` ist ein Shared Member und kann daher ohne Instanziierung der Klasse `Debug` verwendet werden. Insofern ist der Unterschied zur VB6-IDE eher akademischer Natur. Gravierend ist jedoch, dass die Ausgabemethode nicht mehr `Debug.Print()`, sondern `Debug.Write()` bzw. `Debug.WriteLine()` heißt. Der Unterschied zwischen `Debug.Write()` und `Debug.WriteLine()` besteht darin, dass im letzteren Fall die Ausgabe mit einem Zeilenumbruch versehen wird.

Ausgaben mit Console.WriteLine()

**DOS-
Anwendungen**

Beim Anwendungstyp Konsolenanwendung (*Console Application*) gehen Ausgaben mit `Console.WriteLine()` beim Start innerhalb der IDE in ein DOS-Fenster, das beim Programmstart geöffnet und beim Programmende direkt wieder geschlossen wird. Hier ist dafür zu sorgen, dass eine Benutzereingabe abgewartet wird, bevor das Programm endet, da man sonst – bei kurz laufenden Programmen – nicht viel von der Ausgabe hat.

**Windows-
Anwendungen**

Beim Anwendungstyp Windows-Anwendung (*Windows Application*) gehen die Ausgaben mit `Console.WriteLine()` beim Start innerhalb der IDE in das AUSGABEFENSTER.

2.3.4 Erstellung einer Anwendung in VS.NET

Anhand einer einfachen, fensterlosen Anwendung soll die Erstellung einer Anwendung in VS.NET gezeigt werden. Implementiert werden soll das schon im vorherigen Kapitel verwendete XML-Beispiel, dieses Mal wird die aus dem XML-Dokument extrahierte Information in einer Textdatei gespeichert. Verschiedene Ausgaben gehen in das DOS-Fenster. Für die Verwendung der Dateisystem-Klassen ist zusätzlich der Namespace `System.IO` einzubinden.

Das Beispiel aus diesem Kapitel finden Sie auf der CD-ROM unter `[CD:/code/Kapitel2_VB.NET/VSNET_Bsp]`.

```
' =====  
' VSNET_Bsp.sln  
' (C) Holger@Schwichtenberg.de  
' =====
```

```
Imports Microsoft.VisualBasic  
Imports System.Xml
```

```
Imports System.IO

Module VSNET_Bsp
    Sub main()
        Const EINGABE = "<Autor>" + _
            "<Vorname>Holger</Vorname>" + _
            "<Nachname>Schwichtenberg</Nachname>" + _
            "</Autor>"
        Const DATEINAME As String = _
            "d:\data\bcl_bsp.txt"

        Console.WriteLine("XML-Dokument laden...")
        Dim d As XmlDocument
        d = New XmlDocument()
        d.LoadXml(EINGABE)

        Console.WriteLine(_
            "Informationen extrahieren...")
        Dim vorname As XmlNode
        Dim nachname As XmlNode
        vorname = d.SelectSingleNode("*/Vorname")
        nachname = d.SelectSingleNode("*/Nachname")

        Console.WriteLine("Ausgaben erzeugen...")
        Dim Ausgabe As String = _
            "Name des Autors: " & _
            vorname.InnerXml & " " & nachname.InnerXml

        Console.WriteLine("Ausgabe speichern...")
        ' Datei öffnen
        Dim fs As FileStream = _
            New FileStream(DATEINAME, _
                FileMode.OpenOrCreate, FileAccess.Write)
        ' Stream öffnen
        Dim w As StreamWriter = New StreamWriter(fs)
        ' Zeilen schreiben
        w.WriteLine(Ausgabe)
        ' Schließen
        w.Close()
        fs.Close()

        Console.WriteLine("Datei beschrieben mit: " & _
            Ausgabe)

    End Sub
End Module
```

Listing 2.5: Beispielanwendung [VSNET_Bsp.sln]

Kompilierung

Um diese Anwendung in VS.NET zu einer Assembly zu kompilieren, führen Sie folgende Schritte aus:

- ▶ Starten Sie VS.NET.
- ▶ Erzeugen Sie ein Projekt vom Typ VISUAL BASIC/KONSOLEN ANWENDUNG.
- ▶ Wählen Sie als Namen »VS.NET_Bsp«.
- ▶ Wählen Sie ein beliebiges Verzeichnis zur Ablage der Projektdateien.
- ▶ Nach dem Klicken auf OK erstellt VS.NET das Projekt mit folgenden Elementen:
- ▶ *AssemblyInfo.vb*: Diese Datei enthält Meta-Attribute für die Beschreibungsfelder der Assembly-Datei.
- ▶ *Module1.vb*: Diese Code-Datei enthält folgenden Rumpf:

```
Module Module1
    Sub Main()
    End Sub
End Module
```

- ▶ Setzen Sie Verweise zu System, System.Data und System.Xml
- ▶ Fügen Sie den Programmcode in *Module1.vb* ein. Überschreiben Sie dabei den bisherigen Inhalt der Datei.
- ▶ Wenn Sie das Fenster AUFGABENLISTE einblenden, werden Sie sehen, dass dort ein »Build Error« angezeigt wird: »Sub Main was not found in 'VSNET_Bsp-Module1'«. Visual Basic kompiliert automatisch im Hintergrund nach dem Verlassen einer Zeile. Hier ist das Problem, dass in den Projekteigenschaften als Startobjekt »Module1« hinterlegt ist, in dem Beispiel das Startobjekt aber »VSNET_Bsp« heißt. Um dies zu ändern, gehen Sie in die Eigenschaften des Projekts und ändern Sie in der Rubrik ALLGEMEIN die Einstellung für STARTOBJEKT auf »VSNET_Bsp«.
- ▶ Jetzt können Sie das Projekt übersetzen (ERSTELLEN/PROJEKT MAPPE ERSTELLEN) und danach starten (DEBUGGEN/STARTEN). Sie können auch direkt DEBUGGEN/STARTEN aufrufen. Sofern Änderungen im Projekt vorgenommen wurden, ruft VS.NET immer vor dem Start den Compiler auf.

Es ist nicht notwendig, etwas an den vorkonfigurierten Verweisen zu ändern. Die Assembly System.Xml.dll gehört zu den in der Projektvorlage definierten Standardverweisen. Der Namespace System.IO ist implementiert in der System.dll, die sowieso immer eingebunden wird. Auch ein expliziter Verweis auf Microsoft.VisualBasic.dll ist nicht notwendig, da auch diese Assembly implizit immer eingebunden wird. VS.NET zeigt sie aber im Gegensatz zur System.dll nicht unter den Verweisen an.



3 C# vs. Visual Basic .NET

Dieses Buch behandelt sowohl C# als auch VB.NET. Genauer gesagt, die in diesem Buch aufgeführten Konzepte und Vorgehensweisen sind auf das .NET Framework bezogen und nicht auf eine bestimmte Sprache. Eigentlich könnte man alles auch mit Hilfe von Eiffel#, J# oder Cobol .NET durchführen. Alle Klassen der Framework Class Library (FCL) sind von jeder .NET-Sprache aus nutzbar.

Die beiden Hauptsprachen werden dennoch Visual Basic .NET und C# sein. Welche im Endeffekt überwiegen wird, lässt sich jetzt noch nicht abschätzen. Weil es aber in einem Buch immer sehr unübersichtlich wirkt, wenn das gleiche Beispiel in zwei unterschiedlichen Sprachen abgedruckt wird, obwohl eigentlich nur die Syntax und einige Operatoren anders sind, wird der Vergleich zwischen den beiden Sprachen in diesem Kapitel gezogen und im Rest des Buches (bis auf wenige Ausnahmen, bei denen es notwendig ist) Visual Basic .NET verwendet. Dieses Kapitel versetzt Sie in die Lage, die Code-Beispiele leicht von VB.NET nach C# zu konvertieren.

Ein Hinweis vorweg: Visual Basic .NET und C# sind sich ähnlicher, als man zunächst vermuten mag. Fast alles, was mit C# möglich ist, lässt sich ebenso gut mit Visual Basic .NET realisieren, und das Dank des .NET Frameworks ohne irgendwelche Geschwindigkeitseinbußen.



3.1 Grundlagen

Die Syntax von C# ist recht stark an C++ und an Java angelehnt. Die Syntax von Visual Basic hat sich seit den Urzeiten der Sprache Basic praktisch kaum verändert (trotz des gravierenden Sprungs zwischen Visual Basic 6.0 und Visual Basic .NET, vgl. Kapitel 2).

Die Unterschiede zwischen C# und Visual Basic sind nicht so gravierend, als dass man Programmcode nicht verhältnismäßig schnell umsetzen könnte. Die Syntax von Visual Basic .NET ist weit mehr beschreibend als die von C#, dafür sind manche Dinge in C# mit weniger Schreibarbeit umsetzbar.

3.1.1 C# ist case-sensitive

Case-sensitive *Case-sensitive* bedeutet, dass in C# anders als in VB.NET die Groß-/Kleinschreibung überprüft wird. Während in VB.NET die Variablenbezeichner `aValue` und `avalue` auf die gleiche Variable verweisen (wobei Visual Studio .NET automatisch eine Schreibweise, nämlich die beim ersten Auftreten der Variable, konsequent durchhält und Bezeichner entsprechend korrigiert), muss in C# auf korrekte Schreibweise geachtet werden. `aValue` und `avalue` würden in C# also zwei unterschiedliche Variablen bezeichnen.

Damit ist auch folgende Deklaration in C# durchaus möglich, in VB.NET würde sie zu einem Fehler führen:

```
Type type = Type.GetType("System.Int32",false,true);
```

Besonders für Umsteiger von Sprachen wie z.B. Delphi oder VB.NET ist dies ein Kriterium, das sehr oft zu Fehlern führt. Die Entwicklungsumgebung kann natürlich hier nicht eingreifen, da dem Programmierer freigestellt ist, wie er seine Variablen, Felder oder Methoden bezeichnet. Daher muss auf die Schreibweise geachtet werden.

PascalCasing Es gibt zwei Möglichkeiten, wie man Bezeichner schreiben kann, damit man nicht durcheinander gerät. Die eine ist das so genannte *PascalCasing*, bei dem ein Bezeichner grundsätzlich mit einem Großbuchstaben begonnen und jedes weitere Wort innerhalb des Bezeichners ebenfalls wieder mit einem Großbuchstaben begonnen wird.

camelCasing Die zweite Möglichkeit ist das *camelCasing*, bei dem ein Bezeichner mit einem Kleinbuchstaben begonnen und jedes weitere Wort innerhalb des Bezeichners wiederum mit einem Großbuchstaben begonnen wird. Beide Schreibweisen haben den Vorteil, dass die Bezeichner gut lesbar sind. Im Allgemeinen wird in C# eine Mischung aus diesen beiden Möglichkeiten verwendet. Wenn man sich danach richtet, was Visual Studio .NET automatisch tut, dann bilden sich folgende Punkte heraus:

- ▶ Bezeichner von Klassen werden in PascalCasing geschrieben.
- ▶ Bezeichner von Enums, Structs usw. werden ebenfalls in PascalCasing geschrieben.
- ▶ Bezeichner von Eigenschaften und Ereignissen werden ebenfalls in PascalCasing geschrieben.
- ▶ Bezeichner lokaler Variablen werden in camelCasing geschrieben.
- ▶ Private Felder werden in camelCasing geschrieben, öffentliche Felder in PascalCasing.

Natürlich bleibt es letztendlich Ihnen als Programmierer überlassen, wie Sie Ihre Bezeichner benennen. Eines ist jedoch offensichtlich geworden, wenn man sich die Beispielprogramme zum .NET Framework ansieht: Die ungarische Notation, die vor allem bei C++-Programmierern sehr beliebt ist, hat im .NET Framework ausgedient. Eine ähnliche Vorgehensweise wird lediglich noch bei Bezeichnern für Komponenten benutzt, damit deren Typ leichter erkennbar ist. Aber auch das ist nicht zwingend notwendig.

3.1.2 Anweisungen und Zeilen

Visual Basic ist von jeher zeilenorientiert, d.h. das Ende einer Zeile bestimmt auch das Ende einer Anweisung. Möchte man mehr als eine Anweisung in einer Zeile unterbringen, kann der Doppelpunkt zum Trennen verwendet werden; benötigt man aus Übersichtlichkeitsgründen mehrere Zeilen für einen Befehl, so werden diese Zeilen mittels eines Unterstrichs verbunden. Das kann im Quellcode dann so aussehen:

```
Dim aStr As String
aStr = "Dies ist ein sehr langer String, " + _
      "der nicht in eine Zeile passt. Die Anweisung " + _
      "muss daher auf mehr als eine Zeile verteilt " + _
      "werden."
```

Visual Basic .NET

In C# sieht die Sache ganz anders aus. C# benutzt das Semikolon, um eine Anweisung zu beenden. Damit können Anweisungen leichter auf mehrere Zeilen verteilt werden. Das gleiche Beispiel in C# programmiert würde so aussehen:

```
string aStr;
aStr = "Dies ist ein sehr langer String, " +
      "der nicht in eine Zeile passt. Die Anweisung " +
      "muss daher auf mehr als eine Zeile verteilt " +
      "werden.";
```

C#

Die Einrückung dient in beiden Fällen nur der Übersicht. Man sollte sich daran gewöhnen, auch wenn Visual Studio .NET für die Formatierung im Fall von Visual Basic .NET mehr Hilfe bereitstellt als für C#. Andererseits ist der C#-Programmierer damit auch flexibler. Es ist im Prinzip Geschmackssache, was man bevorzugt.

3.1.3 Kommentare

VB.NET-Kommentare

VB.NET-Kommentare

In beiden Sprachen gibt es die Möglichkeit, Quelltext zu kommentieren. C# ist in dieser Hinsicht weitaus flexibler als Visual Basic .NET. VB.NET ermöglicht nur einzeilige Kommentare, die entweder durch das Wort `REM` (steht für *Remark*) oder durch einen Apostroph eingeleitet werden. Kommentare gehen immer bis zum Zeilenende. Hier ein Beispiel für Kommentare in Visual Basic .NET:

```
REM Deklarationen
Dim aValue As Integer
Dim aStr As String

If aValue > 5 Then 'Vergleich ...
    Console.WriteLine("Wert ist größer 5")
End If
```

Auch die Version mit `REM` funktioniert sozusagen mitten in der Zeile. Sie könnten also auch schreiben:

```
If aValue > 5 Then REM Vergleich
```

C#-Kommentare

C#-Kommentare

In C# gibt es zwei Arten von Kommentaren, nämlich einzeilige und mehrzeilige. Weiterhin haben Sie die Möglichkeit, in C# XML-Kommentare einzufügen, die später durch Visual Studio .NET automatisch in eine Hilfedatei umgesetzt werden. Ein einfacher Kommentar bis zum jeweiligen Zeilenende wird eingeleitet durch zwei aufeinander folgende Schrägstriche.

```
//Das ist ein Kommentar
if ( aValue > 5 ) // das ist auch ein Kommentar
    Console.WriteLine( "aValue ist größer als 5" );
```

Kommentare über mehrere Zeilen werden durch die Zeichenfolge `/*` eingeleitet und durch `*/` beendet. Visual Studio .NET setzt für jede neue Zeile automatisch an den Zeilenanfang einen `*`. Damit sieht ein solcher mehrzeiliger Kommentar folgendermaßen aus:

```

/* Das ist ein mehrzeiliger Kommentar
 * Das Visual Studio setzt automatisch das Zeichen *
 * an den Anfang einer jeden neuen Zeile eines
 * solchen Kommentars. Abgeschlossen wird er durch */

```

XML-Kommentare

XML-Kommentare werden durch drei aufeinander folgende Schrägstriche eingeleitet. Visual Studio .NET komplettiert diese dann automatisch, weil es erkennt, dass dieser Kommentar als Beschreibung einer Klasse, einer Eigenschaft oder einer Methode gelten soll. Aus diesem Grund werden einige Informationen gleich mit eingefügt. Wenn Sie oberhalb einer Methodendeklaration drei Schrägstriche schreiben, vervollständigt Visual Studio .NET dies folgendermaßen:

XML-Kommentare

```

/// <summary>
///
/// </summary>
/// <param name="a"></param>
/// <param name="b"></param>
/// <returns></returns>
private int AddValues(int a, int b) {
    return a+b;
}

```

<Summary> bezeichnet hierbei die Zusammenfassung über die Bedeutung der jeweiligen Funktion. Die <Param>-Angaben stehen für jeden übergebenen Parameter, das Tag <returns> steht für den Rückgabewert. Wenn Sie entsprechende Eingaben machen und dann die Erstellung der Kommentardatei starten (über EXTRAS/KOMMENTARSEITEN ERSTELLEN), erzeugt Visual Studio .NET eine entsprechende Website mit Informationen über das Projekt und zeigt sie an. Bei den Angaben

```

/// <summary>
/// Die Methode addiert zwei Werte und
/// liefert das Ergebnis zurück
/// </summary>
/// <param name="a">Ein Int32-Wert</param>
/// <param name="b">Ein zweiter Int32-Wert</param>
/// <returns>Die Summe der beiden Werte</returns>
private int AddValues(int a, int b) {
    return a+b;
}

```

ergibt sich eine Website, wie sie in Abbildung 3.1 dargestellt ist.

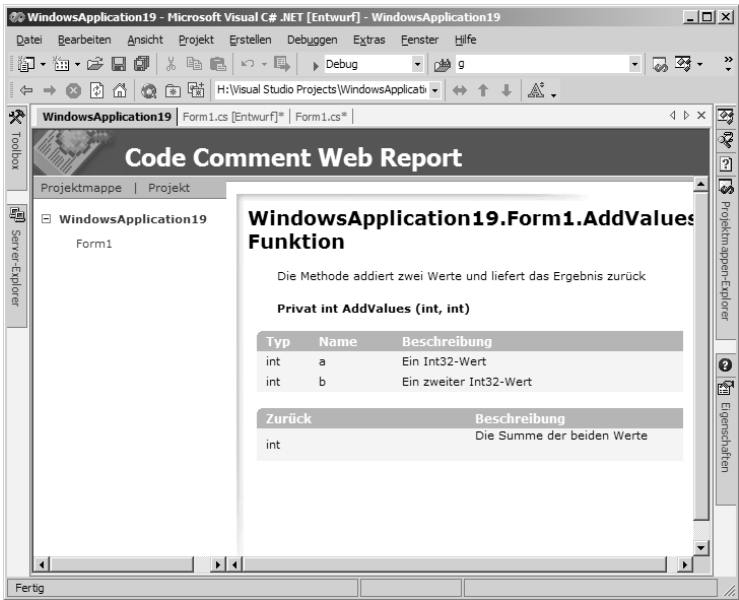


Abbildung 3.1: Die generierte Kommentarseite

Weitere XML-Kommentare

Die dargestellten XML-Elemente sind allerdings nicht die einzigen, die die Entwicklungsumgebung bereitstellt. Tabelle 3.1 zeigt weitere Möglichkeiten von XML-Kommentaren und die jeweilige Bedeutung.

XML	Bedeutung
<summary>	Zusammenfassung bezüglich des aktuellen Elements
<remarks>	Übersicht über eine Klasse oder ein Element
<param>	Information über einen der Parameter bei Methoden. Der Parameter wird über das Attribut name angegeben.
<returns>	Informationen über den Ergebniswert
<newpara>	Fügt einen neuen Absatz ein

Tabelle 3.1: XML-Kommentare



XML-Kommentare sind nur in C# implementiert, nicht in Visual Basic .NET. Es ist allerdings möglich, neue Sprachen ebenfalls mit der Möglichkeit von XML-Kommentaren zu versehen und dieses Feature zu nutzen.

3.1.4 Codeblöcke

In beiden Sprachen werden Codeblöcke benutzt, die mehrere Anweisungen zu einer einzigen zusammenfassen. In Visual Basic .NET ist das dadurch gelöst, dass Klartext benutzt wird. Eine If-Anweisung endet z.B. mit `End If`, wobei alles, was zwischen `If` und `End If` steht, als der relevante Codeblock angesehen wird. Eine `For Each`-Schleife endet mit `Next`, ebenso wie eine `For`-Schleife. Eine `Do`-Schleife endet mit `Loop`. Für den Programmierer ist somit immer klar ersichtlich, wo der Anweisungsblock endet, denn er kann es einfach ablesen.

**Codeblöcke in
VB.NET**

In C# werden Codeblöcke grundsätzlich in geschweifte Klammern eingeschlossen. Das gilt ausnahmslos für alle Codeblöcke, also auch für Methoden, Namespaces, Klassen (das wäre in Visual Basic .NET z.B. `End Class`) und auch Schleifen. Für einen Programmierer, der die C++-Syntax (oder die Java-Syntax) gewohnt ist, sind diese Codeblöcke leicht erkennbar; Programmierer, die von einer Sprache wie Visual Basic 6.0 umsteigen, werden sich anfangs ein wenig daran gewöhnen müssen. Gerade bei C# ist es daher enorm wichtig, mit Hilfe von Einrückungen die Übersicht zu behalten (wobei Visual Studio .NET wieder hilft).

Codeblöcke in C#

Auch hier ein kleines Beispiel:

```
If aValue > 5 Then
    x = x + 1
    aValue = aValue + x
End If
```

Visual Basic .NET

In C# werden alle Codeblöcke über geschweifte Klammern eingeschlossen. Die gleiche Anweisung würde also in C# folgendermaßen aussehen:

```
if ( aValue > 5 )
{
    x += 1;
    aValue = aValue + x;
}
```

C#

Oft gehen C#-Programmierer (wie auch C++- oder Java-Programmierer) so vor, dass sie die öffnende geschweifte Klammer in die gleiche Zeile schreiben wie die eigentliche Anweisung, der ein Codeblock folgt. Das Beispiel sähe dann so aus:

```
if ( aValue > 5 ) {
    x += 1;
    aValue = aValue + x
}
```

C#

3.1.5 Option-Statements

In Visual Basic .NET gibt es die Statements `Option Explicit`, `Option Strict` und `Option Compare`, die sowohl global für ein Projekt als auch nur für eine Datei eingestellt werden können. Von Hause aus werden diese Einstellungen allerdings von Visual Studio .NET etwas versteckt, Sie finden sie im Optionsdialog zu einem Projekt.

Option Explicit Mit `Option Explicit` können Sie angeben, ob Variablen in Visual Basic .NET-Projekten vor ihrer erstmaligen Verwendung deklariert werden müssen. Nicht deklarierte Variablen führen dann zu einem Fehler. Andersherum, mit der Einstellung `Option Explicit Off`, kann der Compiler aber nicht wissen, welchen Datentyp eine Variable annehmen soll, wenn sie nicht deklariert wurde. Aus diesem Grund ist der Datentyp einer nicht deklarierten Variable immer `Object`.

Option Strict `Option Strict` betrifft implizite Typkonvertierungen. Standardmäßig ist der eingestellte Wert »Off«. Wenn `Option Strict` eingeschaltet ist, müssen implizite Typkonvertierungen, z.B. von `Decimal` nach `Integer`, über `CInt()` oder `CType()` durchgeführt werden.

Option Compare `Option Compare` betrifft den Vergleich von Strings. Standardmäßig ist hier die binäre Vergleichsart eingestellt, bei der Strings anhand einer internen Sortierreihenfolge verglichen werden. In der Einstellung »On« wird die Sortierreihenfolge des aktuellen Gebietschemas verwendet.

Diese drei Optionen gibt es in C# nicht. C# verhält sich grundsätzlich entsprechend der Einstellungen

```
Option Explicit On
Option Strict On
Option Compare On
```

Teilweise muss man darauf achten, wenn man bestehende Visual Basic .NET-Projekte in C# umschreiben möchte, je nachdem wie diese Optionen eingestellt sind. Wenn Sie mit beiden Sprachen arbeiten, sollten Sie Visual Basic .NET so konfigurieren, dass es sich wie C# verhält.

3.1.6 Schlüsselwörter

Die Schlüsselwörter beider Sprachen unterscheiden sich. Teils besitzt Visual Basic .NET ein Schlüsselwort, das so nicht in C# enthalten ist; andererseits besitzt auch C# einige Schlüsselwörter, die

man in Visual Basic .NET nicht findet. Meist existiert jedoch ein Äquivalent mit gleicher Bedeutung. Tabelle 3.3 zeigt einige Schlüsselwörter im Überblick, erhebt jedoch keinen Anspruch auf Vollständigkeit. Sortiert ist die Tabelle nach den Namen der VB.NET-Schlüsselwörter.

VB.NET	C#	Bedeutung
AddHandler		Mit <code>AddHandler</code> wird eine Ereignisbehandlungsroutine einem Ereignis zugewiesen. <code>AddHandler</code> wird im Zusammenhang mit <code>AddressOf</code> verwendet, da nicht die Ereignisbehandlungsroutine selbst, sondern deren Adresse benötigt wird. Die Verwendung von <code>AddHandler</code> entspricht dem Abonnieren eines Ereignisses in C# über den <code>+=</code> -Operator.
Class	<code>class</code>	Deklariert eine neue Klasse
Const	<code>const</code>	Deklariert eine Konstante
Delegate	<code>delegate</code>	Deklariert einen Delegate
Event	<code>event</code>	Deklariert ein Ereignis
Friend	<code>internal</code>	Gibt an, dass das Feld zwar von außerhalb der Klasse, aber nur von innerhalb des aktuellen Projekts angesprochen werden kann
Handles		Gibt an, welches Ereignis die Methode bearbeiten soll. Bei Auftreten dieses Ereignisses wird dann die Methode aufgerufen. In C# wird die aufzurufende Methode der Liste von EventHandlern des Ereignisses unter Verwendung des <code>+=</code> -Operators hinzugefügt.
Inherits	<code>:</code>	Bezeichnet die Basisklasse bei der Klassendeklaration, von der abgeleitet wird
Implements	<code>:</code>	Gibt an, welche Schnittstelle von einer Klasse implementiert wird. In C# wird dies wie auch die Ableitung von einer Basisklasse durch den Doppelpunkt implementiert.
Imports	<code>using</code>	Importiert einen Namespace, so dass dessen Klassen benutzt werden können. Das <code>Imports</code> -Statement von Visual Basic .NET ist dabei auch in der Lage, Klassen zu importieren, deren statische Methoden dann ohne Angabe des Klassennamens benutzt werden können.
Interface	<code>interface</code>	Deklariert eine Schnittstelle
Me	<code>this</code>	Referenziert das aktuelle Objekt

Tabelle 3.2: Schlüsselwörter in C# und VB.NET

VB.NET	C#	Bedeutung
MustInherit	abstract	Deklariert eine Klasse, von der abgeleitet werden muss, bevor sie verwendet werden kann
MustOverride	abstract	Deklariert eine Methode, die in einer abgeleiteten Klasse implementiert werden muss
MyBase	base	Referenziert die Basisklasse der aktuellen Klasse, also die Klasse, von der abgeleitet wurde
New	New	Erzeugt eine neue Instanz einer Klasse
NotInheritable	sealed	Wird bei der Klassendeklaration verwendet. Deklariert die Klasse als nicht ableitbar, man bezeichnet diese auch als »versiegelte Klasse«.
NotOverridable	sealed	Deklariert eine Methode, die in einer abgeleiteten Klasse nicht überschrieben werden kann
Overloads		Überlädt eine Methode. Das funktioniert in C# automatisch ohne Angabe eines Modifizierers.
Overridable	virtual	Gibt an, dass diese Methode überschrieben werden kann
Overrides	override	Überschreibt eine Methode
Private	private	Deklaration als privat, von außen kann nicht zugegriffen werden
Protected	protected	Gibt an, dass das aktuelle Element nur von dieser Klasse oder von abgeleiteten Klassen angesprochen werden kann
Public	public	Deklaration als öffentlich
ReadOnly		Deklariert eine Eigenschaft nur zum Lesen
Static		Nur für lokale Variablen: Eine mit <code>Static</code> deklarierte Variable behält ihren Wert auch nach Beendigung der Methode.
Shadows		Verdeckt ein Element der Basisklasse. Wenn Sie ein Element (eine Methode oder ein öffentliches Feld) verdecken, kann auf dieses unter Verwendung der abgeleiteten Klasse nicht mehr zugegriffen werden.
Shared	static	Deklariert ein statisches Element einer Klasse. Diese Elemente sind Bestandteile der Klasse und werden nicht über den Namen einer Instanz, sondern über den Namen der Klasse selbst angesprochen.
Structure	struct	Deklariert einen zusammengesetzten Datentyp (einen Wertetyp, für Referenztypen wird eine Klasse deklariert)

Tabelle 3.2: Schlüsselwörter in C# und VB.NET (Fortsetzung)

VB.NET	C#	Bedeutung
WithEvents		Gibt eine Variable an, die ein Objekt enthält, dessen Ereignisse genutzt werden können. In C# gibt es hierfür ein anderes Konzept, d.h. man muss Code schreiben, um einem Delegate eine Ereignisbehandlungsroutine hinzuzufügen.
WriteOnly	extern	Deklariert eine Eigenschaft nur zum Schreiben Gibt an, dass diese Methode extern (also in einer DLL) deklariert ist
	unsafe	Deklariert einen Block mit unsicherem Code in C#
	volatile	Deklariert ein Objekt, das asynchron geändert werden kann

Tabelle 3.2: Schlüsselwörter in C# und VB.NET (Fortsetzung)

3.1.7 Operatoren

Viele der in VB.NET und C# verwendeten Operatoren sind gleich, die Unterschiede sind recht schnell aufgelistet.

Vergleich auf Gleichheit bei Zahlen

Vergleiche und Zuweisungen werden in Visual Basic .NET durch den gleichen Operator durchgeführt, nämlich durch das Gleichheitszeichen. In C#, das ja C++ und Java als syntaktisches Vorbild hat, wird für die Zuweisung ein einfaches Gleichheitszeichen verwendet, für den Vergleich allerdings ein doppeltes. Ein Vergleich in VB.NET könnte so aussehen:

```
Dim i As Integer
For i = 1 To 10
    If i = 5 Then
        Console.WriteLine( i )
    End If
Next
```

Visual Basic .NET

Das gleiche Programmfragment in C# sähe so aus:

```
for ( int i=1; i<=10; i++ ) {
    if ( i == 5 )
        Console.WriteLine( i );
}
```

C#

Die Unterschiede bzgl. der `for`-Schleife bzw. der `If`-Bedingung werden in einem späteren Abschnitt noch angesprochen. In diesem Fall geht es nur um den Vergleich, ob die Variable `i` tatsächlich den Wert 5 enthält, denn nur dann wird sie ausgegeben.

Vergleich zweier Objekte

Is-Operator Zum Vergleich, ob zwei Objekte identisch sind, muss in Visual Basic .NET der Is-Operator benutzt werden. In C# gibt es hier keine Änderung zum herkömmlichen Vergleichsoperator, der auch zum Vergleichen zweier Objekte benutzt werden kann. Ein Beispiel in Visual Basic .NET:

```
Visual Basic .NET    Dim arrList1 As New ArrayList()
                    Dim arrList2 As ArrayList = arrList1

                    If arrList2 Is arrList1 Then
                        Console.WriteLine( "True" )
                    End If
```

In C# würde dieser Vergleich folgendermaßen aussehen:

```
C#                ArrayList arrList1 = new ArrayList();
                    ArrayList arrList2 = arrList1;

                    if ( arrList1 == arrList2 )
                        Console.WriteLine( "true" );
```

Der folgende Vergleich führt allerdings in beiden Fällen zum Ergebnis False, da es sich nicht um das gleiche Objekt handelt, sondern lediglich um den gleichen Datentyp:

```
Visual Basic .NET    Dim arrList1 As New ArrayList()
                    Dim arrList2 As New ArrayList()

                    If arrList2 Is arrList1 Then
                        Console.WriteLine( "True" )
                    Else
                        Console.WriteLine( "False" )
                    End If
```

TypeOf und typeof Um zu kontrollieren, ob ein Objekt einem bestimmten Datentyp entspricht, muss in Visual Basic .NET der TypeOf-Operator verwendet werden. Einen solchen Operator gibt es auch in C#, dort aber mit einer anderen Bedeutung. Während der Einsatz von TypeOf (VB.NET) lediglich innerhalb eines Vergleichs möglich ist, liefert typeof (C#) ein Objekt vom Typ Type zurück, das den Datentyp repräsentiert.

Ein Vergleich darauf, ob ein Objekt eine Instanz eines bestimmten Datentyps ist, sähe in Visual Basic .NET folgendermaßen aus:

```
Visual Basic .NET    Dim arrList1 As New ArrayList

                    If TypeOf arrList1 Is ArrayList Then
                        Console.WriteLine( "True" );
                    End If
```

Das C#-Äquivalent benutzt hier den `is`-Operator:

```
ArrayList arrList1 = new ArrayList();  
  
if ( arrList1 is ArrayList )  
    Console.WriteLine( "true" );
```

C#

Solche Vergleiche sind vor allem dann notwendig, wenn man kontrollieren will, ob eine bestimmte Schnittstelle und damit eine bestimmte Funktionalität von einer Klasse implementiert wird. Sie können ebenfalls kontrollieren, ob eine Klasse von einer anderen Klasse abgeleitet ist. Der Vergleich auf den Basistyp einer Klasse liefert in diesem Fall ebenfalls `true`.

Immer den Wert `true` liefert ein Vergleich eines Referenztyps darauf, ob er vom Typ `System.Object` ist. `System.Object` ist der Basistyp aller Datentypen im .NET Framework und damit muss ein solcher Vergleich zwangsläufig mit einem wahren Ergebnis enden. Umgekehrt funktioniert es aber auch. Wenn Sie Boxing verwenden, eine Technik, bei der ein Datentyp in den Datentyp `Object` gekapselt (nicht konvertiert) wird, können Sie über einen solchen Vergleich herausfinden, welcher Datentyp in diesem Objekt versteckt ist.

**Vergleich mit
System.Object**

```
Dim arrList1 As New ArrayList()  
Dim o As Object  
  
arrList1.Add("Ein String")  
arrList1.Add(10)  
arrList1.Add("Noch ein String")  
  
For Each o In arrList1  
    If TypeOf o Is String Then  
        Console.WriteLine( "String" )  
    Else  
        Console.WriteLine( "Integer" )  
    End If  
Next
```

Visual Basic .NET

liefert die Ausgabe:

```
String  
Integer  
String
```

Vergleich auf Ungleichheit

Auf Ungleichheit wird in Visual Basic .NET mit der Kombination von Größer- und Kleiner-Zeichen verglichen. C# lehnt sich hier ebenfalls wieder an die Syntax von C++ an und benutzt die Zeichenfolge `!=`. Das entspricht in Visual Basic .NET dem Operator `<>`:

Visual Basic .NET

```
Dim i As Integer
For i = 1 To 10
    If i <> 5 Then
        Console.WriteLine( i )
    End If
Next
```

Das gleiche Programmfragment in C#:

C#

```
for ( int i=1; i<=10; i++ ) {
    if ( i != 5 )
        Console.WriteLine( i );
}
```

Bedingte Zuweisung

IIf In Visual Basic .NET gibt es eine nützliche Funktion namens `IIf`, mit der eine einfache Zuweisungsart ermöglicht wird. Der erste Parameter ist ein Ausdruck, der einen booleschen Wert ergibt. Der zweite Parameter ist das Ergebnis, das zurückgeliefert wird, wenn der Ausdruck `True` ergibt; der dritte Parameter wird zurückgeliefert, wenn der Ausdruck `False` ergibt.

Bedingte Zuweisung In C# ist dies wieder auf typische C++-Weise gelöst: Auch hier gibt es die Möglichkeit einer *Bedingten Zuweisung*, bei der es sich allerdings nicht um eine Methode handelt, sondern um einen Bestandteil der Sprachdefinition selbst. Die C#-Version ist ein wenig kryptischer zu lesen – wenn man sich aber daran gewöhnt hat, ist sie recht übersichtlich.

Die Syntax der bedingten Zuweisung in C# sieht folgendermaßen aus:

```
x = <Ausdruck> ? <Ergebnis wahr> : <Ergebnis falsch>;
```

Zum Vergleich ein Beispiel in Visual Basic .NET und C#:

Visual Basic .NET

```
Public Function CheckValue(ByVal Value As Integer) _
    As String
    Return CStr(IIf(Value > 5, "Ja", "Nein"))
End Function
```


Das gleiche Fragment in C#:

```
public string checkValue( int Value ) {  
    return (Value > 5)? "Ja":"Nein";  
}
```

C#

Während – wie zu sehen ist – in C# direkt der richtige Datentyp zurückgeliefert werden kann, muss bei Verwendung von `IIf` in Visual Basic .NET noch konvertiert werden. `IIf` liefert das Ergebnis immer als Wert vom Typ `Object` zurück, eine Konvertierung ist also notwendig. Standardmäßig erledigt Visual Basic .NET das automatisch, da `Option Strict` auf »Off« steht, die Konvertierung wird aber dennoch vorgenommen.

3.2 Deklarationen

3.2.1 Klassen und Module

Die Art der Programmierung mit Visual Basic hat sich stark geändert. War es vorher noch so, dass sich Module und Klassen durch die Art des Zugriffs unterscheiden haben und in einem Modul deklarierte Variablen global für das gesamte Programm gültig waren, so hat ein Modul unter .NET eine andere Bedeutung. Auch gibt es keine globalen Funktionen oder Subs mehr, diese müssen immer Bestandteil einer Klasse sein und heißen dann Methoden. Über den Modifizierer `shared` können Methoden (wie auch Variablen) aber auch zu einem Bestandteil der Klasse gemacht werden, so dass der Zugriff statt auf Objektebene auf Instanzebene stattfindet.

Ein Modul in .NET ist eine ausführbare Datei (EXE oder DLL), die zumindest eine Klasse enthält. Der Unterschied zwischen einer Assembly und einem Modul ist der, dass eine Assembly noch ein Manifest beinhaltet, das sie selbstbeschreibend macht. Eine Assembly besitzt auch die Endung `.exe` oder `.dll`, kann aber mehrere Module enthalten.

Module

Klassen sind so etwas wie die Grundlage der Programmierung im .NET Framework. Ohne Klassen funktioniert nichts. Globale Variablen, wie sie aus Visual Basic 6 bekannt waren, gibt es in dieser Form nicht mehr, ebenso wenig wie allgemein gültige Funktionen oder Subs. Diese müssen nun immer Bestandteil einer Klasse sein.

Klassen

Die Felder, Eigenschaften und Methoden einer Klasse werden auch unter dem Oberbegriff *Member* zusammengefasst.



Wenn Sie Members einer Klasse mit dem Modifizierer `shared` (oder `static` in C#) versehen, können Sie globale Methoden oder Variablen simulieren. Da der Zugriff auf Klassenebene geschieht, können alle Programmbestandteile, die den entsprechenden Namespace, in dem die Klasse deklariert ist, eingebunden haben, auf diese Members zugreifen. Da keine Instanz der Klasse erzeugt wird, verhalten sich die Members dann wie globale Funktionen, Subs oder Variablen.

Die Definition einer Klasse erfolgt in beiden Sprachen über das reservierte Wort `Class`. Wie in Visual Basic üblich wird das Ende einer Klasse durch eine End-Anweisung (`End Class`) angezeigt. In C# werden die Member der Klasse in geschweifte Klammern eingefasst.

Visual Basic .NET

```
Class myClass
    Private aField As Integer
    Private aSecondField As Integer
End Class
```

Die gleiche Deklaration in C# sieht folgendermaßen aus:

```
C#    class myClass {
        private int aField
        private int aSecondField
    }
```

Um mit einer Klasse arbeiten zu können, müssen Sie zunächst eine Instanz der Klasse, ein Objekt, erzeugen. Das geschieht durch den Aufruf des Konstruktors über das Schlüsselwort `New`. Mehr über Konstruktoren und Destruktoren finden Sie in Abschnitt 3.2.5.

3.2.2 Variablen und Felder

Syntax VB.NET

VB.NET verwendet für Deklarationen lokaler Variablen das Schlüsselwort `Dim`, das es in C# nicht gibt. Stattdessen wird in C# der Datentyp vor den Bezeichner geschrieben. Damit ergibt sich folgende Syntax für die Deklaration einer Variablen in VB.NET:

```
Dim Variablenname As Datentyp
```

Im Falle von Feldern, also Attributen einer Klasse, kann das Schlüsselwort `Dim` auch weggelassen werden; in diesem Fall werden lediglich die Sichtbarkeitsstufe und mögliche weitere Optionen dargestellt, also z.B.:

```
Public Shared Variablenname As Datentyp
```

Die Deklaration einer lokalen Integer-Variablen sähe also in VB.NET so aus:

```
Dim aValue As Integer
```

In C# gibt es kein spezielles Schlüsselwort für die Deklaration von Variablen oder Feldern. Die Syntax für eine lokale Variable sieht folgendermaßen aus:

Syntax C#

```
Datentyp Variablenname;
```

Bei der Deklaration von Feldern ist es das Gleiche, allerdings werden auch hier Sichtbarkeitsstufen und Ähnliches davor geschrieben:

```
public static Datentyp Variablenname;
```

Damit sieht die gleiche Deklaration einer Variablen vom Typ Integer in C# so aus:

```
int aValue;
```

Die Namen Integer in VB.NET und int in C# sind Aliase für den gleichen Datentyp, der im .NET Framework als Int32 im Namespace System deklariert ist. Alternativ könnte man also auch den korrekten Namen verwenden. Wer oft mit unterschiedlichen Programmiersprachen arbeitet, z.B. mit C# eine DLL erstellt und diese dann in einem VB.NET-Projekt weiterverwendet, ist sogar gut beraten, die Originalnamen zu verwenden, da die Verwechslungsgefahr damit quasi nicht mehr vorhanden ist:

Aliase

```
Dim aValue As System.Int32
```

bzw. in C#:

```
System.Int32 aValue;
```

3.2.3 Methoden

Auch die Deklaration von Methoden einer Klasse gestaltet sich in VB.NET wesentlich anders als in C#. Während VB.NET durch die Schlüsselwörter Sub und Function Methoden mit und ohne Ergebniswert unterscheidet, wird bei C# immer ein Ergebniswert angegeben und die Deklaration ist ebenfalls immer die gleiche. Methoden, die kein Ergebnis liefern, haben den speziell dafür vorgesehenen Wert void.

Sub und Function

Dadurch gibt es in C# natürlich auch kein Schlüsselwort, das eine Methode mit Ergebniswert von einer ohne Ergebniswert unterscheidet. Die Syntax der Deklaration einer `Sub` sieht in Visual Basic .NET folgendermaßen aus:

Syntax VB.NET `Private Sub Bezeichner ( [ Parameter As Datentyp, ... ] )`

In C# ist auch hier wieder weniger Schreibarbeit gefordert:

Syntax C# `private void Bezeichner ( [ Datentyp Parameter ] )`

Exit Sub und Exit Function Was Visual Basic .NET mehr hat, sind die Anweisungen `Exit Sub` bzw. `Exit Function` zum Verlassen einer Methode. In C# gibt es kein direktes Äquivalent, allerdings kann hier über die Anweisung `return` eine Methode verlassen werden, auch wenn sie kein Ergebnis zurückliefert.

Return Das Ergebnis einer `Function` bzw. einer Methode mit Ergebniswert wird in beiden Sprachen mit `Return` (bzw. `return` in C#) zurückgeliefert. Alternativ kann der Ergebniswert in Visual Basic .NET auch dem Funktionsnamen zugewiesen werden.

```
Private Function AddValues(ByVal a As Integer, _
    ByVal b As Integer) As Integer
    Return a+b
End Function
```

oder

```
Private Function AddValues(ByVal a As Integer, _
    ByVal b As Integer) As Integer
    AddValues = a+b
End Function
```

3.2.4 Eigenschaften

Properties Sowohl Visual Basic .NET als auch C# unterstützen die Definition von Datenmitgliedern in Klassen (also Attributen) durch Eigenschaften (engl. Properties) statt durch Felder (siehe Abschnitt 3.2.2).

In Klassen kann man Properties definieren, um dem Anwender den direkten Zugriff auf die Daten des Objekts zu verweigern, z. B. wenn ein Wert nur ausgelesen, nicht aber verändert werden soll. Anstatt direkt auf die Daten zuzugreifen, wird bei einer Property sowohl beim Lesen als auch beim Schreiben Programmcode ausgeführt und man hat die volle Kontrolle über die erlaubten Werte und die Art der Speicherung. Dabei definiert man in der Regel

zusätzlich zu der Property ein `private` Feld für die tatsächliche Speicherung des Wertes.

In Visual Basic .NET ist die Vorgehensweise normalerweise so, dass man den Namen des privaten Feldes mit einem Unterstrich beginnt und der Eigenschaft den gleichen Namen wie dem Feld gibt, auf das man zugreifen will. Natürlich ohne Unterstrich. Das ist notwendig, weil VB.NET nicht case-sensitive arbeitet, also nicht zwischen Groß- und Kleinschreibung unterscheidet. Die Deklaration einer Eigenschaft geschieht über das reservierte Wort `Property`:

VB.NET

```
private _feld As String ' Das Feld selbst
```

Visual Basic .NET

```
Public Property Feld() As String ' Die Eigenschaft
    Get
        Return _feld
    End Get
    Set(ByVal Value As String)
        _feld = Value
    End Set
End Property
```

Die Zugriffsmöglichkeiten und noch einige weitere Dinge werden wie in beiden Sprachen üblich durch Modifikatoren angegeben. Eine Eigenschaft, die nur zum Lesen vorgesehen ist, wird mit Hilfe des Modifizierers `ReadOnly` deklariert:

ReadOnly

```
Public ReadOnly Property Feld() As String
    Get
        Return _feld
    End Get
End Property
```

Visual Basic .NET

Die Methoden `Get` und `Set` werden von Visual Studio .NET automatisch eingefügt, nachdem Sie die erste Zeile der Deklaration geschrieben haben. In diesem Fall hilft Visual Studio .NET also beträchtlich weiter. Wenn Sie z. B. den Modifizierer `ReadOnly` angegeben haben, wird auch nur eine `Get`-Methode generiert; haben Sie `WriteOnly` angegeben, wird lediglich eine `Set`-Methode generiert mit einem Parameter `Value`, der den gleichen Datentyp hat wie in der Property-Deklaration angegeben.

**Get und Set-
Methoden**

In C# gestaltet sich die Deklaration ein wenig anders. Zunächst können Feldbezeichner und Eigenschaftsbezeichner die gleichen sein, da C# ja zwischen Groß- und Kleinschreibung unterscheidet. Man schreibt einfach das Feld in camelCasing, die Eigenschaft in

C#

PascalCasing (wie auch vorher beschrieben). C# kennt das reservierte Wort `Property` nicht und benötigt es auch nicht. Stattdessen wird gleich mit dem Codeblock begonnen.

Allerdings ist die Hilfe des Visual Studio .NET hier faktisch nicht vorhanden, es wird davon ausgegangen, dass der Programmierer weiß, was er tut. Die Methoden `get` und `set` (sie heißen in C# genauso, allerdings klein geschrieben) müssen Sie selbst schreiben:

```
C#      private string feld;

          public string Feld {
              get { return feld; }
              set { feld = value; }
          }
```

Value Für manche, die sich noch nicht mit den geschweiften Klammern angefreundet haben, mag diese Syntax etwas verwirrend sein, da in der Methode `Set` ebenfalls ein Parameter `value` verwendet wird, dessen Datentyp dem entspricht, der in der Deklaration der Eigenschaft angegeben wurde. Dieser ist aber nicht deklariert, d.h. man muss wissen, dass es ihn automatisch gibt und man ihn verwenden kann. Hat man sich einmal daran gewöhnt, ist die C#-Syntax ein wenig übersichtlicher, da der relevante Text sofort sichtbar ist.

Readonly-Eigenschaft Wenn Sie mit C# eine Eigenschaft deklarieren wollen, die nur zum Lesen verfügbar ist, so benötigen Sie dafür keinen Modifizierer, Sie lassen einfach die `Set`-Methode weg. Eine Nur-Lese-Eigenschaft würde also in C# folgendermaßen aussehen:

```
C#      public string Feld {
          get { return feld; }
      }
```

In C# haben Sie also – was Eigenschaften betrifft – die bessere Übersichtlichkeit, aber die schlechtere Mithilfe des Visual Studio .NET.

3.2.5 Konstruktoren und Destruktoren

Visual Basic .NET ist jetzt vollständig objektorientiert, d.h. Sie können eigene Klassen erstellen und von vorhandenen Klassen erben. C# wurde von Haus aus bereits objektorientiert aufgebaut und orientiert sich in der Syntax an Sprachen, die bereits in der Vergangenheit objektorientierte Eigenschaften und Möglichkeiten beinhalteten.

Konstruktoren

Konstruktoren werden aufgerufen, wenn eine neue Instanz einer Klasse erzeugt werden soll. In beiden Fällen lautet das entsprechende Schlüsselwort `New` (in VB.NET groß geschrieben, in C# klein). Der Konstruktor selbst wird in VB über eine Methode mit dem Namen `New` deklariert, die Parameter enthalten kann (aber nicht muss) und keinen Ergebniswert beinhaltet (also eine `Sub`). Die Deklaration sieht dann folgendermaßen aus:

```
Public Class AnyClass
```

Visual Basic .NET

```
    ' Methoden, Felder, Eigenschaften, Ereignisse  
    ' sind hier nicht mit angegeben
```

```
Public Sub New ()  
    'Das ist der Konstruktor  
End Sub
```

```
End Class
```

Eine Klasse kann natürlich auch mehrere Konstruktoren besitzen. In C# sieht die Deklaration ein wenig anders aus, sie ist hier von C++ abgeleitet. Der Konstruktor besteht in einer Methode mit dem gleichen Namen wie die Klasse. Es wird in diesem Fall kein Ergebniswert (also auch nicht `void`) angegeben. Die gleiche Klasse würde also in C# folgendermaßen deklariert:

```
public class AnyClass {
```

C#

```
    // Methoden, Felder, Eigenschaften, Ereignisse  
    // sind hier nicht mit angegeben
```

```
    public AnyClass() {  
        //Dies ist der Konstruktor  
    }  
}
```

Auch hier ist die Angabe mehrerer Konstruktoren möglich.

Destruktoren (Finalizer)

Im Bezug auf Destruktoren muss man im .NET Framework etwas vorsichtig sein. Ein C++-Programmierer versteht unter einem Destruktor eine Methode, die dann aufgerufen wird, wenn das Objekt seinen Gültigkeitsbereich verlässt – der Aufruf ist also vorhersehbar. Im .NET Framework ist das nicht so. Destruktoren werden hier von der automatischen Garbage-Collection aufgeru-

**Destruktoren und
Garbage Collec-
tion**

fen, zu einem Zeitpunkt, der nicht vorhersehbar ist. Microsoft selbst sagt sogar, dass es unter gewissen Umständen möglich ist, dass ein Destruktor nie aufgerufen wird.

Finalizer Destrukturen heißen daher im .NET Framework auch Finalizer. Die Bezeichnung Destruktor wurde lediglich beibehalten, um C++-Programmierern einen schnelleren Umstieg zu ermöglichen. Sie wurde auch erst in der Beta 2 eingeführt, in Beta 1 hieß es ausnahmslos Finalizer. Daher kommt auch die Bezeichnung für die entsprechende Methode, die Sie in VB.NET deklarieren können und die dann aufgerufen wird, wenn ein Objekt freigegeben wird. Sie heißt `Finalize()`, hat ebenfalls keinen Ergebniswert und auch keine Parameter (da sie von der Garbage-Collection aufgerufen wird und nicht aus dem Programm heraus).

Die Deklaration eines Finalizers (oder Destrukturen) in VB.NET sieht also folgendermaßen aus:

```
Visual Basic .NET      Public Class AnyClass

                          ' Methoden, Felder, Eigenschaften, Ereignisse
                          ' sind hier nicht mit angegeben

                          Public Sub New ()
                              'Das ist der Konstruktor
                          End Sub

                          Public Sub Finalize()
                              'Das ist der Finalizer
                          End Sub
                          End Class
```

C# ist wieder an C++ angelehnt. Dort wird wieder der Klassenname als Bezeichner für den Destruktor verwendet, diesmal aber mit dem Umkehrzeichen (der Tilde, ~). Auch diese Methode hat keinen Ergebniswert.

```
C#      public class AnyClass {

                // Methoden, Felder, Eigenschaften, Ereignisse
                // sind hier nicht mit angegeben

                public AnyClass() {
                    //Dies ist der Konstruktor
                }

                public ~AnyClass() {
                    //Dies ist der Destruktor
                }
            }
```


Normalerweise werden Sie in eigenen Klassen keinen Destruktor benötigen. Sie müssen ihn nicht deklarieren – Sie dürfen aber. Allerdings empfiehlt es sich nicht, weil man die Garbage-Collection damit sehr leicht durcheinander bringen kann. Wenn Sie wirklich Aufräumarbeiten durchführen wollen – und nur dazu sind Finalizer wirklich geeignet –, sollten Sie die Methode `Dispose()` bzw. die Methode `Close()` implementieren. Auch in den Klassen des .NET Frameworks ist das so gemacht. Lassen Sie die Garbage-Collection einfach arbeiten, betrachten Sie sie als eine Black-Box und verwenden Sie Finalizer respektive Destruktoren nur dann, wenn Sie genau wissen, was Sie tun, und wenn es unbedingt notwendig ist.

3.3 Programmier Techniken

3.3.1 Schleifen

Schleifen sind Konstruktionsmerkmale einer jeden Programmiersprache. In Visual Basic .NET und C# sind vier verschiedene Schleifentypen implementiert, die unterschiedlich flexibel sind. Im Großen und Ganzen ist aber mit der einen Sprache auch das möglich, was die andere Sprache kann.

For-Schleife

Mit der For-Schleife können Anweisungen oder Anweisungsblöcke mehrfach wiederholt werden. In Visual Basic .NET nutzt die For-Schleife eine Laufvariable, mit deren Hilfe die Anzahl der Durchläufe festgelegt wird. C# ist da flexibler, d.h. die Schleifenkonstruktion folgt einem ganz anderen Ansatz und bietet damit mehr Möglichkeiten.

Die Syntax einer For-Schleife in Visual Basic .NET stellt sich wie folgt dar:

```
For Laufvariable = Start To Ende [Step Schrittweite]
    Anweisungen
[ Exit For ]
    Anweisungen
Next
```

Syntax VB.NET

Es wird also immer eine Laufvariable benötigt, mit der gezählt wird und der ein Anfangs- bzw. Endwert angegeben wird. *Schritt-*

weite gibt an, um wie viel die Laufvariable jeweils erhöht oder erniedrigt werden soll, da die *For*-Schleife ja auch rückwärts zählen kann.

Die Anweisung `Exit For` weist Visual Basic .NET an, die *For*-Schleife zu verlassen; die Anweisung `Next` startet den nächsten Schleifendurchlauf. Ein typisches Beispiel für eine *For*-Schleife wäre das folgende:

```
Visual Basic .NET      Dim i As Integer
                        For i = 1 To 1000
                          Console.WriteLine( i )
                        Next
```

Das Äquivalent von C# ist in dieser Hinsicht ein gutes Stück flexibler, da es nicht von einer Laufvariable abhängig ist, aber eine benutzen kann. Die entsprechende Syntax in C# sieht wie folgt aus:

```
Syntax C#           for ( [Initialisierung]; [Abbruchbedingung]; [Iteration] ) {
                        Anweisungen
                        [ break; ]
                        [ continue; ]
                        Anweisungen
                      }
```

Initialisierung legt den Startwert einer etwaigen Laufvariablen fest. *Abbruchbedingung* legt fest, bei welcher Bedingung die Schleife verlassen werden soll, nämlich dann, wenn die Abbruchbedingung `False` wird. *Iteration* erhöht oder erniedrigt die Laufvariable um einen bestimmten Wert. Bei den Anweisungen kann es sich um eine einzelne Anweisung oder um einen Anweisungsblock handeln, der dann aber in geschweifte Klammern gesetzt werden muss. Die geschweiften Klammern werden nicht benötigt, wenn es sich lediglich um eine Anweisung handelt, die ausgeführt werden soll.

Das gleiche Beispiel in C#-Syntax sieht wie folgt aus:

```
C#                  for ( int i=1; i<=1000; i++ )
                      Console.WriteLine( i );
```

In dieser Schleife wird die Laufvariable direkt im Schleifenkopf deklariert und ist somit auch nur innerhalb der Schleife gültig. Das funktioniert in VB nicht, da hier die Variable vorher deklariert werden muss (`Option Explicit On` immer vorausgesetzt). Die Schleife wird abgebrochen, wenn *i* größer wird als 1000, und bei jedem Schleifendurchlauf wird *i* um eins erhöht.

Initialisierungsbedingung und Iteration sind aber optional, d.h. auch folgende Schleife würde funktionieren:

```
int i = 1;
for ( ; i<=1000 ; ) {
    Console.WriteLine( i );
    i++;
}
```

C#

Hier ist keine Initialisierung innerhalb der Schleife angegeben und auch der Iterator fehlt. Dennoch funktioniert es, weil die Variable `i` innerhalb der Schleife hochgezählt wird und somit die Abbruchbedingung greift. Auch folgende Schleife würde funktionieren:

```
bool doStatements = true;
int i = 1;
for (;doStatements;) {
    Console.WriteLine( i );
    i++;
    doStatements = ( i<= 1000 );
}
```

C#

In diesem Fall wird überhaupt keine Schleifenvariable benutzt, allerdings sehr wohl eine Abbruchbedingung, nämlich `doStatements`. Sobald diese `False` wird, wird die Schleife entsprechend abgebrochen und mit den nachfolgenden Anweisungen weitergemacht. In diesem Fall verhält sich die `for`-Schleife wie eine `while`-Schleife.

Auch in C# gibt es eine Abbruchbedingung für Schleifen, die allerdings nicht nur für die `for`-Schleife sondern auch für andere Schleifen und Verzweigungsstrukturen gilt. Es handelt sich um die Anweisung `break`. Wenn man diese benutzt, kann man mit Hilfe der `for`-Schleife auch eine Endlosschleife bilden und diese abhängig von einer Bedingung innerhalb der Schleife abbrechen:

```
int i = 1;
for (;;) {
    Console.WriteLine( i );
    i++;
    if ( i > 1000 )
        break;
}
```

C#

Ein weiterer Schleifendurchlauf wird durch die Anweisung `Continue` gestartet. Auch diese Anweisung gilt für jede Schleife, nicht allein für `for`. Wenn `Continue` eingesetzt wird, werden die Anweisungen im Schleifenkopf durchlaufen, d.h. es wird sowohl die Laufvariable erhöht (oder erniedrigt) als auch die Abbruchbedingung geprüft:

Continue

```

C#      for ( int i=1; i<=10; i++ ) {
           if ( (i % 2) == 0 )
               continue;
           Console.WriteLine( i );
       }

```

Do-Schleife

Die *do*-Schleife in C# ist nicht so mächtig wie die von Visual Basic .NET. Während in VB.NET sowohl am Anfang als auch am Ende Bedingungen angegeben werden können, begnügt die *do*-Schleife in C# sich mit der Möglichkeit, am Ende der Schleife einen Ausdruck zu kontrollieren und dementsprechend die Anweisungen innerhalb der Schleife zu wiederholen oder auch nicht. Damit ist die *Do*-Schleife in Visual Basic .NET sowohl als abweisende Schleifenform als auch als nicht-abweisende Schleifenform verwendbar, die *do*-Schleife in C# ist immer nicht-abweisend.

Die Syntax der *Do*-Schleife ist in Visual Basic .NET dementsprechend umfangreich:

Syntax VB.NET

```

Do { While | Until } <Bedingung>
    Anweisungen
[ Exit Do ]
    Anweisungen
Loop

```

bzw.

```

Do
    Anweisungen
[ Exit Do ]
    Anweisungen
Loop { While | Until } <Bedingung>

```

In C# ist die Syntax wesentlich einfacher:

Syntax C#

```

Do {
    Anweisungen
    [ break; ]
    [ continue; ]
    Anweisungen
} while ( <Bedingung> );

```

Auch hier (wie bei allen Schleifen) besitzt Visual Basic .NET wieder ein *Exit*-Kommando, das in C# der Anweisung *break* entspricht. Ebenfalls möglich ist in C# die Anweisung *continue*, mit der der nächste Schleifendurchlauf gestartet wird. Die Bedingungen werden auch hier wieder kontrolliert.

Der Vergleich beider Konstruktionen mittels eines Beispiels ist hier nur bei einer Art des Einsatzes möglich, nämlich als nicht-abweisende Schleife. In Visual Basic .NET sieht das dann so aus:

```
Dim i As Integer = 0;
Do
    Console.WriteLine( i )
    i = i + 1
Loop While i<10
```

Visual Basic .NET

In C# programmiert sieht dies folgendermaßen aus:

```
int i = 0;
do {
    Console.WriteLine( i );
    i++;
} while ( i<10 );
```

C#

While-Schleife

Die `while`-Schleife verhält sich in beiden Sprachen gleich und unterscheidet sich lediglich ein klein wenig in der Syntax. Hier die Syntax für Visual Basic .NET:

```
While <Bedingung>
    Anweisungen
[ Exit While ]
    Anweisungen
End While
```

Syntax VB.NET

Und hier die Syntax für C#:

```
while ( <Bedingung> ) {
    Anweisungen
    [ break; ]
    [ continue; ]
    Anweisungen
}
```

Syntax C#

Die geschweiften Klammern werden nicht benötigt, wenn es sich lediglich um eine Anweisung handelt, die ausgeführt werden soll.

foreach-Schleife

Die `foreach`-Schleife dient in beiden Sprachen dazu, über die Werte einer Collection oder eines Arrays zu iterieren. Die `foreach`-Schleife empfiehlt sich dann, wenn Sie auf alle Elemente eines Arrays nacheinander zugreifen müssen, z.B. zur Ausgabe oder um eine Kontrolle durchzuführen, aber nicht wissen, wie viele Elemente enthalten sind.

Auch die `foreach`-Schleife unterscheidet sich lediglich in der Syntax. Hier zunächst die Syntax von Visual Basic .NET:

Syntax VB.NET

```
For Each <Element> In <Array>
    Anweisungen
[ Exit For ]
    Anweisungen
Next [ <Element> ]
```

Die Syntax von C# liest sich ähnlich:

Syntax C#

```
foreach ( <Element> in <Array> ) {
    Anweisungen
    [ break; ]
    [ continue; ]
    Anweisungen
}
```

Ein Unterschied besteht noch in der Deklaration der Elementvariablen. C# erlaubt hier wieder wie z.B. auch bei der `for`-Schleife eine Deklaration im Schleifenkopf, was mit Visual Basic .NET nicht möglich ist. Eine Implementation würde z.B. so aussehen:

C#

```
public void WriteAll( string[] sArray ) {
    foreach ( string s in sArray ) {
        Console.WriteLine( s );
    }
}
```

In Visual Basic .NET müsste hier die Variable `s` vor der Schleife deklariert werden:

Visual Basic .NET

```
Public Sub WriteAll(ByVal sArray As String())
    Dim s As String

    For Each s In sArray
        Console.WriteLine( s )
    Next
End Sub
```

3.3.2 Verzweigungen

If-Anweisung

Die `If`-Anweisung verhält sich in beiden Sprachen gleich und unterscheidet sich auch hier lediglich durch die Syntax. In Visual Basic .NET werden die zu kontrollierenden Bedingungen nach dem Schlüsselwort `If` angegeben und mit `Then` beendet, worauf der Anweisungsblock folgt und danach ein abschließendes `End If`.

C# verwendet wieder geschweifte Klammern zur Blockbildung, die zu kontrollierenden Bedingungen werden in runden Klammern nach dem Schlüsselwort `If` angegeben. Hier die Syntax von Visual Basic .NET:

```
If <Bedingung> Then
    Anweisungen
End If
```

Syntax VB.NET

Und hier die Syntax von C#:

```
if ( <Bedingung> ) {
    Anweisungen
}
```

Syntax C#

Die geschweiften Klammern werden nicht benötigt, wenn es sich nur um eine Anweisung handelt, die ausgeführt werden soll.

Select-Anweisung (switch-Anweisung)

Visual Basic .NET implementiert zum Vergleichen eines Werts mit mehreren möglichen Ergebnissen die `Select Case`-Anweisung, deren Syntax sich folgendermaßen gestaltet:

```
Select [ Case ] Ausdruck
    [ Case Ergebnis 1:
        [ Anweisungen ] ]
    [ Case Ergebnis 2:
        [ Anweisungen ] ]
    ...
    [ Case Else
        [ Else-Anweisungen ] ]
End Select
```

Syntax VB.NET

Ein Beispiel für die `Select Case`-Anweisung wäre der Vergleich einer Variablen auf verschiedene mögliche Werte:

```
Private Sub DoSomething(ByVal DoWhat As Integer)
    Select Case DoWhat
        Case 1
            Console.WriteLine("Wert 1 wurde gewählt")
        Case 2
            Console.WriteLine("Wert 2 wurde gewählt")
        Case 3
            Console.WriteLine("Wert 3 wurde gewählt")
        Case Else
            Console.WriteLine("Ein anderer Wert wurde gewählt")
    End Select
End Sub
```

Visual Basic .NET

In C# existiert für derartige Vergleiche die `switch`-Anweisung, deren Syntax allerdings ein wenig von der der `Select-Case`-Anweisung abweicht. Die Syntax der `switch`-Anweisung sieht folgendermaßen aus:

Syntax C#

```
switch (Ausdruck)
{
    case Ergebnis:
        Anweisungen
        Sprunganweisung
    [default:
        Anweisungen
        Sprunganweisung]
}
```

Zwischen diesen beiden Anweisungen gibt es einige Unterschiede. Die mächtigere Variante ist ohne Zweifel die VB.NET-Variante. Grundsätzlich kann sie zwar das Gleiche, allerdings wesentlich einfacher.

Case-Block Einer der größten Unterschiede ist, dass die Anweisungen innerhalb eines `Case`-Blocks mit einer Sprunganweisung abgeschlossen werden müssen. Dabei kann es sich sowohl um einen Sprung zu einer anderen `Case`-Anweisung handeln als auch um einen Sprung aus dem `switch`-Statement heraus (mittels `break`).

Weiterhin kann in C# nur immer ein Ergebnis pro `Case`-Anweisung ausgewertet werden. Sollen mehrere Werte zum gleichen Ergebnis führen, kann man die Sprunganweisung einfach weglassen (bzw. alle Anweisungen), wodurch das Programm einfach von einer zur nächsten `Case`-Anweisung durchfällt, bis Anweisungen gefunden werden und auch die obligatorische Sprunganweisung.

Mehrere Auswertungen In Visual Basic .NET ist es möglich, mehrere Ergebniswerte innerhalb einer `Case`-Anweisung auszuwerten. Die einfachste Möglichkeit ist dabei, die gewünschten Werte einfach durch Kommata zu trennen. Das folgende Beispiel in Visual Basic .NET und C# führt zum gleichen Ergebnis und zeigt den Unterschied auf:

Visual Basic .NET

```
Private Sub DoSomething(ByVal DoWhat As Integer)
    Select Case DoWhat
        Case 1, 2, 3
            Console.WriteLine("Wert ist kleiner als 4")
        Case 4
            Console.WriteLine("Wert ist gleich 4")
        Case 5, 6, 7, 8
            Console.WriteLine("Wert ist größer 4")
        Case Else
```



```

        Console.WriteLine("Wert ist größer 8")
    End Select
End Sub

```

Umgesetzt in C# ist hier etwas mehr Schreibarbeit erforderlich:

```

private void DoSomething(int doWhat) {
    switch ( doWhat ) {
        case 1:
        case 2:
        case 3:
            Console.WriteLine("Wert ist kleiner 4");
            break;
        case 4:
            Console.WriteLine("Wert ist gleich 4");
            break;
        case 5:
        case 6:
        case 7:
        case 8:
            Console.WriteLine("Wert ist größer 4");
            break;
        default:
            Console.WriteLine("Wert ist größer 8");
            break;
    }
}

```

C#

Die Funktion ist dabei genau die gleiche. In Visual Basic .NET könnte die Anweisung auch folgendermaßen geschrieben werden (wieder mit dem gleichen Ergebnis):

```

Private Sub DoSomething(ByVal DoWhat As Integer)
    Select Case DoWhat
        Case 1 To 3
            Console.WriteLine("Wert ist kleiner als 4")
        Case 4
            Console.WriteLine("Wert ist gleich 4")
        Case 5 To 8
            Console.WriteLine("Wert ist größer 4")
        Case Else
            Console.WriteLine("Wert ist größer 8")
    End Select
End Sub

```

Visual Basic .NET

Über das Schlüsselwort **To** wird angegeben, dass alle Werte gültig sind, die zwischen den beiden angegebenen Werten liegen. Das vereinfacht die Programmierung der **Select Case**-Anweisung in einigen Fällen enorm. Eine weitere Möglichkeit bietet die Anweisung **Is** in Verbindung mit einem Vergleichsoperator. Beachten Sie, dass **Is** innerhalb einer **Select Case**-Anweisung eine andere Bedeu-

To und Is

tion besitzt als der gleichnamige Vergleichsoperator. Die folgende Anweisung verwendet den `Is`-Operator:

Visual Basic .NET

```
Private Sub DoSomething(ByVal DoWhat As Integer)
    Select Case DoWhat
        Case 1 To 3
            Console.WriteLine("Wert ist kleiner als 4")
        Case 4
            Console.WriteLine("Wert ist gleich 4")
        Case 5 To 8
            Console.WriteLine("Wert ist größer 4")
        Case Is > 8
            Console.WriteLine("Wert ist größer als 8")
        Case Else
            Console.WriteLine("Wert ist kleiner 1")
    End Select
End Sub
```

Zuletzt ist es auch noch möglich, die verschiedenen Möglichkeiten innerhalb eines `Case`-Statements zusammenzufassen (also gemeinsam zu benutzen). Das würde dann in etwa so aussehen:

Visual Basic .NET

```
Select Case DoWhat
    Case 1, 2, 3 To 7, 9, Is > 15
        Console.WriteLine("Ein Wert ... ")
    Case 8, 10 To 15
        Console.WriteLine("Ein anderer Wert ...")
End Select
```

3.3.3 Casting

Als *Casting* bezeichnet man die explizite (erzwungene) Umwandlung eines Datentyps in einen anderen. Casting wird dann benötigt, wenn z.B. ein numerischer Datentyp mit großer Bit-Breite (z.B. der Datentyp `System.Int64`) in einen ebenfalls numerischen Datentyp kleinerer Bit-Breite (z.B. `System.Int32`) konvertiert werden soll. Der Grund für die Notwendigkeit, diese Umwandlung zu erzwingen, ist, dass Daten verloren gehen können.



Standardmäßig zeigt Visual Basic .NET hier keinen Fehler an, da die Konvertierung eines großen Datentyps in einen kleinen mittels Option Strict auf »Off« erlaubt ist. Wenn Sie in den Projektoptionen Option Strict auf »On« stellen, sehen Sie auch, dass Visual Studio .NET einen Fehler anzeigt, wenn eine explizite Konvertierung erforderlich ist.

In Visual Basic .NET wird dieses Casting durch die Anweisungen `CInt`, `CDBl`, `CStr` usw. durchgeführt, allgemein kann auch `CType` ver-

wendet werden, mit dem in einen beliebigen Datentyp gecastet werden kann.

In C# wird ein anderes Konzept verfolgt: Hier wird der Datentyp, in den umgewandelt werden soll, einfach in Klammern vor den Datentyp geschrieben, der den Wert enthält. Zum Vergleich wieder ein kleines Codefragment zunächst in Visual Basic .NET:

```
Dim long1 As Long = 5000
Dim int1 As Integer
```

Casting in C#

```
int1 = CInt(long1)
```

Visual Basic .NET

Und nun die C#-Variante:

```
long long1 = 5000;
```

C#

```
int i = (int)long1;
```

3.3.4 With-Anweisung

Die With-Anweisung ist eine nützliche Vereinfachung für den Programmierer. Wenn Sie auf Elemente eines Objekts zugreifen wollen, müssen Sie dieses Element normalerweise qualifizieren, d.h. den Namen des Objekts mit angeben. Die With-Anweisung ermöglicht es, den Namen des Objekts nur einmal angeben zu müssen und die Elemente innerhalb der With-Anweisung einfach zu benutzen. Die Qualifikation der Elemente erfolgt dann mittels eines Punkts.

With

```
With Button1
    .Width = 150
    .Height = 120
    .Left = 100
    .Top = 100
    .BackColor = Color.AliceBlue
End With
```

Visual Basic .NET

Die With-Anweisung existiert leider nur in Visual Basic .NET, in C# ist es grundsätzlich notwendig, die vollständige Bezeichnung anzugeben. Das ist wirklich ein Manko von C#.

In der Beta 1 des Visual Studio .NET war die with-Anweisung in der Tat auch in C# implementiert. Leider wurde dieses nützliche Statement aus unerfindlichen Gründen wieder aus der Sprachdefinition entfernt; in jedem Fall wäre es eine sinnvolle Erweiterung in der nächsten Version.



3.3.5 Arrays

Eindimensionale Arrays

Deklaration in VB.NET

Grundsätzlich haben Arrays in beiden Sprachen die gleiche Bedeutung. Allerdings verhalten sie sich – was die Deklaration und die Initialisierung angeht – ein wenig unterschiedlich. Beiden gemein ist, dass das erste Element eines Arrays den Index 0 hat. Bei der Deklaration allerdings wird in C# die Anzahl der enthaltenen Elemente angegeben, in Visual Basic .NET der Index des letzten Elements. Damit liefert die Deklaration

```
Dim myArray(6) As Integer
```

ein Array mit sieben Elementen, nämlich von 0 bis 6. Die Deklaration

```
Dim myArray() As Integer = New Integer(6) {}
```

tut das Gleiche unter Verwendung des `New`-Operators. Wenn dieser verwendet werden soll, darf allerdings vorher keine Größe angegeben worden sein. Weiterhin dürfen Sie nicht die geschweiften Klammern am Ende der Zeile vergessen, die auch dann benötigt werden, wenn das Array nicht gleich mit Werten vorbelegt wird.

Die Verwendung von `New` hat den Vorteil, dass die im Array enthaltenen Werte direkt initialisiert werden (in diesem Fall mit 0), wogegen die Größenangabe im ersten Beispiel es erforderlich macht, dass später den einzelnen Elementen auch Werte zugewiesen werden, bevor man das Array benutzt (also Werte ausliest).

Deklaration in C#

In C# läuft die Deklaration ein wenig anders ab. Hier wird grundsätzlich mit dem `new`-Operator gearbeitet. Die Deklaration des gleichen Arrays würde also in C# folgendermaßen aussehen:

```
C#    int[] myArray = new int[7];
```

Beachten Sie den Unterschied: in C# Angabe der Anzahl der Elemente, in Visual Basic .NET Angabe des höchsten Indexes. Das führt sehr oft zu Fehlern bei der Umstellung, vor allem wenn ein Programm von Visual Basic .NET auf C# umgestellt werden soll – in diesem Fall hätte das Array in C# nämlich einen Eintrag zu wenig.

Vorbelegung mit Werten

Arrays können bereits bei der Deklaration mit Werten vorbelegt werden. Das geschieht in beiden Sprachen durch Angabe der Werte in geschweiften Klammern nach der Deklaration:

```
Dim myArray() As Integer = New Integer(2) {1, 2, 3}
```

Visual Basic .NET

oder in C#

```
int[] myArray = new int[3] {1, 2, 3};
```

C#

In C# ist auch noch eine Abkürzung möglich, indem man die Deklaration mit `new` weglässt. In beiden Fällen ist es nicht notwendig, eine Größe anzugeben, da diese bereits durch die hinter der Deklaration angegebenen Werte bestimmt wird. Wird aber (im Falle von Visual Basic .NET) eine Größenangabe gemacht, muss diese mit der Anzahl der angegebenen Einträge übereinstimmen. Obiges Array könnte also auch so deklariert werden:

```
Dim myArray() As Integer = New Integer() {1, 2, 3}
```

Visual Basic .NET

oder in C#:

```
int[] myArray = {1, 2, 3};
```

C#

Mehrdimensionale Arrays

Die Deklaration mehrdimensionaler Arrays gestaltet sich bis auf den Unterschied in der Syntax in beiden Sprachen gleich. Ein zweidimensionales Array würde in Visual Basic .NET folgendermaßen deklariert:

Deklaration

```
Dim myArray(,) As Integer = New Integer(,) {}
```

Visual Basic .NET

und in C# folgendermaßen:

```
int[,] myArray = new int[,];
```

C#

Auch hier ist es möglich, sowohl die Größe gleich anzugeben bzw. das Array gleich vorzubelegen. Dabei müssen die Elemente in beiden Sprachen in geschweiften Klammern geschachtelt werden:

Vorbelegung mit Werten

```
int[,] myArray2 = new int[,] {{1,2},{1,3}};
```

C#

oder in Visual Basic .NET:

```
Dim myArray(,) As Integer = New Integer(,) _  
    {{1,2},{1,3}}
```

Visual Basic .NET

Jagged Arrays

Auch für jagged Arrays, also Arrays innerhalb von Arrays, gelten in beiden Sprachen die gleichen Voraussetzungen, wieder mit der geänderten Syntax.

Redimensionierung

In Visual Basic .NET ist es möglich, Arrays zu redimensionieren, ohne die enthaltenen Werte zu verlieren. Das geschieht durch die Anweisung `ReDim Preserve`. In C# ist das nicht möglich, hier müssen Sie, wenn Sie ein Array vergrößern wollen, zunächst die enthaltenen Werte irgendwohin kopieren und dann die Größe des Arrays durch eine neue `New`-Anweisung ändern.

Ein großer Nachteil von `ReDim Preserve` ist allerdings die Geschwindigkeit, denn diese Anweisung ist recht langsam. Falls Sie dynamische Arrays benötigen, sollten Sie die Klasse `System.Collections.ArrayList` verwenden, die speziell für diesen Zweck im .NET Framework enthalten ist.

3.4 Unterschiede in der Entwicklungsumgebung

Codevervollständigung

Zwar ist Visual Studio .NET die Entwicklungsumgebung sowohl für Visual Basic .NET als auch für C#, doch gibt es für jede Sprache einen Editor. Diese Editoren weisen beachtliche Unterschiede auf.

Bei der Arbeit mit Visual Basic .NET merkt man gleich, dass der Editor sehr viel Hilfestellung gibt. Die automatische Vervollständigung arbeitet nicht nur bei `Sub`- und `Function`-Konstruktionen, sondern auch bei Schleifen und anderen Blockkonstrukten. Außerdem wird der Inhalt eines Codeblocks automatisch eingezogen und korrekt positioniert.

Der C#-Editor hilft an dieser Stelle nicht so sehr mit. Geschriebener Code bleibt an der Stelle, an der Sie ihn geschrieben haben, und auch die schließende geschweifte Klammer wird nicht automatisch gesetzt. Sie haben jedoch auch hier die Möglichkeit, Codezeilen automatisch an die richtige Stelle zu setzen. Wenn Sie eine Codezeile in einen Block eingefügt haben, dann die schließende geschweifte Klammer entfernen und nochmals setzen, wird auch in C# der Codeblock korrekt positioniert.

Während der VB.NET-Editor vergessene runde Klammern beim Aufruf oder bei der Deklaration einer Methode vervollständigt, erzeugt Visual Studio .NET für C#-Code in diesem Fall dafür lieber drei Fehlermeldungen. Man könnte daher sagen, C# macht mehr Mühe.

Die Entwicklungsumgebung bietet für deklarierte und typisierte Variablen beim Kodieren eine Liste aller verfügbaren Methoden und Attribute an (Funktion MEMBER AUFLISTEN). Da C# eine Sprache ist, die zwischen Groß- und Kleinschreibung unterscheidet, versagt der Code-Editor logischerweise sofort seine Mithilfe, wenn die Schreibweise nicht stimmt (während sie in Visual Basic .NET einfach angepasst wird). Das ist damit zu begründen, dass es in C# ebenso gut eine Variable mit der gegebenen Schreibweise geben könnte. Es ist also teilweise etwas störend, hat seinen Ursprung aber in der Unterscheidung der Groß- und Kleinschreibung.

List Members

Ein Trick besteht darin, die Funktion MEMBER AUFLISTEN über das Kontextmenü oder die Tasten `[STRG] + [J]` manuell aufzurufen, während der Cursor auf einem Wort steht, dessen Schreibweise offensichtlich nicht stimmt. Voraussetzung dafür, dass MEMBER AUFLISTEN funktioniert, ist aber in jedem Fall die Einbindung der entsprechenden Komponente als Referenz in das Projekt. Visual Studio .NET nutzt dann die .NET-Meta-Daten bzw. eine COM-Typbibliothek zur Ermittlung der verfügbaren Klassen und Klassenmitglieder.

Regions

In allen Code-Editoren kann der Entwickler die Implementierung von einzelnen Klassen und Unterrouinen durch die Zeichen + und – ähnlich wie bei den Verzeichnissen im Windows Explorer verbergen. Es bleibt dann nur der Klassen- bzw. Unterrouinen-Rumpf stehen mit einem Verweis, dass die Implementierung verborgen ist. Mit der Compiler-Direktive `#Region...#Endregion` kann der Entwickler auch eigene zuklappbare Code-Abschnitte definieren. Bei diesen Regions gibt es auch Unterschiede. Für C# gilt, dass Sie die Bezeichnung direkt hinter das Wort `Region` schreiben können, Visual Basic .NET erfordert es, die Bezeichnung in Anführungszeichen zu setzen.

Kompilieren im Hintergrund

Der Code-Editor prüft bereits beim Verlassen einer Codezeile diese auf Syntax- und Typfehler. Eine geschlängelte Linie weist Sie auf einen Fehler hin, wenn Sie die Maus darüber ziehen, wird der Fehler auch im Klartext angezeigt. Parallel dazu erzeugt Visual Studio .NET einen Eintrag im »Task-Window«, in dem die Compiler auch die Kompilierungsfehler melden. Der Entwickler wird nicht mehr wie bei der automatischen Syntaxprüfung in Visual Basic 6.0 durch

Dialogboxen gestört. Zudem bemerkt Visual Studio .NET wesentlich mehr Fehler, z.B. dass eine Variable oder eine verwendete Unterroutine undeklariert ist. Aber auch hier ist der Visual Basic .NET-Editor besser als der C#-Editor: Die Hintergrundkompilierung erkennt hier eine hinsichtlich des Datentyps fehlerhafte Zuweisung sofort nach dem Verlassen der Zeile. Der C#-Editor bemerkt den Typfehler bzw. die Verbesserung eines Typfehlers erst beim Kompilieren.

Zugriff auf die Namespaces

Globale Namespace-Einbindung

Um einen Typ in seiner Kurzschreibweise (also ohne den kompletten Namespace) verwenden zu können, ist ein Import des Namespaces notwendig. Der Import erfolgt in C# mit `using` und in Visual Basic .NET mit `Imports`. Während in C# alle Namespaces in jeder einzelnen Quellcodedatei explizit mit `using` eingebunden werden müssen, ermöglicht es Visual Studio .NET für VB.NET-Projekte, in den Projekteigenschaften `Imports`-Anweisungen festzulegen, die für alle Quellcodedateien in dem Projekt gelten. Den entsprechenden Dialog finden Sie durch einen Klick auf den PROJEKTOPTIONEN-Button im Eigenschaftsfenster oder über das Kontextmenü des Projekts (Auswahl EIGENSCHAFTEN). In diesem Dialog können Sie nun die zu importierenden Namespaces festlegen.

Automatische Formatierung

Quelltext-Formatierung

Die Quelltext-Formatierung ist in Visual Basic .NET wesentlich ausgeprägter als in C#. Das hat mehrere Gründe, unter anderem weil Visual Basic .NET nicht zwischen Groß- und Kleinschreibung unterscheidet und sich so Schreibweisen z.B. von Namespaces einfach herausucht und sie so schreibt, wie sie ursprünglich deklariert wurden. Das gilt auch für Variablen, die ebenfalls grundsätzlich so formatiert werden, wie sie bei der Deklaration geschrieben wurden.

Die Programmierung mit Visual Basic .NET gestaltet sich dadurch für den einen oder anderen wesentlich schneller, weil man einfach drauflos schreiben kann. In C# ist das nicht möglich. Allerdings hat C# den Vorteil, dass die Formatierung zu einem gewissen Grad auch dem Programmierer überlassen wird. Ob die bessere IntelliSense-Hilfe ein Grund ist, auf C# als Sprache zu verzichten und stattdessen doch Visual Basic .NET zu benutzen, bleibt die Entscheidung jedes einzelnen.

4 Namespace System

Der Namespace `System` ist die Hauptwurzel der FCL-Namespace-Hierarchie. `System` enthält zahlreiche Unternamespaces, aber auch viele zentrale Klassen direkt. Dazu gehören insbesondere Datentypen des *Common Type Systems* (CTS) sowie elementare Funktionen wie der Zugriff auf die Umgebung eines Prozesses, auf Application Domains und die Kommandozeile (DOS-Fenster).

System

Der Namespace `System` enthält auch die Definition der Klasse `Object`, die Oberklasse aller Klassen im .NET Framework ist.



4.1 Klasse `System.Object`

Die Klasse `Object` ist die Basisklasse aller Klassen im .NET Framework. Alle Klassen erben von dieser Klasse, wobei das Erben nicht direkt erfolgen muss, sondern auch indirekt erfolgen kann: Die oberste Klasse der Vererbungshierarchie einer jeden Klasse muss `Object` sein. `System.Object` nimmt daher eine ähnliche Rolle wie die Schnittstelle `IUnknown` in COM ein.

**Absolute
Basisklasse**

`Object` ist keine abstrakte Basisklasse, d.h. man kann Instanzen dieser Klasse erzeugen. Instanzen von `Object` werden für das späte Binden eingesetzt.

CLS-Regel 23

Die besondere Bedeutung von `System.Object` ist in Regel 23 der Common Language Specification (CLS) im Rahmen der Common Language Infrastructure (CLI), dem ECMA-Standard zu .NET, festgelegt: »*System.Object ist CLS-konform. Jede andere CLS-konforme Klasse muss von einer CLS-konformen Klasse erben.*«

Anders ausgedrückt: Jede .NET-Klasse erbt direkt oder indirekt von `Object`. Indirektes Erben bedeutet, dass eine .NET-Klasse die Bedingung, von `Object` zu erben, auch erfüllen kann, indem sie von einer anderen Klasse erbt, die (direkt oder indirekt) von `Object` erbt.

Vererbte Methoden

Vererbte Methoden

`Object` implementiert vier Methoden, die an alle .NET-Objekte vererbt werden. Die einzelnen erbenden Klassen sollten die meisten davon reimplementieren.

- ▶ `ToString()`
- ▶ `GetHashCode()`
- ▶ `Equals()`
- ▶ `GetType()`

Nicht vererbte Methoden

Nicht vererbte Methoden

Die Klasse `Object` besitzt zwei weitere Methoden, die allerdings `Protected` sind und daher nicht direkt, sondern aus Unterklassen heraus aufgerufen werden können.

- ▶ `Finalize()`
- ▶ `MemberwiseClone()`

Shared Member

Shared Member

Außerdem gibt es noch ein `Shared Member`, das direkt über den Klassennamen (auch von erbenden Klassen) aufgerufen werden kann:

- ▶ `ReferenceEquals()`

4.1.1 GetType()

Hierarchischer Klassenname

Die Methode `GetType()` liefert ein Objekt vom Typ `System.Type`, das den Datentyp repräsentiert, zu dem das Objekt gehört.

```
Dim t As Type = obj.GetType()
```

`GetType()` liefert stets den Datentyp des in einer Variablen enthaltenen Typs, nicht den Datentyp, mit dem die Variable deklariert wurde. Für ein Objekt `o` vom Typ `System.Object` liefert `o.GetType.FullName` die Zeichenkette »System.Object«. Wird einer Variablen

vom Typ `Object` eine Instanz einer anderen Klasse zugewiesen, dann liefert `GetType()` ein `Type`-Objekt für die Klasse des zugewiesenen Objekts.

```
Dim obj As Object = 15
out(obj.GetType.FullName) ' liefert System.Int32
```

Diese Methode muss in abgeleiteten Klassen nicht reimplementiert werden, da die CLR dafür sorgt, dass bei abgeleiteten Klassen das richtige `Type`-Objekt zurückgeliefert wird.

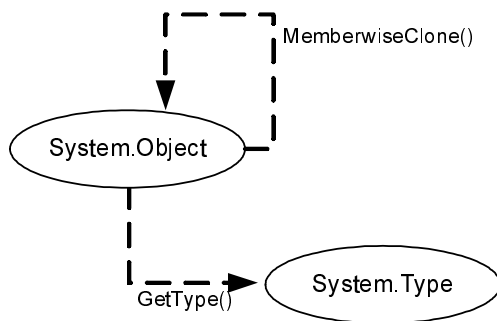


Abbildung 4.1: Objekthierarchie für `System.Object`

4.1.2 ToString()

Diese Methode liefert eine textliche Repräsentation des Wertes eines Objekts.

**Textliche
Repräsentation
des Wertes**

```
Dim s As String = obj.ToString()
```

Die Standardimplementierung in der Klasse `System.Object` ruft aber einfach `GetType().FullName` auf und liefert so den Namen der Klasse (inkl. Namespaces), nicht irgendeinen Wert, der den Inhalt der Instanz repräsentieren würde. Eine Reimplementierung in abgeleiteten Klassen ist also geboten.

In vielen FCL-Klassen ist diese Methode reimplementiert, so dass ein `String` zurückgeliefert wird, der den Objektinhalt repräsentiert.

Diese textliche Repräsentation soll eher zu Testzwecken bei Entwicklung und Fehlersuche dienen, um den Inhalt von Objekten für einen Menschen lesbar ausgeben zu können.



4.1.3 GetHashCode()

Hash-Wert Diese Methode liefert einen Hash-Wert (Integer), der die verschiedenen Werte, die ein Objekt enthält, in einem Wert repräsentiert.

```
Dim i As Int32 = Obj.GetHashCode()
```

Die Klasse `Object` stellt eine Standardimplementierung für `GetHashCode()` bereit, die jedoch unbedingt in abgeleiteten Klassen reimplementiert werden sollte. In der Standardimplementierung werden die Objekte fortlaufend innerhalb einer Application Domain gezählt. Eine Nummer kann jedoch nach Vernichtung eines Objekts neu vergeben werden.



Eine Reimplementierung von `GetHashCode()` sollte die Bedingung erfüllen, dass der Hash-Wert über zwei verschiedene Objekte, die die gleichen Werte enthalten, den gleichen Hash-Wert liefert. Der Hash-Wert soll aber keine eindeutige Objekt-ID sein, denn zwei (ähnliche) Objekte dürften den gleichen Hash-Wert liefern.

4.1.4 Equals()

Objektvergleiche Die Methode `Equals()` prüft, ob zwei Objekte gleich sind. Die Standardimplementierung prüft auf Referenzidentität, d.h. ob zwei Variablen auf den gleichen Speicherbereich zeigen. Rückgabewert ist `True` oder `False`.

```
Dim b As Boolean = o1.Equals(o2)
```

Wertidentität Abgeleitete Klassen können stattdessen auch Wertidentität implementieren, d.h. prüfen, ob die in zwei verschiedenen Instanzen enthaltenen Zustände gleich sind. So geht beispielsweise die Klasse `System.String` vor, so dass eine zwei verschiedenen Variablen zugewiesene Zeichenkette beim Vergleich mit `Equals()` das Ergebnis `True` liefert.

Bedingungen Eine Implementierung von `Equals()` muss folgende Bedingungen erfüllen:

- ▶ Reflexivität
Folgende Bedingung muss erfüllt sein:
`o1.Equals(o1) = True`
- ▶ Symmetrie
Folgende Bedingung muss erfüllt sein:
`o1.Equals(o2) = True`

daraus folgt

```
o2.Equals(o1) = True
```

► Transitivität

Folgende Bedingung muss erfüllt sein:

```
o1.Equals(o2) = True
```

und

```
o2.Equals(o3) = True
```

daraus folgt

```
o1.Equals(o3) = True
```

► Konsistenz

Bei unveränderten Inhalten der zu vergleichenden Objekte o1 und o2 ist das Ergebnis von o1.Equals(o2) immer gleich.

4.1.5 ReferenceEquals()

Die statische Methode `ReferenceEquals()` prüft auf Referenzidentität. Sie darf nicht in einer Unterklasse reimplementiert werden. Diese Methode sollte eingesetzt werden um zu prüfen, ob zwei Objektzeiger auf dasselbe Objekt verweisen. Da die Methode eine statische Methode ist, berücksichtigt sie nicht den Zustand des Objekts, auf dem die Methode aufgerufen wird.

Referenzidentität

```
Dim b As Boolean = Object.ReferenceEquals(o1,o2)
```

`ReferenceEquals()` entspricht dem Verhalten der Standardimplementierung von `Equals()`.

4.1.6 Beispiel zur Anwendung der Klasse System.Object

Im folgenden Listing werden die fünf vorgestellten Methoden an Objekten des Typs `Object`, `String` und `Integer` gezeigt.

Beispiele zur Klasse »Object«

```
' =====
' Beispiel aus FCL-Buch.sln
' Experimente mit dem Basistyp 'Object'
' (C) Holger@Schwichtenberg.de
' =====
Sub Object_test()
    Dim o1 As New Object()
    Dim o2 As New Object()
    Dim o3 As Object
    Dim o4 As Integer = 100 ' Integer-Objekt
    Dim o5 As String = "Holger Schwichtenberg" ' String-Objekt
```

Objekt Dim o6 As String = "Holger Schwichtenberg" ' String-

```

o3 = o1
say("---01:")
info(o1)
say("---02:")
info(o2)
say("---03:")
info(o3)
say("---04:")
info(o4)
say("---05:")
info(o5)
say("---06:")
info(o5)
say("--- Neues 01:")

' --- Vergleichen
say("---Vergleiche")
say(o1.Equals(o2)) ' False
say(o1.Equals(o3)) ' True
say(o5.Equals(o6)) ' True
say(Object.ReferenceEquals(o1, o3)) ' True
End Sub

' Hilfsroutine zur Ausgabe der Basisinformationen
Sub info(ByVal o As Object)
    say(o.ToString & ":" & _
        o.GetType.ToString & ":" & _
        o.GetHashCode.ToString)
End Sub

```

Listing 4.1: Experimente mit dem Basistyp 'Object' [object.vb]

Ausgabe

Ausgabe Das Programm liefert folgende Ausgabe:

```

1  --01:
2  System.Object:System.Object:50
3  --02:
4  System.Object:System.Object:55
5  --03:
6  System.Object:System.Object:50
7  --04:
8  100:System.Int32:100
9  --05:
10 Holger Schwichtenberg:System.String:-1399733058
11 --06:
12 Holger Schwichtenberg:System.String:-1399733058
13 -- Neues 01:

```

```

14 System.Object: System.Object: 31
15 --Vergleiche
16 False
17 True
18 True
19 True

```

Reimplementierung von Methoden aus System.Object

Das folgende Beispiel demonstriert die Reimplementierung der Methoden `ToString()`, `GetHashCode()` und `Equals()`. Die Klasse `TestKlasse` besitzt keine explizite Vererbung, sondern erbt implizit von `System.Object`. Dabei werden drei Methoden reimplementiert:

Beispiel zur Reimplementierung

- ▶ `ToString()` liefert nicht nur den Wert des Attributs `Name`, sondern auch noch einen zusätzlichen statischen Text.
- ▶ Als Hash-Code, den `GetHashCode()` zurückliefert, wird einfach die Länge des Wertes im Attribut `Name` verwendet.
- ▶ Der Vergleich zweier Objekte dieses Typs mit `Equals()` wird abgebildet auf den Vergleich des Inhalts des Attributs `Name`. Der Vergleich ist case-insensitive, berücksichtigt also nicht die Groß-/Kleinschreibung (Schrägstrich in andere Zeile). Wenn ein Objekt eines anderen Typs übergeben wird, schlägt der Vergleich immer fehl.
- ▶ Im Client für diese Klasse werden zwei Objekte instanziiert, die sich in der Groß-/Kleinschreibung des Namens unterscheiden. Der Vergleich liefert aufgrund der Reimplementierung von `Equals()` das Ergebnis `True`.

ToString()

GetHashCode()

Equals()

Module System_Object_2

```

Sub TestKlasse_test()
    Dim t1 As New TestKlasse()
    t1.name = "Holger Schwichtenberg"
    out("Typ: " & t1.GetType.ToString)
    out("ToString: " & t1.ToString())
    out("HashCode: " & t1.GetHashCode())

    Dim t2 As New TestKlasse()
    t2.name = "holger schwichtenberg"
    out("Gleich?: " & t1.Equals(t2))
End Sub

```

```

' =====
' Beispiel aus FCL-Buch.sln

```

```

' Überschreiben von ToString()
' (C) Holger@Schwichtenberg.de
' =====
Public Class TestKlasse

    Public name As String

    Public Overrides Function ToString() As String
        Return ("Dies ist das Testobjekt von " & name)
    End Function

    Public Overrides Function GetHashCode() As Integer
        Return (name.Length)
    End Function

    Public Overloads Overrides Function Equals( _
        ByVal o As Object) As Boolean

        Equals = False
        If o.GetType Is Me.GetType Then
            out("Vergleich von " & o.name & " mit " & _
                Me.name)
            If String.Compare(o.name, Me.name, False) Then
                Equals = True
            End If
        End If
    End Function
End Class

End Module

```

Listing 4.2: Reimplementierung von Methoden aus System.Object

Ausgabe

Ergebnis Das obige Beispiel liefert folgende Ausgabe:

```

1 Typ: FCL_Buch.System_Object_2+TestKlasse
2 ToString: Dies ist das Testobjekt von Holger
  Schwichtenberg
3 GetHashCode: 21
4 Vergleich von holger schwichtenberg mit Holger
  Schwichtenberg
5 Gleich?: True

```

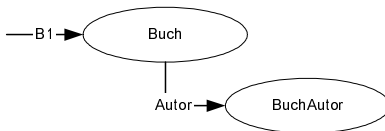
4.1.7 MemberwiseClone()

Die Methode `MemberwiseClone()` erstellt eine so genannte »flache Kopie« (engl. shallow copy) eines Objekts.

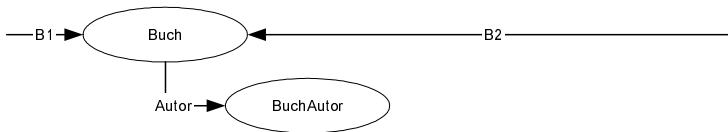
Der Unterschied zwischen einer flachen und einer tiefen Kopie kommt nur bei Referenztypen zum Tragen: Bei einer flachen Kopie wird zwar das Objekt kopiert, jedoch werden für die untergeordneten Objekte nur die Zeiger kopiert. Bei einer tiefen Kopie werden auch die Inhalte der referenzierten Objekte kopiert. Eine normale Zuweisung zwischen zwei Objekten legt keine flache Kopie an, sondern dupliziert nur den Verweis auf das Objekt. Das tiefe Kopieren muss man auf Basis der Schnittstelle `System.IClonable` selbst definieren. Eine in der FCL vorhandene – aber umständliche – Möglichkeit zur Erstellung tiefer Kopien ist die Serialisierung und anschließende Deserialisierung eines Objektbaums.

Flache Kopie vs. tiefe Kopie

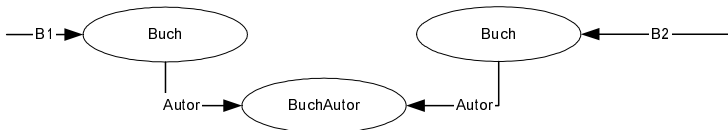
Ausgangszustand



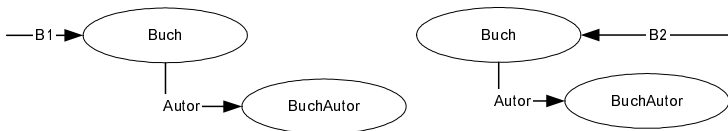
Referenzkopie



Seichte Kopie



Tiefe Kopie



(C) Holger@Schwichtenberg.de 2002

Abbildung 4.2: Verschiedene Formen des Kopierens

Diese Methode `MemberwiseClone()` ist `Protected` und kann daher nicht direkt aufgerufen werden. Eine abgeleitete Klasse kann diese Methode jedoch aufrufen.

Beispiel zu MemberwiseClone()

Beispiel Die folgende Routine basiert auf der in obiger Grafik dargestellten einfachen Objekthierarchie aus den Klassen Buch und BuchAutor. Buch implementiert dabei eine Methode Clone(), die Memberwise Clone() aufruft.

```
' =====
' Beispiel aus FCL-Buch.sln
' Klasse für Clone-Beispiel
' (C) Holger@Schwichtenberg.de
' =====
Class BuchAutor
    Public Name As String
End Class

' =====
' Beispiel aus FCL-Buch.sln
' Klasse für Clone-Beispiel
' (C) Holger@Schwichtenberg.de
' =====
Class Buch
    Public Titel As String
    Public Autor As BuchAutor
    Public Function Clone()
        Return (Me.MemberwiseClone())
    End Function
End Class
```

Die folgende Routine verwendet diese Klassen.

```
' =====
' Beispiel aus FCL-Buch.sln
' Beispiel zum Klonen von Objekten
' (C) Holger@Schwichtenberg.de
' =====
Sub test_clonen()

    Dim a1 As BuchAutor, a2 As BuchAutor
    Dim b1 As Buch, b2 As Buch
    Dim o As Object

    ' --- Instanzen bilden
    b1 = New Buch()
    b1.Titel = "Windows Scripting"
    a1 = New BuchAutor()
    a1.Name = "Holger Schwichtenberg"
    b1.Autor = a1

    ' =====
    ' Beispiel aus FCL-Buch.sln
    ' flache Kopie
```

```

' (C) Holger@Schwichtenberg.de
' =====
b2 = b1.Clone
b2.Titel = "COM-Komponenten-Handbuch"
b2.Autor.Name = "H. Schwichtenberg"

out(b1.Titel & " geschrieben von " & b1.Autor.Name)
out(b2.Titel & " geschrieben von " & b2.Autor.Name)
out("Gleiches Buch? " & b1.ReferenceEquals(b1, b2))
out("Gleicher Autor? " & b1.ReferenceEquals( _
    b1.Autor, b2.Autor))

' =====
' Beispiel aus FCL-Buch.sln
' Kopie der Referenz
' (C) Holger@Schwichtenberg.de
' =====
b2 = b1
b2.Titel = "COM-Komponenten-Handbuch"
b2.Autor.Name = "Schwichtenberg"

out(b1.Titel & " geschrieben von " & b1.Autor.Name)
out(b2.Titel & " geschrieben von " & b2.Autor.Name)
out("Gleiches Buch? " & b1.ReferenceEquals(b1, b2))
out("Gleicher Autor? " & b1.ReferenceEquals( _
    b1.Autor, b2.Autor))
End Sub

```

Listing 4.3: Beispiel zum Klonen von Objekten [object.vb]

Ausgabe

Zunächst werden ein Buch-Objekt und ein Buchautor-Objekt angelegt und das Buch-Objekt mit dem Buchautor-Objekt verknüpft. Die Ausführung einer flachen Kopie mit Clone() führt dazu, dass beide Buch-Objekte auf das gleiche Buchautor-Objekt verweisen. Beide Bücher können eigene Titel haben, jedoch gilt eine Veränderung an dem Buchautor-Objekt für beide Bücher. Die Ausgabe beweist, dass die Änderung des Autorennamens von »Holger Schwichtenberg« zu »H. Schwichtenberg« auf beide Bücher wirkt.

Ergebnis

```

1 Windows Scripting geschrieben von H. Schwichtenberg
2 COM-Komponenten-Handbuch geschrieben von H. Schwichtenberg
3 Gleiches Buch? False
4 Gleicher Autor? True

```

Im Vergleich dazu wird anschließend die Zuweisung der Objektzeiger ausgeführt. In diesem Fall gibt es nur ein Buch-Objekt. Eine Veränderung des Buchtitels führt dazu, dass über beide Zeiger das Gleiche ausgegeben wird.

- 5 COM-Komponenten-Handbuch geschrieben von Schwichtenberg
- 6 COM-Komponenten-Handbuch geschrieben von Schwichtenberg
- 7 Gleiches Buch? True
- 8 Gleicher Autor? True

4.2 Klassen für Basis-Datentypen

Eine Neuerung im .NET Framework gegenüber anderen Konzepten ist, dass alle Sprachen die gleichen elementaren Datentypen (Basisdatentypen wie String, Integer, Boolean) nutzen. Dieses Konzept heißt *Common Type System* (CTS) und ist Teil der *Common Language Specification* (CLS).

4.2.1 Übersicht

Die elementaren Datentypen der einzelnen .NET-Sprachen werden auf die Basisdatentypen abgebildet, z.B. entspricht ein `Int32` dem Typ `Integer` in Visual Basic .NET und dem `int` in C#.

Die Basisdatentypen sind in der Framework Class Library im Namespace `System` implementiert. Die meisten Basisdatentypen sind Wert-Typen, die als Structure implementiert und daher von der Klasse `System.ValueType` abgeleitet sind. Einige dieser Klassen sind aber auch Referenz-Typen, die direkt von `System.Object` erben

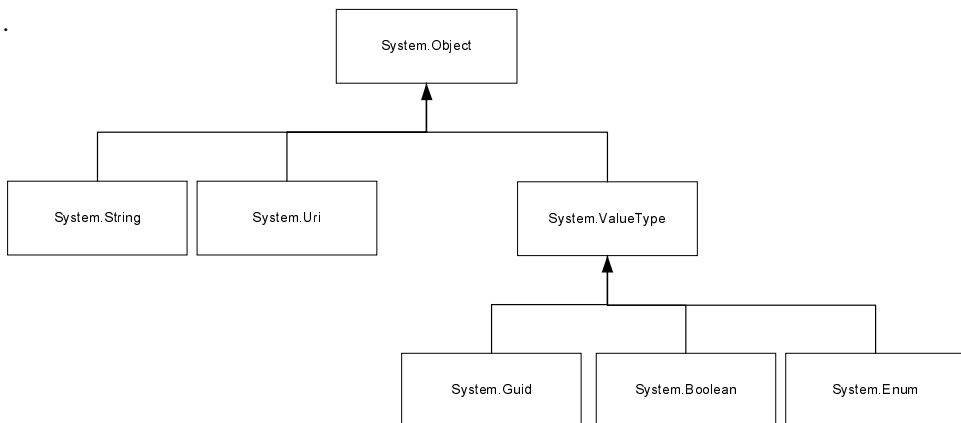


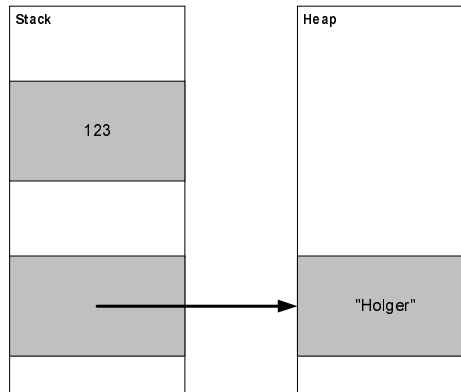
Abbildung 4.3: Vererbungshierarchie ausgewählter Basisdatentypen im Namespace `System`

Wert-Typ

Dim i as System.Int32 = 123

Referenz-Typ

Dim s as System.String = "Holger"



© Holger@Schwichtenberg.de 2001

Abbildung 4.4: Wert-Typ (z.B. Int32) vs. Referenz-Typ (z.B. String)

Die folgende Tabelle liefert eine Übersicht über die Basisdatentypen, die im Namespace `System` definiert sind. Dies sind die Datentypen des Common Type Systems (CTS) aus der Common Language Specification (CLS) sowie einige zusätzliche Datentypen, die *CLS-konform* sind, aber in der MSIL eine Entsprechung haben.

Common Type System (CTS)

Daten-typname	CLS-Typ	Implementierung	Größe (Bit)	Wertebereich	Vorzeichenbehaftung	Name in MSIL	Name in VB.NET	Name in C#
System.Byte	Ja	Struktur	8 Bit	0 bis 255	Nein	unsigned int8	Byte	byte
System.SByte	Nein	Struktur	8 Bit	-128 bis 127	Ja	int8	-	sbyte
System.Int16	Ja	Struktur	16 Bit	-32768 bis 32767	Ja	int16	Short	short
System.Int32	Ja	Struktur	32 Bit	-2147483648 bis 2147483647	Ja	int32	Integer	int
System.Int64	Ja	Struktur	64 Bit	-9223372036854775808 bis 9223372036854775807	Ja	int64	Long	long
System.UInt16	Nein	Struktur	16 Bit	0 bis 65535	Nein	unsigned int16	-	ushort
System.UInt32	Nein	Struktur	32 Bit	0 bis 4294967295	Nein	unsigned int32	-	uint
System.UInt64	Nein	Struktur	64 Bit	0 to 184467440737095551615	Nein	unsigned int64	-	ulong
System.Single	Ja	Struktur	32 Bit	-3.402823e38 bis 3.402823e38	Ja	float32	Single	float

Tabelle 4.1: NET-Basisdatentypen

Daten- typname	CLS- Typ	Imple- mentie- rung	Größe (Bit)	Wertebereich	Vorzei- chenbe- haftung	Name in MSIL	Name in VB.NET	Name in C#
System. Double	Ja	Struktur	64 Bit	1.79769313486232e308 bis 1.79769313486232e308	Ja	float64	Double	double
System. Object	Ja	Klasse	32 Bit (für den Zeiger)	beliebig	-	class System. Object	Object	object
System. Char	Ja	Struktur	16 Bit	Unicode-Zeichen		wchar	Char	char
System. String	Ja	Klasse		String von Unicode Zeichen (bis zu 2 Milliarden)		class System. String	String	string
System. Decimal	Ja	Struktur	96 Bit	- 7922816251426433759354 3950335 bis 7922816251426433759354 3950335	Ja	decimal	Decimal	decimal
System. Boolean	Ja	Struktur	32 Bit	True(1 oder -1)/False(0)		bool	Boolean	bool

Tabelle 4.1: NET-Basisdatentypen (Fortsetzung)

4.2.2 System.Void

Void Dieser Typ wird verwendet für Methoden, die keinen Rückgabewert liefern. System.Void ist definiert als eine Struktur, von der man keine Instanzen erzeugen kann.

4.2.3 Operationen der Basisdatentyp-Klassen

Datentyp-Operation Da die Basisdatentypen als Klassen bzw. Strukturen implementiert sind, besitzen sie auch Attribute und Methoden. Diese Attribute und Methoden können sowohl über Variablen des entsprechenden Typs aufgerufen werden als auch über Literale.

Die Abbildung 4.5 zeigt, dass Visual Studio .NET dafür Eingabeunterstützung anbietet.

Ermittlung der Ober- und Untergrenze bei Zahlen-Typen

MinValue und MaxValue Die Datentypen für Integer-Zahlen und Fließkomma-Zahlen (System.Int32, System.Int64, System.Double etc.)(Klammer nicht in andere Zeile) bieten jeweils zwei konstante Attribute: MinValue liefert den minimalen Wert, den der Typ aufnehmen kann. MaxValue liefert den maximalen Wert.

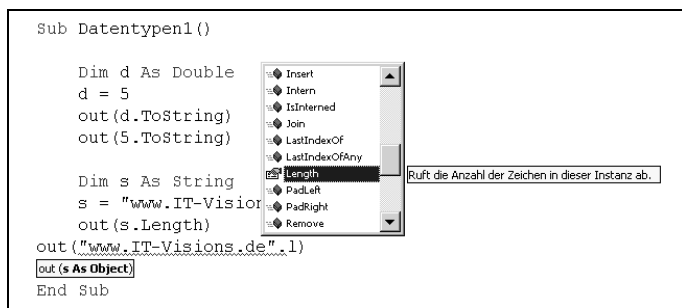


Abbildung 4.5: Eingabeunterstützung für die Mitglieder eines Basisdatentyps in Visual Studio .NET

```
' =====
' Beispiel aus FCL-Buch.sln
' Informationen über Wertebereiche von .NET-Datentypen
' (C) Holger@Schwichtenberg.de
' =====

Sub wertebereiche()

    Dim d As Double
    say("Double: von " & d.MinValue & " bis " & _
        d.MaxValue)

    Dim i32 As Int32
    say("Int32: von " & i32.MinValue & " bis " & _
        i32.MaxValue)

    Dim b As Byte
    say("Byte: von " & b.MinValue & " bis " & b.MaxValue)

    Dim bool As Boolean
    say("Boolean: von " & bool.MinValue & " bis " & _
        bool.MaxValue)

End Sub
```

Listing 4.4: Informationen über Wertebereiche von .NET-Datentypen [Datentypen.vb]

```
1 Double: von -1,79769313486232E+308 bis
1,79769313486232E+308
2 Int32: von -2147483648 bis 2147483647
3 Byte: von 0 bis 255
4 Boolean: von 0 bis 255
```

Beispiel

Ausgabe

Konvertierungen

In der Beta1 der FCL bot jede Basisdatentyp-Klasse noch zahlreiche Konvertierungsroutinen an. Diese sind nun in der Klasse `System.Convert` zentral verfügbar.

System.Convert

4.2.4 System.String

Der Basisdatentyp `String` (Klasse `System.String`) bietet zahlreiche Methoden zur Arbeit mit Zeichenketten an. Daher ist dieser Basisdatentyp ein eigenes Unterkapitel wert.

String-Vergleiche

- Compare()** Die Methode `Compare(s1, s2)` vergleicht zwei Strings. Im Standard werden Groß- und Kleinschreibung berücksichtigt. Dies kann mit einem dritten Parameter abgeschaltet werden: `String.Compare(s1, sla, True)`. `Compare()` liefert 0, 1 und -1 zurück und zeigt damit nicht nur an, ob zwei Strings verschieden sind, sondern auch ob der erste String lexikalisch kleiner (-1) oder größer ist als der zweite (1). Bei dem Vergleich wird die aktuelle Ländereinstellung (`Thread.CurrentThread.CurrentCulture`) berücksichtigt.
- Equals()** Die von `System.Object` geerbte Methode `Equals()` (Beispiel: `s1.Equals(s2)`) liefert lediglich ein `True` oder `False`. Der Vergleich ist immer case-sensitive und die Ländereinstellung ist irrelevant.

Beispiel

```
' =====
' Beispiel aus FCL-Buch.sln
' String-Vergleiche
' (C) Barbara Hildebrandt
' =====

Sub String_vergleich()
    Dim s1 As String
    Dim sla As String
    Dim s2 As String
    Dim i As Integer

    s1 = "Holger"
    sla = "holger"
    s2 = "Babsi"

    ' --- String-Vergleich ohne Berücksichtigung der
    ' --- aufrufenden Instanz
    ' --- (als aufrufende Instanz wird hier der
    ' --- Klassenname ("String") verwendet)

    '1. Die Strings sind gleich (Rückgabewert: 0)
    i = String.Compare(s1, s1)
    Console.WriteLine("1. '" & s1 & " _
        "' verglichen mit '" & s2 & "' ergibt: ")
    Console.WriteLine(i & " (Die Strings sind gleich)" _
        & vbCrLf)
```



```
'2. Die Strings sind nicht gleich, da defaultmäßig
'Groß- und Kleinschreibung berücksichtigt wird
' (Rückgabewert: 1, d.h.: s1 ist größer als sla)
i = String.Compare(s1, sla)
Console.WriteLine( _
    "2. s1 verglichen mit sla ergibt: ")
Console.WriteLine(i & vbNewLine)

'3. Die Strings sind gleich, da Groß- und
'Kleinschreibung nicht berücksichtigt werden
' (Rückgabewert: 0)
i = String.Compare(s1, sla, True)
Console.WriteLine( _
    "3. s1 verglichen mit sla ergibt: ")
Console.WriteLine(i & vbNewLine)

'4. Die Strings sind gleich, da Groß- und
'Kleinschreibung berücksichtigt werden
' (Rückgabewert: 1, d.h.: s1 ist größer als sla)
i = String.Compare(s1, sla, False)
Console.WriteLine( _
    "4. s1 verglichen mit sla ergibt: ")
Console.WriteLine(i & vbNewLine)

'5. String s1 ist größer als String s2
' (Rückgabewert: 1)
i = String.Compare(s1, s2)
Console.WriteLine( _
    "5. s1 verglichen mit s2 ergibt: ")
Console.WriteLine(i & vbNewLine)

'6. String s2 ist kleiner als String s1
' (Rückgabewert: -1)
i = String.Compare(s2, s1)
Console.WriteLine( _
    "6. s2 verglichen mit s1 ergibt: ")
Console.WriteLine(i & vbNewLine)

' --- String-Vergleich, der den Inhalt der
' ---aufrufenden Instanz berücksichtigt
' (Rückgabewert: 0, d.h.: die Strings sind gleich)
i = s1.CompareTo(s1)
Console.WriteLine("s1 verglichen mit s1 ergibt: ")
Console.WriteLine(i & vbNewLine)
```

End Sub

Listing 4.5: Beispiele zum String-Vergleich [String.vb]

Teilstring-Vergleiche

Vergleiche für Teile einer Zeichenkette können mit

```
b = s1.EndsWith(s2)

b = s1.StartsWith(s2) und

i1 = s1.IndexOf(s2)
```

`EndsWith()` und `StartsWith()` liefern `True` und `False`. `IndexOf()` liefert die Startposition von `s2` in `s1` oder `-1`, wenn `s2` nicht in `s1` enthalten ist.

Beispiel

```
' =====
' Beispiel aus FCL-Buch.sln
' Vergleiche für Teilstrings mit IndexOf()
' (C) Barbara Hildebrandt
' =====

Sub String_vergleich4()
    Dim s1 As String
    Dim s2 As String
    Dim i1 As Integer
    Dim i2 As Integer

    s1 = "Ein freies Wochenende ist eine feine Sache!"
    ' Beginne mit der Suche nach s2 bei Position 10
    i1 = 10
    s2 = "eine"
    i2 = s1.IndexOf(s2, i1)
    Console.WriteLine("Nun lautet der String: '{0}'", s1)
    Console.WriteLine( _
        "Das Wort '{0}' beginnt an Position {1}.", s2, i2)
    Console.WriteLine( _
        "Und taucht nochmal auf an Position {1}.", s2, _
        s1.IndexOf(s2, i2 + 1))
    s2 = ""
    ' Wenn der gesuchte String leer ist;
    ' wird der Wert der Startposition zurückgegeben...
    Console.WriteLine( _
        "Das Wort '{0}' beginnt an Position {1}.", s2, _
        s1.IndexOf(s2, 11))

End Sub
```

Listing 4.6: Anwendung von `IndexOf()` [String.vb]

Zeichenketten verbinden**Concat()**

Strings können in Visual Basic mit einem kaufmännischen Und (`&`) und in C# mit dem Pluszeichen (`+`) verknüpft werden. Der Vorteil der Methode `Concat()` in der Klasse `System.String` ist, dass eine beliebige Menge von Zeichenketten in einem Befehl verknüpft werden kann. `Concat()` akzeptiert eine beliebig lange Liste

von Parametern und sogar ein Array. Parameter kann dabei nicht nur ein `String/Array of String`, sondern auch jedes beliebige Objekt sein. Dies ist möglich, da jedes .NET-Objekt von `System.Object` die Methode `ToString()` erbt. Sofern diese Methode in der jeweiligen Klasse aber nicht reimplementiert ist, macht diese Verknüpfungsoperation wenig Sinn, denn die Standardimplementierung von `ToString()` liefert den Namen der Klasse (vgl. Kapitel 1.1.2).

Beispiel

```
' =====
' Beispiel aus FCL-Buch.sln
' Beispiel zur Anwendung von Concat
' (C) Barbara Hildebrandt
' =====

Sub String_verbindung()
    Dim s1 As String
    Dim s2 As String
    Dim s3 As String
    Dim sa() As String = _
        {"Eins", "Zwei", "Drei", "Vier", "Fünf"}

    s1 = "Barbara"
    s2 = "Hildebrandt"

    ' --- direktes Aneinanderhängen der beiden Strings
    s3 = String.Concat(s1, s2)
    out("s1 verknüpft mit s2 ergibt: ")
    out(s3 & vbCrLf)

    ' --- Verbinden der beiden Strings mit Leerzeichen
    s3 = String.Concat(s1, " ", s2)
    out("s1 verknüpft mit s2 ergibt: ")
    out(s3 & vbCrLf)

    ' --- Verbinden der Elemente des Arrays zu
    ' --- einem String
    s3 = String.Concat(sa)
    out("Verknüpfung der Elemente aus sa: ")
    out(s3 & vbCrLf)

End Sub
```

Listing 4.7: Anwendung von `Concat()` [`String.vb`]

Eine andere Möglichkeit der Zusammensetzung von Zeichenketten ist `Insert()`.

Insert()

```
s1 = s1.Insert(Position, s2)
```

Hier kann die Position angegeben werden, in der s2 in s1 eingefügt werden soll.

Beispiel

```
' =====
' Beispiel aus FCL-Buch.sln
' Zeichenketten zusammensetzen
' (C) Barbara Hildebrandt
' =====

Sub String_Insert()
    Dim s1 As String = "Heute ist es !"
    Dim s2 As String = "kalt"
    s1 = s1.Insert(13, s2)
    out("Der zusammengesetzte String lautet: " & s1)
End Sub
```

Listing 4.8: Anwendung von Insert() [String.vb]

Join() Join() verbindet die Elemente eines Arrays zu einer Zeichenkette unter Verwendung eines Trennzeichens. Das folgende Beispiel liefert die Ausgabe: »zusammengesetzter String: Zwei,Drei,Vier«.

Beispiel

```
' =====
' Beispiel aus FCL-Buch.sln
' Verbinden von Array-Elementen
' (C) Barbara Hildebrandt
' =====

Sub String_Join()
    Dim s1 As String = "," ' Das Komma als Trennzeichen
    Dim sal() As String = _
        {"Eins", "Zwei", "Drei", "Vier", "Fünf"}
    ' Beginne mit dem zweiten Element des Arrays
    ' (Array ist null-basiert) und
    ' verbinde nur die nächsten 3 Elemente des Arrays
    s1 = String.Join(s1, sal, 1, 3)
    Console.WriteLine("zusammengesetzter String: " & s1)
End Sub
```

Listing 4.9: Anwendung von Join() [String.vb]



Eine wesentlich performantere Methode, um Zeichenketten miteinander zu verknüpfen, ist der Einsatz der StringBuilder-Klasse (siehe Kapitel 7).

Zeichenkette verändern

Remove(), Replace() Remove(StartIndex, Länge) entfernt Zeichen an definierten Positionen aus einer Zeichenkette. Replace(Suchezeichen, Ersatz) ersetzt alle Vorkommen einer bestimmten Zeichenfolge durch eine andere Zeichenfolge.

```

' =====
' Beispiel aus FCL-Buch.sln
' Zeichen aus Zeichenkette entfernen
' (C) Barbara Hildebrandt
' =====
Sub String_Remove()
    Dim s1 As String
    Dim i1 As Integer
    s1 = "123-----123"
    out("Ursprünglicher String: " & s1)
    s1 = s1.Remove(3, 5)
    out("Gekürzter String: " & s1)
End Sub

```

Beispiel**Listing 4.10: Anwendung von Remove() [String.vb]**

```

1  Ursprünglicher String: 123-----123
2  Gekürzter String: 123123

```

Ausgabe

```

' =====
' Beispiel aus FCL-Buch.sln
' Zeichen ersetzen
' (C) Barbara Hildebrandt
' =====
Sub String_Replace()
    Dim s1 As String
    Dim s2 As String
    Dim s3 As String
    s1 = "Heute ist es kalt!"
    s2 = "kalt"
    s3 = "heiß"
    out("Ursprünglicher String: " & s1)
    s1 = s1.Replace(s2, s3)
    out("Veränderter String: " & s1)
End Sub

```

Beispiel**Listing 4.11: Anwendung von Replace() [String.vb]**

```

1  Ursprünglicher String: Heute ist es kalt!
2  Veränderter String: Heute ist es heiß!

```

Ausgabe

Formatierung

Die Methode `Format()` ermöglicht die Zusammensetzung eines Strings aus beliebigen Werten. Die Werte können dabei formatiert werden. `Format()` hat immer mindestens zwei Parameter: Der erste Parameter ist die Formatierungsvorschrift, der zweite ein zu formatierender Wert. `Format()` verwendet geschweifte Klammern als Platzhalter in der Formatierungsvorschrift. In `{1:D}` steht die erste Zahl für die laufende Nummer des Parameters und `D` für das anzuwendende Format. Die Zählung der Parameter beginnt bei 0. `D` steht für langes Datumsformat.

Format()

Weitere Informationen

Die komplette Syntax für die Formatierungsvorschriften ist sehr umfangreich und findet leider keinen Platz in diesem Buch. Sie finden(Wdh.) die Syntax unter dem Stichwort »Formatierung von Typen« in der .NET Framework SDK-Dokumentation.

Länderspezifische Ausgaben

`Format()` berücksichtigt bei der Ausgabe die aktuelle Ländereinstellung, wie das folgende Beispiel nach Veränderung von `Thread.CurrentThread.CurrentCulture` beweist. Länderspezifische Formatierungen sind aber auch möglich, ohne direkt den Thread zu verändern: Vor der Formatierungsvorschrift kann `Format()` auch ein `CultureInfo`-Objekt übergeben werden, das dann nur für diesen einen Befehl wirkt.

Beispiel

```
' =====
' Beispiel aus FCL-Buch.sln
' Formatieren einer Zeichenkette
' (C) Barbara Hildebrandt
' =====

Sub String_format()
    Dim s1 As String
    Dim s2 As String
    Dim s3 As String
    Dim i1 As Integer
    Dim MyDateTime As DateTime

    ' --- aktuelles Datum ermitteln
    MyDateTime = MyDateTime.Now
    ' --- Wert
    i1 = 1974

    ' --- Formatierung bei aktueller Landeseinstellung:

    s1 = "Unformatierte Ausgaben: " & vbNewLine
    s1 = s1 & "    Datum: " & MyDateTime & vbNewLine
    s1 = s1 & "    Zahl: " & i1 & vbNewLine
    s1 = s1 & "Formatierte Ausgaben: " & vbNewLine
    s1 = s1 & String.Format("    Datum: {0:D}" & _
        vbNewLine & "    Zahl als Währung: {1:C}", _
        MyDateTime, i1)
    s1 = s1 & vbNewLine & String.Format( _
        "    Zahl formatiert: {0:#,#}", i1)
    out(s1 & vbNewLine)

    ' --- Formatierung bei anderer CultureInfo:

    ' CultureInfo für US-Englisch erzeugen
    Dim ci As CultureInfo
    ci = New CultureInfo("en-US")
```

```

s2 = "Unformatierte Ausgaben: " & vbCrLf
s2 = s2 & "    Datum: " & MyDateTime & vbCrLf
s2 = s2 & "    Zahl: " & i1 & vbCrLf

s2 = s2 & "Formatierte Ausgaben mit anderer CultureInfo: "
& vbCrLf
s2 = s2 & String.Format(ci, "    Datum: {0:D}" & _
    vbCrLf & "    Betrag: {1:C}", MyDateTime, i1)
s2 = s2 & vbCrLf & String.Format(ci, _
    "    Zahl formatiert: {0:#,##}", i1)
out(s2 & vbCrLf)

' Thread auf andere CultureInfo umstellen
Thread.CurrentThread.CurrentCulture = ci

s3 = "Unformatierte Ausgaben nach Umstellung der
CultureInfo: " & vbCrLf
s3 = s3 & "    Datum: " & MyDateTime & vbCrLf
s3 = s3 & "    Zahl: " & i1 & vbCrLf

s3 = s3 & "Formatierte Ausgaben: " & vbCrLf
s3 = s3 & String.Format("    Datum: {0:D}" & vbCrLf & "
Betrag: {1:C}", MyDateTime, i1)
s3 = s3 & vbCrLf & String.Format("    Zahl formatiert:
{0:#,##}", i1)
out(s3 & vbCrLf)

' CultureInfo auf DE zurückstellen:
ci = New CultureInfo("de-DE")
Thread.CurrentThread.CurrentCulture = ci

```

End Sub

Listing 4.12: Anwendung von Format() [String.vb]

```

Unformatierte Ausgaben:
Datum: 03.04.2002 01:59:34
Zahl: 1974
Formatierte Ausgaben:
Datum: Mittwoch, 3. April 2002
Zahl als Währung: 1.974,00 _
Zahl formatiert: 1.974

Unformatierte Ausgaben:
Datum: 03.04.2002 01:59:34
Zahl: 1974
Formatierte Ausgaben mit anderer CultureInfo:
Datum: Wednesday, April 03, 2002
Betrag: $1,974.00
Zahl formatiert: 1,974

```

Ausgabe

```
Unformatierte Ausgaben nach Umstellung der CultureInfo:
Datum: 4/3/2002 1:59:34 AM
Zahl: 1974
Formatierte Ausgaben:
Datum: Wednesday, April 03, 2002
Betrag: $1,974.00
Zahl formatiert: 1,974
```

Weitere Möglichkeiten

Weitere Methoden Die Klasse `System.String` bietet noch mehr Möglichkeiten, die hier aus Platzgründen nur kurz erwähnt werden können:

- ▶ `Trim()`, `TrimStart()` und `TrimEnd()` entfernen Leerzeichen vom Anfang und/oder Ende einer Zeichenkette. Es kann als Parameter auch ein `Array of Char` übergeben werden, wenn andere Zeichen entfernt werden sollen.
- ▶ `ToCharArray()` liefert aus einem `String` ein `Array of Char` zurück.
- ▶ `Split(Trennzeichen)` spaltet einen `String` an dem angegebenen Trennzeichen in ein `Array of String` auf.
- ▶ `Substring(Startindex, Länge)` liefert einen Teil der Zeichenkette.
- ▶ `PadLeft(Länge, Zeichen)` und `PadRight(Länge, Zeichen)` füllen einen `String` links bzw. rechts solange mit dem angegebenen Zeichen auf, bis die gesamte Zeichenkette die angegebene Länge erreicht hat.
- ▶ `ToLower()` bzw. `ToUpper()` liefern die Zeichenkette in Klein- bzw. Großschreibweise.

4.3 Weitere Datentypen

Der Namespace `System` definiert weitere zusammengesetzte Datentypen für besondere Anwendungsfälle.

Datentyp	Implementierung	Typart	Bemerkung
<code>System.TimeSpan</code>	Sktruktur	Wertetyp	Eine Zeitspanne
<code>System.Guid</code>	Sktruktur	Wertetyp	Ein Global Unique Identifier (GUID) in der Form {00000000-0000-0000-0000-000000000000}

Tabelle 4.2: Weitere .NET-Datentypen

Datentyp	Implementierung	Typart	Bemerkung
System.URI	Class	Referenztyp	Uniform Resource Identifier (URI) in der Form <code>http://protokoll/pfad</code>
System.Version	Class	Referenztyp	Repräsentiert eine viergliedrige Versionsnummer in der Form <code>a.b.c.d</code>

Tabelle 4.2: Weitere .NET-Datentypen (Fortsetzung)

4.3.1 System.Guid

Ein Global Unique Identifier (GUID) ist eine 16 Byte (128 Bit) lange Zahl, die einen Bereich von rund $3,4028236 \times 10^{38}$ Werten (2 hoch 128) abdeckt. Ein GUID soll eine über Raum und Zeit eindeutige Zahl sein. GUIDs werden mit Hilfe eines im *Distributed Computing Environment* (DCE) der *Open Software Foundation* (OSF) spezifizierten Algorithmus aus dem Netzwerknamen (vorzugsweise Adresse der Ethernet-Karte) des Computers und der genauen aktuellen Uhrzeit gebildet. Sofern der Computer über keinen Netzwerknamen verfügt, wird eine Zufallszahl gebildet, die höchstwahrscheinlich eindeutig ist. Der Nummernraum soll theoretisch ausreichen, um in der Zukunft auch über unseren eigenen Planeten hinaus Eindeutigkeit zu gewährleisten.

Die OSF ist ein Konsortium von Soft- und Hardware-Herstellern. DCE definiert eine Umgebung für verteilte Systeme mit Werkzeugen und Diensten für verteilte Anwendungen. Zentraler Dienst im DCE ist der *Remote Procedure Call* (RPC), der entfernte Aufruf von Unterrouinen.

Global Unique Identifier (GUID)



Einsatz von GUIDs in COM

GUIDs sind Identifikatoren für alle zentralen Bausteine im Component Object Model (COM). Die folgende Tabelle zeigt, welche Bausteine in COM mit GUIDs identifiziert werden.

GUIDs in COM

COM-Baustein	GUID-Name
COM-Klasse	Class Identifier (CLSID)
COM-Schnittstelle	Interface Identifier (IID) oder Interface Pointer Identifier (IPID)

Tabelle 4.3: Verschiedene Typen von GUIDs in COM

COM-Baustein	GUID-Name
COM-Typbibliothek	Library Identifier (LibID)
COM-Komponentenkategorie	Category Identifier (CatID)
COM-Anwendung	Application Identifier (AppID)

Tabelle 4.3: Verschiedene Typen von GUIDs in COM (Fortsetzung)

GUIDs in .NET

GUIDs in .NET In .NET werden Klassen, Schnittstellen etc. nicht mehr durch GUIDs, sondern durch hierarchische Zeichenketten (in Verbindung mit Mechanismen zur eindeutigen Identifizierung einer Assembly) identifiziert. Nicht nur für die Interoperabilität, sondern auch bei der Entwicklung eigener Anwendungen kann es sinnvoll sein, einen GUID einzusetzen, da es leicht möglich ist, bei textlichen Namen eine Überschneidung zu erzielen.

Darstellung

GUIDs werden in der Regel als eine 16-stellige hexadezimale Zahl, eingerahmt in geschweifte Klammern, dargestellt.

Beispiel	GUID
VBScript Scripting Engine	{B54F3741-5B07-11cf-A4B0-00AA004A55E8}
Microsoft Word 2000	{000209FF-0000-0000-C000-000000000046}
Klasse für LDAP-Namespace	{228D9A82-C302-11cf-9AA4-00AA004A5691}
IUnknown-Schnittstelle	{00000000-0000-0000-C000-000000000046}
IDispatch-Schnittstelle	{00020400-0000-0000-C000-000000000046}

Tabelle 4.4: Beispiele für GUIDs in COM

Die Klasse System.Guid

NewGuid() Die Klasse `System.Guid` bietet neben `ToString()` nur eine wichtige Methode an: `NewGuid()` erzeugt einen neuen GUID. Um einen neuen GUID zu erzeugen, muss diese Methode aufgerufen werden. Bei der Instanziierung der Klasse wird kein GUID erzeugt; die neue Instanz wird mit dem Wert 0 initialisiert. `NewGuid()` ist eine statische Methode.

Beispiel

Im folgenden Beispiel wird zunächst eine Instanz von `System.Guid` erzeugt. Die Ausgabe beweist, dass dieser GUID nur aus Nullen besteht. Mit `NewGuid()` wird danach ein neuer GUID erzeugt.

Erzeugung eines GUIDs

```
' =====
' Beispiel aus FCL-Buch.sln
' GUID erzeugen und ausgeben
' (C) Holger@Schwichtenberg.de
' =====
Sub GUID()
    Dim g As New System.Guid()
    say("'Null'-GUID: " & g.ToString)
    g = g.NewGuid()
    say("Neuer GUID: " & g.ToString)
End Sub
```

Listing 4.13: GUID erzeugen und ausgeben [Datentypen.vb]

Ausgabe

```
1  'Null'-GUID:00000000-0000-0000-0000-000000000000
2  Neuer GUID: 0e854f4b-83a0-4707-b4b5-9eb867ca269f
```

4.3.2 System.Uri und System.UriBuilder

Ein Uniform Resource Identifier (URI) ist eine Zeichenkette, die eine Ressource eindeutig identifiziert. Dagegen ist ein Uniform Resource Locator (URL) eine URI, die eine konkret existierende Resource (in einem Netzwerk) beschreibt.

URI

Die allgemeine Form eines URIs ist:

```
schema://benutzer:kennwort@host/pfad/#fragment?query
```

Beispiele für URIs sind:

URI-Beispiele

- ▶ `http://www.it-Visions.de/global/download/download.asp ?Datei=/adsi/groupmaker.hta`
- ▶ `LDAP://sonne2000/CN=H.Schwichtenberg,OU=Basta,DC=IT-Visions,DC=de`
- ▶ `ftp://hs:egal@www.it-visionen.de/www/dotnet`

Arbeit mit URIs in der FCL

Der Namespace `System` bietet zwei Klassen zur Arbeit mit URIs:

FCL-Klassen

- ▶ `System.Uri` speichert einen URI.
- ▶ `System.UriBuilder` dient der Erzeugung eines URIs.

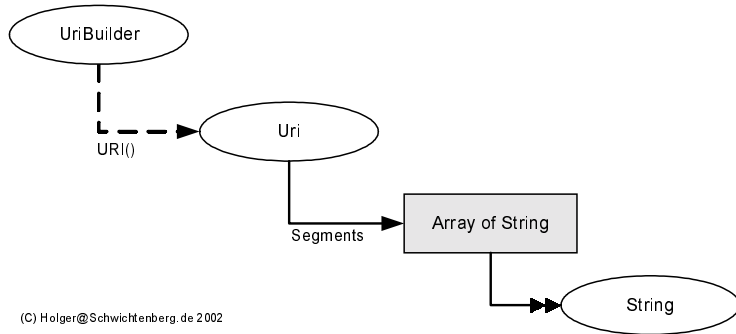


Abbildung 4.6: Objekthierarchie der Klassen »Uri« und »UriBuilder«

URIs speichern und aufteilen

System.Uri

Die Klasse `System.Uri` speichert einen URI und bietet verschiedene Attribute und Methoden, um einen URI in seine Bestandteile zu zerlegen. Bei der Instanziierung kann ein URI als String übergeben werden.



Die `URI`-Klasse speichert nur absolute URIs. Großbuchstaben im Schema-Namen oder Rechnernamen werden automatisch in Kleinbuchstaben verwandelt.

Die Methode `Uri1.MakeRelative(Uri2)` vergleicht zwei URIs und liefert den relativen Pfad, mit dem der `Uri2` von `Uri1` aus erreichbar ist. `Uri1` und `Uri2` sind jeweils vom Typ `System.Uri`.

Beispiel

Zerlegen eines GUIDs

In der folgenden Routine wird ein URI in seine Teile zerlegt und anschließend mit einem zweiten URI verglichen.

```

' =====
' Beispiel aus FCL-Buch.sln
' Arbeit mit URIs
' (C) Holger@Schwichtenberg.de
' =====
Sub Uri_zerteilen()
    Dim u As Uri

    Const URL1 = "HTTP://www.IT-Visions.de/global/
download/download.asp?Datei=/adsi/groupmaker.hta"

    ' --- URI anlegen
    u = New Uri(URL1)

```

```

' --- Bestandteile ausgeben
out("--- Bestandteile des URI:" & u.AbsoluteUri)
out(u.Host)
out(u.Authority)
out(u.AbsolutePath)
out(u.HostNameType)
out(u.IsDefaultPort)
out(u.IsFile)
out(u.IsLoopback)
out(u.IsUnc)
out(u.LocalPath)
out(u.PathAndQuery)
out(u.Port)
out(u.Query)
out(u.Scheme)
out(u.UserEscaped)
out(u.UserInfo)

out("--- Teile der URI:")
Dim s As String
For Each s In u.Segments
    out(s)
Next

' --- relative Sprungadresse erzeugen
Const URL2 = "http://www.IT-Visions.de/global
/databases/test.mdb"

out("--- Relativer Pfad von " & _
URL1 & " nach " & URL2 & ":" & _
u.MakeRelative(New Uri(URL2)))

End Sub

```

Listing 4.14: Arbeit mit URIs [Datentypen.vb]

Ausgabe

```

1  --- Bestandteile von URI:http://www.it-visions.de/
global/download/download.asp?Datei=adsi/groupmaker.hta
2  www.it-visions.de
3  www.it-visions.de
4  /global/download/download.asp
5  2
6  True
7  False
8  False
9  False
10 /global/download/download.asp
11 /global/download/download.asp?Datei=
adsi/groupmaker.hta
12 80

```

```

13 ?Datei=/adsi/groupmaker.hta
14 http
15 False
16
17 --- Teile der URI:
18 /
19 global/
20 download/
21 download.asp
22 --- Relativer Pfad von HTTP://www.IT-Visions.de/global/
download/download.asp? Datei=/adsi/groupmaker.hta nach http://
www.IT-Visions.de/global/databases/test.mdb:../databases/
test.mdb

```

URIs erzeugen mit dem UriBuilder

Erzeugen eines URIs

Die Klasse `System.UriBuilder` dient der Erzeugung eines URIs aus einzelnen Bestandteilen. Die Bestandteile können dem Konstruktor bei der Instanziierung übergeben oder über Attribute gesetzt werden. Die Methode `Uri()` liefert ein Objekt vom Typ `System.Uri` mit einem URI, der den festgelegten Einzelteilen entspricht.

Beispiel

In der folgenden Routine wird ein FTP-URI erzeugt.

```

' =====
' Beispiel aus FCL-Buch.sln
' Erzeugen eines URIs mit dem UriBuilder
' (C) Holger@Schwichtenberg.de
' =====
Sub URI_aufbauen()
    Dim ub As UriBuilder
    Dim u As Uri

    ' --- URI erzeugen
    ub = New UriBuilder()
    ub.Scheme = "ftp"
    ub.Host = "www.IT-Visions.de"
    ub.Port = "2222"
    ub.Path = "download"

    ' --- URI-Objekt erzeugen
    u = ub.Uri()

    ' --- URI ausgeben
    out("Erzeugte URI: " & u.AbsoluteUri)

End Sub

```

Listing 4.15: Erzeugen eines URIs mit dem UriBuilder [Datentypen.vb]

Ausgabe

Diese Routine liefert folgenden URI zurück:

ftp://www.it-visions.de:2222/download

4.3.3 System.Version

Die Klasse `System.Version` repräsentiert eine viergliedrige Versionsnummer (z. B. für eine `.dll`-Datei, eine `.exe`-Datei oder eine Assembly). Die Klasse bietet die Möglichkeit, vier einzelne Zahlen (a, b, c und d) in einen Versionsnummernstring der Form »a.b.c.d« umzuwandeln bzw. einen derartigen String wieder in seine vier Bestandteile aufzuteilen. Diese Klasse erspart in diesem speziellen Anwendungsfall aufwendigere Zeichenkettenoperationen.

Datei-Versionsnummern

Element	Name des Elements	Beispiel
1	Hauptversion (Major Version)	1
2	Unterversion (Minor Version)	8
3	Überarbeitung (Revision)	19
4	Erstellungsversion (Build Number). Dies wird auch als die Quick Fix Engineering (QFE)-Nummer bezeichnet.	72

Tabelle 4.5: Bedeutung der Glieder einer Versionsnummer

Beispiel

In folgender Routine wird zunächst eine Versionsnummernzeichenkette (»1.2.3.4«) aufgespalten. Danach wird eine Versionsnummer aus den Bestandteilen 1, 8, 19 und 72 neu erzeugt.

Versionsnummern erzeugen und aufspalten

```
' =====
' Beispiel aus FCL-Buch.sln
' Arbeit mit viergliedrigen Versionsnummern (a.b.c.d)
' (C) Holger@Schwichtenberg.de
' =====
Sub Versionsnummer()
    Dim v As New System.Version()

    ' --- Versionsnummer auseinander nehmen
    v = New System.Version("1.2.3.4")
    say("Hauptversionsnummer (Major):" & v.Major)
    say("Unterversionsnummer (Minor):" & v.Minor)
    say("Überarbeitung (Revision):" & v.Revision)
    say("Erstellungsversion (Build):" & v.Build)

    ' --- Versionsnummer erzeugen
    v = New System.Version(1, 8, 19, 72)
```

```
say("Version: " & v.ToString)
```

```
End Sub
```

Listing 4.16: Arbeit mit viergliedrigen Versionsnummern (a.b.c.d) [Datentypen.vb]

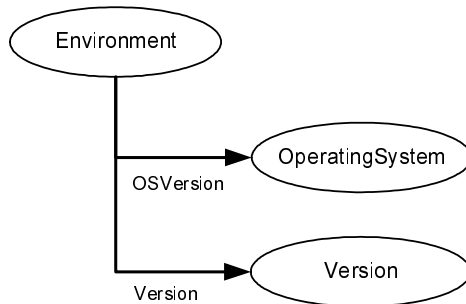
Ausgabe

```
1 Hauptversionsnummer (Major):1
2 Unterversionsnummer (Minor):2
3 Überarbeitung (Revision):4
4 Erstellungsnummer (Build):3
5 Version: 1.8.19.72
```

4.4 Klasse System.Console

Standardein- und -ausgabe

Die Klasse `Console` stellt den Zugriff auf die Standardein- und -ausgabe (also in der Regel das DOS-Fenster) bereit. Dabei gibt es drei Unterobjekte `In`, `Out` und `Error` vom Typ `System.IO.TextReader` bzw. `System.IO.TextWriter` (siehe Objektdiagramm).



(C) Holger@Schwichtenberg.de 2002

Abbildung 4.7: Objektmodell von `System.Console`

Die Klasse `Console` muss nicht instanziiert werden: Alle Attribute und Methoden sind statisch. Damit wirklich eine Interaktion mit der Standardein- und -ausgabe möglich ist, muss das Projekt als »Konsolenanwendung« definiert sein. Bei einer »Windows-Anwendung« bleiben alle Zugriffe auf die Standardein- und -ausgabe ohne Effekt. Beim Start einer Windows-Anwendung innerhalb der Visual Studio .NET-Entwicklungsumgebung wird jedoch die Ausgabe auf das Ausgabefenster geleitet.

Zentrale Funktionen können sowohl über die Klasse `Console` als auch über die drei Unterobjekte erreicht werden. So ist `Console.WriteLine()` ein Synonym für `Console.Out.WriteLine(s)` und `Console.ReadLine()` ein Synonym für `Console.In.ReadLine()`.

Instanziierung

Die Klasse `Console` enthält nur statische Mitglieder und ist daher wie ein Intrinsic Object zu verwenden, das nicht explizit instanziiert werden kann. Eine Instanz ist unter dem gleichen Namen wie die Klasse immer verfügbar. Um auf diese Instanz unter anderem Namen zugreifen zu können, reicht die einfache Deklaration eines Objekts vom Typ `System.Console`.

```
Dim c As System.Console
```

Intrinsic Object

Write() und WriteLine()

Beide Methoden senden den übergebenen Wert an die Standardausgabe. `WriteLine()` sendet zusätzlich am Ende das plattformspezifische *NewLine*-Zeichen (vgl. Klasse `System.Environment`). Beide Methoden sind mehrfach überladen, so dass neben Strings auch andere elementare Datentypen wie `Boolean`, `Int32`, `Char`, `Double` etc. akzeptiert werden.

Ausgaben

Eine vorherige explizite Konvertierung mit einer sprachspezifischen Funktion wie `CStr()` oder der Standardmethode `ToString()` ist nicht notwendig. Bei allen Klassen wird die Methode `ToString()` automatisch aufgerufen. Sofern diese nicht reimplementiert wurde, wird also der Name der Klasse ausgegeben. Eine Reimplementierung kann jeden beliebigen anderen String liefern.



`Write()` und `WriteLine()` unterstützen Platzhalter, die durch weitere Argumente mit Werten belegt werden können. Die Platzhalter sind Zahlen in geschweiften Klammern, wobei `{0}` für das erste zusätzliche Argument, `{1}` für das zweite usw. steht.

Read() und ReadLine()

`Read()` liest einzelne Zeichen, `ReadLine()` eine komplette Zeile. `Read()` bekommt aber das einzelne Zeichen nicht sofort nach Eingabe, sondern erst nach Abschluss einer Eingabe mit der Enter-Taste.

Eingaben

Beispiel

Das folgende Beispiel gibt zunächst drei Zeilen mit `Write()` und `WriteLine()` aus. Danach werden zwei Eingaben vom Anwender eingelesen und mit `WriteLine()` wieder eingegeben. Anschließend wartet die Routine auf einen Druck auf die Taste `[e]`.

```
' =====
' Beispiel aus FCL-Buch.sln
' Ein- und Ausgaben an der Kommandozeile
' (C) Holger@Schwichtenberg.de
' =====
Module system_console_1

Sub console_test()
    Dim c As System.Console ' Intrinsic Object
    ' --- Einfache Ausgaben
    c.WriteLine("Zeile1")
    c.Out.WriteLine("Zeile2")
    c.Out.Write("Zeile...")
    c.Write("3!")
    c.WriteLine()
    Dim o As Object
    c.WriteLine(o)

    ' --- zeilenweise Eingabe lesen
    c.WriteLine("Ihr Vorname?")
    Dim vorname As String
    vorname = c.ReadLine()
    Dim nachname As String
    c.WriteLine("Ihr Nachname?")
    nachname = c.ReadLine()

    ' --- Ausgabe mit Platzhaltern
    c.WriteLine("Hallo {0} {1}!", vorname, nachname)

    ' -- Warten bis ein e eingegeben wurde!
    Dim eingabe As Integer
    While True
        eingabe = c.Read
        ' Lesen, bis "e" eingegeben wird
        If eingabe = Asc("e") Then Exit While
        c.WriteLine(eingabe)
    End While

End Sub
End Module
```

Listing 4.17: Ein- und Ausgaben an der Kommandozeile [CD:/code/
Kapitel4_System.Console/Console.sln]

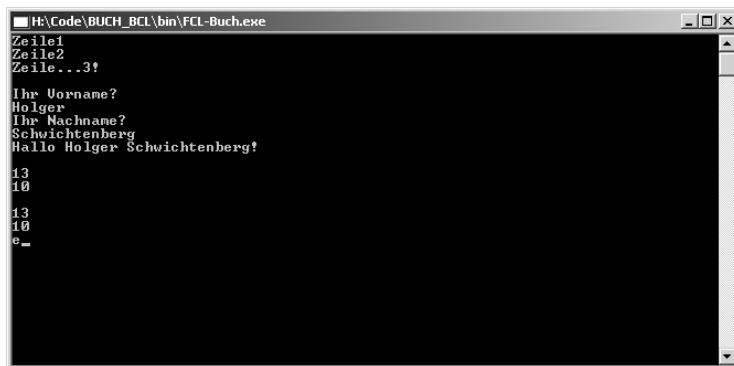


Abbildung 4.8: Ausgabe des Beispiels

4.5 Klasse System.BitConverter

Die Klasse `System.BitConverter` bietet eine Reihe von statischen Methoden zur Umwandlung zwischen .NET-Basisdatentypen und Byte-Folgen (Array of Byte).

`GetBytes()` wandelt einen Wert vom Typ `Boolean`, `Short`, `Long`, `Integer`, `Single`, `Double` oder `Char` in ein Array of Byte um.

GetBytes()

Zur Rückwandlung gibt es verschiedene Methoden der Form `ToDATENTYP()`, z.B. `ToChar()` oder `ToInt32()`.

Rückwandlung

Die Methode `ToString()` wandelt die Bytefolge nicht in einen String gemäß ASCII-Code, sondern in die Hexadezimal-Darstellung der Bytefolge um.

ToString()

Beispiel

```
' =====
' Beispiel aus FCL-Buch.sln
' Verwendung des BitConverters zur Umwandlung eines
' Integer-Wertes in eine Byte-Folge und umgekehrt
' (C) Holger@Schwichtenberg.de
' =====

Sub BITConverter1()
    Dim BC As System.BitConverter
    Dim i As Int32 = 1972
    Dim bytes() As Byte
    Dim b As Byte
    out("Ausgangswert: " & i)
    out("Bytefolge: ")
    ' --- Umwandlung
```

```

bytes = BC.GetBytes(i)
' --- Ausgabe der Byte-Folge
For Each b In bytes
    out(b)
Next
out("In Hex-Darstellung: " & BC.ToString(bytes, 0))
' --- Rückwandlung
i = BC.ToInt32(bytes, 0)
out("Wiedergewonnener Wert:" & i)
End Sub

```

Listing 4.18: Verwendung des BitConverters zur Umwandlung eines Integer-Wertes in eine Byte-Folge und umgekehrt [Datentypen.vb]

Ausgabe

```

1 Ausgangswert: 1972
2 Byte-Folge:
3 180
4 7
5 0
6 0
7 In Hex-Darstellung: B4-07-00-00
8 Wiedergewonnener Wert:1972

```

Umwandlung eines Strings

String- Umwandlung

Die direkte Umwandlung eines Strings in eine Byte-Folge ist nicht möglich, die Klasse `BitConverter` kann nur `Char`-Werte umwandeln. Daher muss ein String zunächst in ein Array of `Char` verwandelt werden und dieses dann elementweise in ein Array of `Bytes`.

```

' =====
' Beispiel aus FCL-Buch.sln
' Verwendung des BitConverters zur Umwandlung eines
' Strings In eine Byte-Folge
' (C) Holger@Schwichtenberg.de
' =====
Sub BITConverter2()
    Dim BC As System.BitConverter
    Dim s As String = "Essen-Byfang"
    Dim chars() As Char
    Dim c As Char
    Dim bytes(), bytes2() As Byte
    Dim b As Byte
    Dim x As Integer
    out("Ausgangswert: " & s)

    ' --- Umwandlung
    chars = s.ToCharArray
    ReDim Preserve bytes(s.Length * 2 - 1)

```

```

x = 0
For Each c In chars
    bytes2 = BC.GetBytes(c)
    bytes2.CopyTo(bytes, x)
    x += 2
Next
' --- Ausgabe
out("Bytefolge: ")

For Each b In bytes
    out(b)

Next

End Sub

```

Listing 4.19: Verwendung des `BitConverter` zur Umwandlung einer Zeichenkette in eine Byte-Folge [`Datentypen.vb`]

4.6 Klasse System.Convert

Die Klasse `System.Convert` bietet Methoden zur Datentypkonvertierung zwischen .NET-Basisdatentypen. Die Methoden haben Namen der Form `toDATENTYP()`, wobei `DATENTYP` ein .NET-Basistyp (CLR-Typ) ist. Jede Methode ist 17-fach überladen, entsprechend der Anzahl an Basisdatentypen.

**Datentyp-
konvertierung**

Alle Methoden sind statisch, daher ist eine Instanziierung nicht möglich und auch nicht notwendig.

**Statische
Methoden**

Natürlich sind einige Konvertierungen nicht sinnvoll. Es gibt dafür zwar eine überladene Methode, der Aufruf führt aber zu einer `InvalidCastException`. Nicht unterstützte Konvertierungen sind in der folgenden Tabelle genannt.

Quelltyp	Zieltyp
Char	Boolean Single Double Decimal DateTime
Boolean Single Double Decimal DateTime	Char

Tabelle 4.6: Nicht unterstützte Konvertierungen

Quellentyp	Zieltyp
DateTime	Alle außer String
Alle außer String	DateTime

Tabelle 4.6: Nicht unterstützte Konvertierungen (Fortsetzung)

Beispiel

Konvertierungs- Beispiele

Die folgende Routine demonstriert eine Reihe von Konvertierungen.

```
' =====
' Beispiel aus FCL-Buch.sln
' Konvertierungen mit System.Convert
' (C) Holger@Schwichtenberg.de
' =====
Sub konvertieren()

    Dim s As String
    Dim dt As Date
    Dim b As Boolean
    Dim i As Int32
    Dim d As Double

    ' Double -> String
    say("Double -> String")
    d = 12.34
    say(d)
    s = System.Convert.ToString(d)
    say(s)

    ' String -> Double
    say("String -> Double")
    d = System.Convert.ToDouble(s)
    say(d)

    ' Boolean -> String
    say("Boolean -> String")
    b = True
    s = System.Convert.ToString(b)
    say(s)

    ' String -> Boolean
    say("String -> Boolean")
    ' nur "True" oder "False"
    b = System.Convert.ToBoolean("False")
    say(b)

    ' Integer -> Boolean
    say("Integer -> Boolean (-1)")
```

```
i = -1
b = System.Convert.ToBoolean(i)
say(b) ' liefert True

say("Integer -> Boolean (0)")
i = 0
b = System.Convert.ToBoolean(i)
say(b) ' liefert True

say("Integer -> Boolean (2)")
i = 2
b = System.Convert.ToBoolean(i)
say(b) ' liefert True

' String -> Date
say("String -> Date")
s = "8/1/1972"
dt = System.Convert.ToDateTime(s)
say(dt) ' liefert 8.1.1972

s = "1.8.1972"
dt = System.Convert.ToDateTime(s)
say(dt) ' liefert 1.8.1972

' Date -> String
say("Date -> String")
dt = #8/1/1972#
say(dt) ' liefert 1.8.1972
s = System.Convert.ToString(dt)
say(dt) ' liefert 1.8.1972

dt = #1/8/1972#
say(dt) ' liefert 8.1.1972
s = System.Convert.ToString(dt)
say(dt) ' liefert 8.1.19
```

End Sub

Listing 4.20: Konvertierungen mit System.Convert [Datentypen.vb]

Ausgabe

```
1 Double -> String
2 12,34
3 12,34
4 String -> Double
5 12,34
6 Boolean -> String
7 True
8 String -> Boolean
9 False
```

```

10 Integer -> Boolean (-1)
11 True
12 Integer -> Boolean (0)
13 False
14 Integer -> Boolean (2)
15 True
16 String -> Date
17 08.01.1972
18 01.08.1972
19 Date -> String
20 01.08.1972
21 01.08.1972
22 08.01.1972
23 08.01.1972

```

4.7 Klasse System.Math

Diese Klasse stellt zwei mathematische Konstanten (e und pi) sowie eine Reihe von mathematischen Funktionen bereit: Abs(), Acos(), Asin(), Atan(), Atan2(), Ceiling(), Cos(), Cosh(), Exp(), Floor(), IEEERemainder(), Log(), Log10(), Max, Min(), Pow(), Round(), Sign(), Sin(), Sinh(), Sqrt(), Tan(), Tanh().

Alle Mitglieder dieser Klasse sind statisch.

Beispiel

```

Sub math1()
    Dim m As System.Math
    Dim x, y As Double
    x = 2.234
    y = m.Sin(x)
    out("Sin(" & x & ") = " & y)
    out("gerundet " & m.Round(y, 2))
    y = m.PI
    out("PI=" & y)
End Sub

```

Listing 4.21: Beispiel zur Anwendung der System.Math-Klasse [math.vb]

Ausgabe

```

1 Sin(2,234) = 0,78802395477922
2 gerundet 0,79
3 PI=3,14159265358979

```

4.8 Klasse System.Random

Pseudo-Zufalls- zahlen erzeugen

Diese Klasse dient der Erzeugung von Pseudo-Zufallszahlen. Die Zufallszahlen werden als »Pseudo-Zufallszahlen« bezeichnet, weil die Ermittlung nicht wirklich zufällig ist, sondern auf einem

mathematischen Verfahren beruht. Das Verfahren stellt aber eine statische Verteilung sicher, die einer zufälligen Verteilung fast entspricht. Daher kann diese Klasse für die meisten Anwendungsfälle eingesetzt werden.

Die Zufallszahlenerzeugung geht von einem zu übergebenden Grundwert aus. Bei gleichem Grundwert wird immer die gleiche Reihe von Zahlen erzeugt. Daher sind die Zahlen nicht wirklich zufällig.

Grundwert

Indem als Grundwert die aktuelle Zeit genommen wird, kann man die Zufälligkeit der erzeugten Zahlen verbessern.



Beispiel

Im folgenden Beispiel kommen die beiden wichtigsten Methoden von System.Random zum Einsatz: Next(x,y) erzeugt eine ganzzahlige Zufallszahl im Wertebereich $\geq x$ und $< y$. NextDouble() erzeugt eine Zufallszahl ≥ 0.0 und < 1.0 .

```
' =====
' Beispiel aus FCL-Buch.sln
' Erzeugung von Zufallszahlen
' (C) Holger@Schwichtenberg.de
' =====
Sub rnd1()
    Dim r As New System.Random()
    Dim a As Byte
    For a = 1 To 5
        out("Zufallszahl: " & r.Next(10, 20))
    Next
    For a = 1 To 5
        out("Zufallszahl: " & r.NextDouble())
    Next
End Sub
```

Listing 4.22: Erzeugung von Zufallszahlen [math.vb]

```
1 Zufallszahl: 13
2 Zufallszahl: 15
3 Zufallszahl: 13
4 Zufallszahl: 11
5 Zufallszahl: 11
6 Zufallszahl: 0,804063599931106
7 Zufallszahl: 0,37215324089497
8 Zufallszahl: 0,648711908445094
9 Zufallszahl: 0,00241243653111739
10 Zufallszahl: 0,107465830681597
```

4.9 Klasse System.GC

Garbage Collector steuern

Diese Klasse dient der manuellen Steuerung des .NET Garbage Collectors (GC). Der Garbage Collector arbeitet automatisch im Hintergrund, kann aber über die Klasse `System.GC` auch manuell bedient werden. GC bietet folgende Methoden:

- ▶ `Collect()` startet die Garbage Collection.
- ▶ `GetTotalMemory()` ermittelt die (ungefähre) Menge des für die Application Domain allokierten Speichers. `GetTotalMemory(False)` liefert den aktuellen Wert. `GetTotalMemory(True)` wartet einen Moment auf die Ausführung einer Garbage Collection.
- ▶ `WaitForPendingFinalizers()` hält das Programm an, bis alle Finalizer ausgeführt sind.
- ▶ `SuppressFinalize(OBJECT)` vermerkt für das angegebene Objekt, dass `Finalize()` nicht aufgerufen werden soll.
- ▶ `ReRegisterForFinalize(OBJECT)` macht die Vormerkung von `SuppressFinalize(OBJECT)` rückgängig.



Die Existenz von `SuppressFinalize(OBJECT)` belegt: Sie können niemals sicher sein, dass der Destruktor `Finalize()` aufgerufen wird. Die Kontrolle hat immer der Client.

Beispiel

Beispiel zur Steuerung des Garbage Collectors

Das folgende Beispiel demonstriert die Arbeitsweise des Garbage Collectors und die manuelle Steuerung des GCs über `System.GC`. Für diese Demonstration wird eine Klasse benötigt, die genau mitzählt, wie viele Instanzen es von ihr gibt. Um die Anzahl der Instanzen zählen zu können, werden *klassenweite Variablen (shared members)* benötigt. Die Klasse `Instanzenzaehler` besitzt zwei Shared Members:

- ▶ Das Attribut `Anzahl_Instanzen` zählt, wie viele Instanzen der Klasse es gibt. Dieser Wert wird im Konstruktor erhöht und im Destruktor erniedrigt.
- ▶ Das Attribut `Anzahl_Instanziierungen` zählt, wie oft der Konstruktor aufgerufen wurde. Dieser Wert wird im Konstruktor erhöht, aber im Destruktor nicht erniedrigt.

- Darüber hinaus besitzt jede Instanz eine eindeutige Nummer (InstanzNummer), die in einem Instanz-Member gespeichert wird. InstanzNummer wird im Konstruktor gleich dem aktuellen Wert von Anzahl_Instanziierungen gesetzt.

Klasse Instanzenzaehler

Das folgende Listing zeigt die Implementierung der Klasse.

Hilfsklasse

```
' =====
' Beispiel aus FCL-Buch.sln
' Klassen für GC-Beispiel
' (C) Holger@Schwichtenberg.de
' =====

Public Class Instanzenzaehler
    ' --- Gemeinsame Mitglieder aller Instanzen
    Private Shared Anzahl_Instanzen As Integer
    Private Shared Anzahl_Instanziierungen As Integer

    ' --- Objektmitglieder
    Dim InstanzNummer As Integer

    ' =====
    ' Beispiel aus FCL-Buch.sln
    ' Konstruktor für erste Instanz
    ' (C) Holger@Schwichtenberg.de
    ' =====

    Shared Sub New()
        out("Erste Instanz wurde erzeugt!")
    End Sub

    ' --- Objektweiser Konstruktor
    Sub New()
        Dim ausgabe As String

        ' --- Zähler erhöhen
        Anzahl_Instanzen += 1
        Anzahl_Instanziierungen += 1
        InstanzNummer = Anzahl_Instanziierungen
        ' --- Ausgaben
        out(" Konstruktor: Instanziierung #" & _
            Anzahl_Instanziierungen)
        out(" Konstruktor: Anzahl lebende Instanzen: " & _
            Anzahl_Instanzen)

    End Sub

    ' --- Destruktor
```

```

Protected Overrides Sub Finalize()
    ' --- Zähler erniedrigen
    Anzahl_Instanzen -= 1
    ' --- Ausgaben
    out(" Destruktor: für Objekt #" & InstanzNummer)
    out(" Destruktor: Anzahl restlicher Instanzen: " &
        & Anzahl_Instanzen)
End Sub
End Class

```

Listing 4.23: Klassen für GC-Beispiel [GarbageCollector.vb]

Client garbagecollector_test()

Client für Hilfsklasse

Die folgende Routine verwendet die oben definierte Klasse Instanzenzaehler. Zunächst werden sechs Instanzen der Klasse Instanzenzaehler erzeugt und in einem Array gespeichert. Dann werden nacheinander einzelne Objekte vernichtet. Bei Objekt #6 wird direkt nach der Objektvernichtung der GC aufgerufen. Dass der GC arbeitet, wird durch die Ausgaben des Destruktors bewiesen. Der freie Speicher wird dann direkt dreimal nacheinander ausgegeben, um zu demonstrieren, dass die Freigabe des Speichers im Hintergrund abläuft und kurze Zeit dauern kann.

Anschließend wird Objekt #3 vernichtet, ohne Aufruf des GCs. Das Ausbleiben der Ausgaben des Destruktors und die konstante Speicherallokation beweisen, dass der Destruktor noch gar nicht aufgerufen wurde.

Objekt #2 wird dann ebenfalls ohne Aufruf von GC.Collect() vernichtet. Jedoch bewirkt der Parameter true bei GetTotalMemory (True), dass der GC gestartet wird. In diesem Moment wird nicht nur der Destruktor von Objekt #2, sondern auch von dem zuvor vernichteten Objekt #3 aufgerufen.

Vor der Vernichtung von Objekt #1 wird dann SuppressFinalize() für dieses Objekt gestartet. Nach dem anschließenden Collect() stellt man zwar eine Speicherabnahme fest, jedoch bleibt der Aufruf des Destruktors für Objekt #1 aus.

```

' --- Manueller Aufruf des .NET Garbage Collectors
Sub garbagecollector_test()

    Dim i As Integer
    Dim o(10) As Object

    'Dim x As Object = Application
    outtitle("Manuelle Bedienung des GC")

```

```
out("### Objekte erzeugen...")
For i = 1 To 6
    ' --- Objekt erzeugen
    o(i) = New Instanzenzaehler()
    ' --- freien Speicher ausgeben
    out("Speicher: " & GC.GetTotalMemory(True))
Next

out("### Objekte schrittweise vernichten...")

' ---- 1. Objekt vernichten
out("Objekt 6 vernichten...")
o(6) = Nothing
' --- GC starten
GC.Collect()
' --- freien Speicher ausgeben
out("Speicher: " & GC.GetTotalMemory(False))
out("Speicher: " & GC.GetTotalMemory(False))
out("Speicher: " & GC.GetTotalMemory(False))

' ---- 2. Objekt vernichten
out("Objekt 3 vernichten...")
o(3) = Nothing
' --- freien Speicher ausgeben
out("Speicher: " & GC.GetTotalMemory(False))
out("Speicher: " & GC.GetTotalMemory(False))
out("Speicher: " & GC.GetTotalMemory(False))

' ---- 3. Objekt vernichten
out("Objekt 2 vernichten...")
o(2) = Nothing
' --- GC + freien Speicher ausgeben
out("Speicher: " & GC.GetTotalMemory(True))

' ---- 4. Objekt vernichten
out("Objekt 1 vernichten...")
' --- Finalize() unterdrücken
GC.SuppressFinalize(o(1))
o(1) = Nothing
' --- GC starten
GC.Collect()
out("Speicher: " & GC.GetTotalMemory(True))

' --- Endausgabe
out("### ENDE DES GC-TEST")
End Sub
```

Listing 4.24: Manueller Aufruf des .NET Garbage Collectors [GarbageCollector.vb]

Ausgabe

Speicherbedarf Durch obigen Client wird die nachfolgend wiedergegebene Ausgabe erzeugt. Man sieht bei der Instanziierung (Zeilen 6, 9, 12, 15, 18 und 21), dass jeweils zwölf Bytes für eine Instanz verbraucht werden. Nach dem Aufruf von `GC.Collect()` werden aber – nach kurzer Verzögerung – mehr als zwölf Bytes freigegeben (Zeile 27), weil der Garbage Collector auch mit internen Objekten aufräumt. Dass nach der Vernichtung der Objekte #2 und #3 in Zeile 38 der Speicherbedarf wieder ansteigt, ist ebenfalls auf interne Vorgänge zurückzuführen. Nach der Vernichtung von Objekt #1 sieht man aber deutlich den Rückgang von zwölf Bytes (Zeile 40 im Vergleich zu Zeile 38). Nach dem Ende der Unteroutine werden erst die Destruktoren für die Objekte #4 und #5 aufgerufen, die von dem Programm in Ruhe gelassen wurden.

Zählfehler? Dass das Programm am Ende (Zeile 45) des Destruktors von Objekt #4 meint, dass es nun immer noch eine Instanz gäbe, ist erklärbar: Durch die Unterdrückung des Aufrufs des Destruktors von Objekt #1 wurde das Klassenmitglied `Anzahl_Instanzen` der Klasse `Instanzenzaehler` bei der Vernichtung von Objekt #1 nicht erniedrigt.

```

1  Manuelle Bedienung des GC
2  ### Objekte erzeugen...
3  Erste Instanz wurde erzeugt!
4  Konstruktor: Instanziierung #1
5  Konstruktor: Anzahl lebende Instanzen: 1
6  Speicher: 49052
7  Konstruktor: Instanziierung #2
8  Konstruktor: Anzahl lebende Instanzen: 2
9  Speicher: 49064
10 Konstruktor: Instanziierung #3
11 Konstruktor: Anzahl lebende Instanzen: 3
12 Speicher: 49076
13 Konstruktor: Instanziierung #4
14 Konstruktor: Anzahl lebende Instanzen: 4
15 Speicher: 49088
16 Konstruktor: Instanziierung #5
17 Konstruktor: Anzahl lebende Instanzen: 5
18 Speicher: 49100
19 Konstruktor: Instanziierung #6
20 Konstruktor: Anzahl lebende Instanzen: 6
21 Speicher: 49112
22 ### Objekte schrittweise vernichten...
23 Objekt 6 vernichten...
24 Destruktor: für Objekt #6
25 Destruktor: Anzahl restlicher Instanzen: 5

```

```

26 Speicher: 49112
27 Speicher: 57304
28 Speicher: 57304
29 Objekt 3 vernichten...
30 Speicher: 57304
31 Speicher: 57304
32 Speicher: 57304
33 Objekt 2 vernichten...
34 Destruktor: für Objekt #3
35 Destruktor: Anzahl restlicher Instanzen: 4
36 Destruktor: für Objekt #2
37 Destruktor: Anzahl restlicher Instanzen: 3
38 Speicher: 49992
39 Objekt 1 vernichten...
40 Speicher: 49980
41 ### ENDE DES GC-TEST
42 Destruktor: für Objekt #5
43 Destruktor: Anzahl restlicher Instanzen: 2
44 Destruktor: für Objekt #4
45 Destruktor: Anzahl restlicher Instanzen: 1

```

4.10 Klasse System.Environment

Diese Klasse stellt verschiedene Informationen über das Betriebssystem und die aktuelle Anwendung bereit. Die folgende Tabelle listet alle wichtigen Attribute und Methoden der Klasse `System.Environment` auf.

**Umgebungs-
informationen**

Attribut	Erläuterung
UserName	Name des angemeldeten Benutzers
UserDomainName	Name der Domäne, zu der der angemeldete Benutzer gehört
MaschineName	Name des Computers
UserInteractive	Liefert wahr, wenn der Benutzer interaktiv an Windows angemeldet ist. Liefert falsch, wenn die Anwendung ein Dienst oder eine Web-Anwendung ist.
CommandLine	Name und Pfad der Anwendung
GetCommandLineArgs()	Liefert ein Array of String der übergebenen Kommandozeilenparameter
WorkingSet	Menge des allokierten Speichers für diese Anwendung

Tabelle 4.7: Mitglieder der Klasse `System.Environment`

Attribut	Erläuterung
StackTrace	Zeichenkette mit Informationen über die aktuelle Aufrufhierarchie StackTrace() liefert die Aufrufreihenfolge der Unter-routinen bis zum aktuellen Punkt in einem Programm in Form eines Strings. Diese Information ist identisch mit den Daten, die Sie bei einem Laufzeitfehler von der CLR erhalten. Das Programm wird aber durch den Aufruf von StackTrace() nicht abgebrochen.
SystemDirectory	Pfad zum Systemverzeichnis
CurrentDirectory	Pfad zum aktuellen Verzeichnis
GetFolderPath (SONDERORDNER_TYP)	Liefert den Pfad eines bestimmten Sonderordners
NewLine	Informationen über das Trennzeichen, das auf der jeweiligen Plattform dazu dienen, Zeilenumbrüche in Zeichenketten zu definieren
TickCount	Anzahl der vergangenen Millisekunden seit System-start
GetEnvironment Variables()	Liefert eine Collection mit DictionaryEntry-Objekten für alle Umgebungsvariablen
GetEnvironment Variable(VAR)	Liefert den Wert einer bestimmten Umgebungs-variablen
ExpandEnvironment Variables (STRING)	Ersetzt alle Umgebungsvariablen in dem angegebenen String
OSVersion	Das Attribut OSVersion bietet den Zugriff auf ein Unterobjekt vom Typ OperatingSystem. Diese Klasse implementiert zwei Attribute: Plattform: Typ des Betriebssystems (DOS-basiert oder NT-basiert) Version: Version des Betriebssystems
Version	Version der CLR
GetLogicalDrives()	GetLogicalDrives() liefert die Namen der logischen Laufwerke auf dem System in Form eines Arrays von String-Werten.
ExitCode	Attribut zum Setzen des Rückgabewertes der Anwendung
Exit()	Beenden der Anwendung

Tabelle 4.7: Mitglieder der Klasse System.Environment (Fortsetzung)

Objektmodell

- Die folgende Grafik zeigt die Objekthierarchie der von `System.Environment` ausgehenden Klassen.

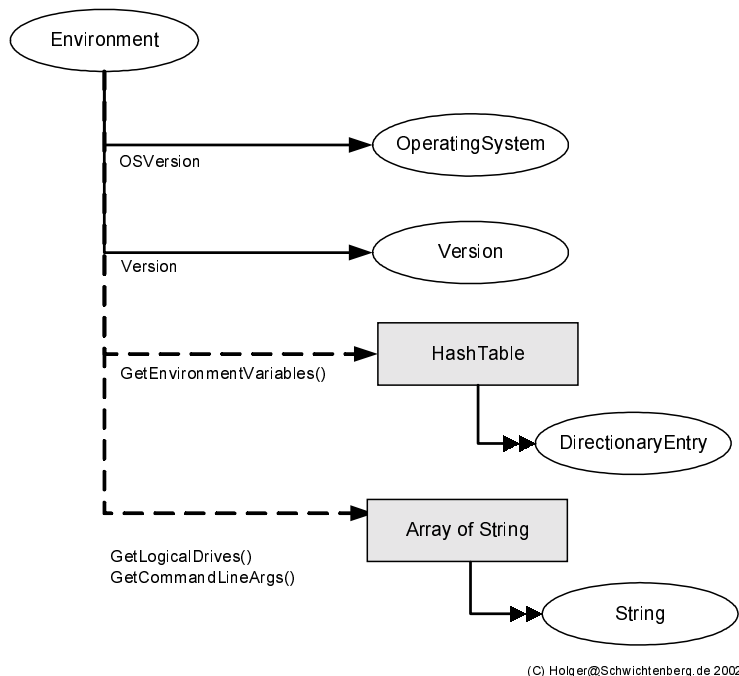


Abbildung 4.9: Objektmodell von »`System.Environment`«

Instanziierung

Alle Mitglieder der Klasse `Environment` sind statisch. Daher muss die Klasse nicht instanziiert werden. Sie verhält sich wie ein Intrinsic Object. Um auf diese Instanz unter anderem Namen zugreifen zu können, reicht die einfache Deklaration eines Objekts vom Typ `System.Environment`.

Intrinsic Object

Beispiel

Das folgende Listing zeigt die Ausgabe der wichtigsten Informationen zur Klasse `Environment`. Bitte beachten Sie, dass die Attribute `Version` und `OSVersion.Version` Objekte vom Typ `System.Version` liefern. Zur Ausgabe müssen Sie die `toString()`-Methode verwenden.

Ausgabe von Environment-Informationen

```

' =====
' Beispiel aus FCL-Buch.sln
' Ausgabe von Informationen über System und Prozess
' (C) Holger@Schwichtenberg.de
' =====
Sub env_test()
    Dim e As Environment
    Dim o As DictionaryEntry
    Dim s As String

    ' --- Verschiedenes
    say("ASCII-Wert des Zeilenumbruchs:" & Asc(e.NewLine))

    ' --- Zeitinfos
    say("ms seit Systemstart:" & e.TickCount)

    ' --- Benutzerinfos
    say("Computer: " & e.MachineName)
    say("Benutzer: " & e.UserName)
    say("Domäne: " & e.UserDomainName)
    say("Interaktiv? " & e.UserInteractive)

    ' --- Anwendungs-Infos
    say("Aufruf: " & e.CommandLine)
    say("Aufruf-Parameter:")
    For Each s In e.GetCommandLineArgs
        out("- " & s)
    Next
    say("Allokierter Speicher: " & e.WorkingSet)
    say("Anwendungsversion: " & e.Version.ToString)
    say("Stacktrace:")
    say(e.StackTrace)

    ' --- Verzeichnisinfos
    say("Systemverzeichnis: " & e.SystemDirectory)
    say("Aktuelles Verzeichnis:" & e.CurrentDirectory)

    say("Logische Laufwerke:")
    For Each s In e.GetLogicalDrives()
        say("- " & s)
    Next

    ' --- Plattform-Infos
    say("Plattform-Typ:" & e.OSVersion.Platform)
    say("CLR Version: " & e.OSVersion.Version.ToString)

    ' --- Umgebungsvariablen
    say("Spezielle Umgebungsvariablen")
    say(e.GetEnvironmentVariable("Path"))
    say(e.ExpandEnvironmentVariables( _
        "Betriebssystem = %OS%"))

```

```

say("Alle Umgebungsvariablen")
For Each o In e.GetEnvironmentVariables
    say(" - " & CStr(o.Key) & "=" & CStr(o.Value))
Next

End Sub

```

Listing 4.25: Ausgabe von Informationen über System und Prozess [environment.vb]

Ausgabe

Die Routine lieferte auf einem Testrechner folgende Ausgabe:

Ausgabe

```

1  ASCII-Wert des Zeilenumbruchs:13
2  ms seit Systemstart:1031547827
3  Computer: MARS
4  Benutzer: hs
5  Domäne: SONNENSYSTEM
6  Interaktiv? True
7  Aufruf: "H:\Code\BUCH_FCL\bin\FCL-Buch.exe"
8  Aufruf-Parameter:
9  - H:\Code\BUCH_FCL\bin\FCL-Buch.exe
10 Allokierter Speicher: 12050432
11 Anwendungsversion: 1.0.3328.4
12 Stacktrace:
13 at System.Environment.GetStackTrace(Exception e)
   at System.Environment.GetStackTrace(Exception e)
   at System.Environment.get_StackTrace()
   at FCL_Buch.environment_test.env_test() in
   H:\Code\BUCH_FCL\System\environment.vb:line 33
   at FCL_Buch.main.ablauf() in
   H:\Code\BUCH_FCL\main.vb:line 15
   at FCL_Buch.main.main() in
   H:\Code\BUCH_FCL\main.vb:line 162
14 Systemverzeichnis: C:\WINDOWS\System32
15 Aktuelles Verzeichnis:H:\Code\BUCH_FCL\bin
16 Logische Laufwerke:
17 - A:\
18 - C:\
19 - D:\
20 - E:\
21 - F:\
22 - H:\
23 - L:\
24 - M:\
25 - N:\
26 - R:\
27 - W:\
28 Plattform-Typ:2
29 CLR Version: 5.1.2600.0
30 Spezielle Umgebungsvariablen
31 C:\WINDOWS\system32;C:\WINDOWS;

```

```

C:\WINDOWS\System32\Wbem;C:\Programme\Microsoft SQL
Server\80\Tools\Binn\;C:\WINDOWS\Microsoft.NET\
Framework\v1.0.3328
32 Betriebssystem = Windows_NT
33 Alle Umgebungsvariablen
34 System.Collections.Hashtable
35 - SystemDrive=C:
36 - USERPROFILE=C:\Dokumente und Einstellungen\hs
37 - INCLUDE=E:\VSNET\FrameworkSDK\include\;
    E:\Dev\wmisd\include
38 - Path=C:\WINDOWS\system32;C:\WINDOWS;
    C:\WINDOWS\System32\Wbem;C:\Programme\Microsoft SQL
    Server\80\Tools\Binn\;
    C:\WINDOWS\Microsoft.NET\Framework\v1.0.3328
39 - LOGONSERVER=\\SONNE2000
40 - PROCESSOR_REVISION=0404
41 - ProgramFiles=C:\Programme
42 - NUMBER_OF_PROCESSORS=1
43 - CommonProgramFiles=
    C:\Programme\Gemeinsame Dateien
44 - TMP=C:\DOKUME~1\hs\LOKALE~1\Temp
45 - APPDATA=C:\Dokumente und Einstellungen\hs
    \Anwendungsdaten
46 - ComSpec=C:\WINDOWS\system32\cmd.exe
47 - PROCESSOR_IDENTIFIER=x86 Family 6 Model 4
    Stepping 4, AuthenticAMD
48 - HOMEDRIVE=C:
49 - COMPUTERNAME=MARS
50 - PROCESSOR_LEVEL=6
51 - OS=Windows_NT
52 - SystemRoot=C:\WINDOWS
53 - USERDNSDOMAIN=IT-VISIONS.DE
54 - windir=C:\WINDOWS
55 - ALLUSERSPROFILE=C:\Dokumente und Einstellungen
    \All Users
56 - USERNAME=hs
57 - PATHEXT=.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;
    .WSF;.WSH
58 - USERDOMAIN=SONNENSYSYSTEM
59 - HOMEPATH=\Dokumente und Einstellungen\hs
60 - PROCESSOR_ARCHITECTURE=x86
61 - TEMP=C:\DOKUME~1\hs\LOKALE~1\Temp
62 - LIB=E:\VSNET\FrameworkSDK\Lib\;E:\Dev\wmisd\lib
63 - VSCOMNTOOLS="E:\VSNET\Common7\Tools\"
64 - SESSIONNAME=Console
65 - Cor_Debugging_Control_424242=1

```

4.1.1 Weitere Klassen im Namespace System

Die folgende Tabelle enthält eine knappe Erläuterung zu weiteren Klassen im Namespace `System`. Die als abstrakt gekennzeichneten Klassen können nicht instanziiert werden, sondern nur der Erzeugung abgeleiteter Klassen dienen.

Weitere Klassen

Klasse	Funktion	Abstrakte Klasse?
<code>System.AppDomain</code>	Diese Klasse dient der Erzeugung von Application Domains.	Nein
<code>System.EventArgs</code>	Diese Klasse ist die Oberklasse für Klassen, die als Parameter von Ereignissen zurückgeliefert werden.	Nein, aber Instanziierung nur durch CLR vorgesehen
<code>System.Attribute</code>	Oberklasse für alle Meta-Attribute	Ja
<code>System.Type</code>	Klasse zum Zugriff auf Metadaten eines beliebigen Typs	Ja
<code>System.ValueType</code>	Oberklasse für Wertetypen	Ja
<code>System.TimeZone</code>	Repräsentiert eine Zeitzone	Ja

Tabelle 4.8: Weitere Klassen im Namespace System

5 Namespace System.Collections

Der Namespace `System.Collections` enthält Klassen, in denen Objektmengen gespeichert werden können. Zu den Möglichkeiten, die diese Klassen bieten, zählen ganz normale Listen, FIFO- und LIFO-Speicher, Arrays, Hash-Tabellen und Dictionaries (Mengen mit Schlüssel-/Werte-Paaren). Unter diesem Namespace gibt es einen weiteren, `System.Collections.Specialized`, der, wie der Name schon sagt, spezialisierte Listen enthält.

**System.
Collections**

5.1 Klassen in System.Collections

Die interessantesten enthaltenen Klassen des Namespaces `System.Collections` finden Sie in Tabelle 5.1.

Klasse	Funktion
<code>ArrayList</code>	Implementiert ein dynamisches Array aus Objekten, das seine Größe automatisch verändert, wenn neue Werte hinzugefügt werden
<code>BitArray</code>	Implementiert ein Array aus Bit-Werten. Alle Werte werden als boolesche Werte interpretiert, d.h. ist ein Bit 1, wird <code>true</code> zurückgeliefert, ist es 0, <code>false</code> .
<code>CaseInsensitiveComparer</code>	Vergleicht zwei Objekte. Groß-/Kleinschreibung wird nicht berücksichtigt.
<code>CollectionBase</code>	Abstrakte Klasse. Repräsentiert die Grundlage einer streng typisierten Collection.
<code>Comparer</code>	Vergleicht zwei Objekte unter Berücksichtigung der Groß-/Kleinschreibung

Tabelle 5.1: Die Klassen des Namespaces `System.Collections`

Klasse	Funktion
DictionaryBase	Abstrakte Klasse. Repräsentiert die Grundlage einer Collection auf Basis von Schlüssel-/Werte-Paaren.
HashTable	Repräsentiert eine HashTable, also eine Sammlung von Schlüssel-/Werte-Paaren, die basierend auf dem Hash-Code des Schlüssels organisiert sind
Queue	Repräsentiert einen FIFO-Speicher (First In – First Out)
SortedList	Repräsentiert eine sortierte Liste. Auch bei dieser Liste handelt es sich um eine Schlüssel-/Wert-Kombination, die nach dem Schlüssel sortiert ist. Der Zugriff erfolgt entweder über den Schlüssel oder über einen Index.
Stack	Repräsentiert eine LIFO-Struktur (Last In – First Out)

Tabelle 5.1: Die Klassen des Namespaces System.Collections (Fortsetzung)

Alle im Namespace enthaltenen Klassen sind von System.Object abgeleitet. Das Diagramm in Abbildung 5.1 zeigt die komplette Hierarchie inklusive der Klassen aus System.Collections.Specialized.

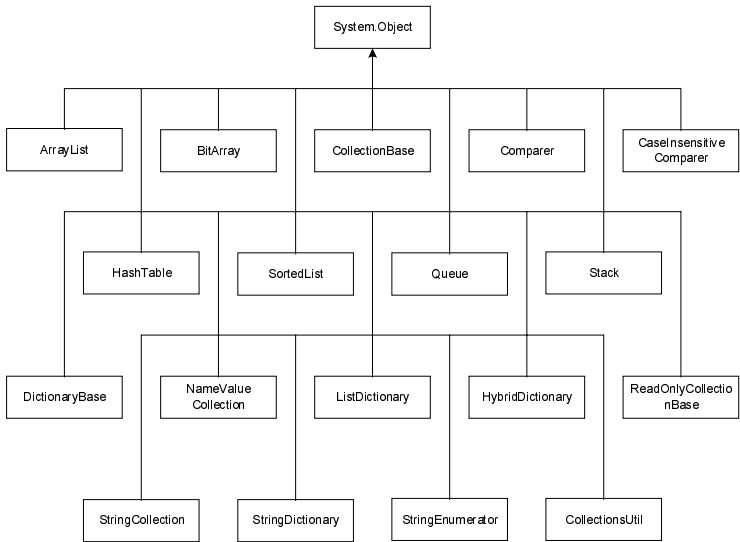


Abbildung 5.1: Die Hierarchie der Namespaces System.Collections und System.Collections.Specialized

Weiterhin sind einige Schnittstellen (Interfaces) in diesem Namespace enthalten, deren Funktionalität für die einzelnen Klassen benötigt wird. Die Schnittstellen werden, falls nötig, im Laufe dieses Kapitels angesprochen. Interessanter ist aber, wie man mit den Klassen, die der Namespace zur Verfügung stellt, arbeiten kann.

Schnittstellen

5.1.1 Die Klasse ArrayList

Die Klasse `ArrayList` wird vermutlich sehr häufig verwendet werden. Ganz gleich, welche Daten Sie speichern wollen, mit der `ArrayList` ist das grundsätzlich möglich. Die Daten werden als `Object` gespeichert, wodurch es wirklich nicht von Interesse ist, welche Art von Daten Sie in Ihrer `ArrayList` unterbringen.

ArrayList

Die `ArrayList` ist auch hinsichtlich der Möglichkeiten die umfangreichste Liste in diesem Namespace, da viele Dinge nicht vorgegeben sind. So kann z. B. in einem `Stack` ein neues Objekt immer nur obenauf gelegt werden, in einer `Queue` kann es immer nur von hinten in die Schlange gestellt werden – bei einer `ArrayList` ist es prinzipiell egal, wo und wie die Elemente eingefügt werden.

Sollen Elemente am Ende der `ArrayList` einfach hinzugefügt werden, so können Sie die Methode `Add()` benutzen. Sie fügt ein Element am Ende der Liste an und vergrößert gegebenenfalls die Kapazität der Liste. `AddRange()` dient ebenso zum Anfügen neuer Elemente, fügt aber nicht ein Element an, sondern eine ganze Sammlung von Elementen.

Add() und AddRange()

Wollen Sie neue Elemente irgendwo mitten in der `ArrayList` einfügen, verwenden Sie `Insert` oder `InsertRange()`. Ebenso wie bei `Add()` fügt `Insert()` ein einziges Element ein, während `InsertRange()` dazu gedacht ist, mehrere Elemente auf einen Schlag hinzuzufügen.

Insert() und InsertRange()

Das Entfernen von Elementen gestaltet sich ebenso einfach. Über die Methode `Remove()` können Sie ein Element entfernen, wenn Sie wissen, welches Objekt Sie entfernen wollen. Gibt es mehrere Objekte, so wird das erste auftretende Objekt entfernt. `RemoveAt()` arbeitet über den Index, `RemoveRange()` ermöglicht das Entfernen mehrerer Elemente auf einen Schlag.

Remove(), RemoveAt() und RemoveRange()

Werte hinzufügen und entfernen

In diesem Beispiel wird eine `ArrayList` benutzt, um Werte hinzuzufügen oder zu entfernen. Alle Werte werden in einer `Listbox` mit

Werte hinzufügen und entfernen

Namen `lbxValues` angezeigt. Die eigentliche `ArrayList` mit Namen `myList` ist als Feld des Formulars deklariert.

```
'=====
' SimpleArrayList.sln
' Copyright 2002 Frank Eller
' Hinzufügen und Entfernen von Werten
'=====

Public Class Form1
    Inherits System.Windows.Forms.Form

    #Region " Windows Form Designer generated code "

        'Der Code, der von VS.NET eingefügt wurde, wird
        'aus Übersichtsgründen nicht dargestellt.

    #End Region

    Private myList As ArrayList

    Private Sub FillListBox()
        lbxValues.BeginUpdate()
        lbxValues.Items.Clear()
        lbxValues.Items.AddRange(myList.ToArray())
        lbxValues.EndUpdate()
    End Sub

    Private Sub btnAdd_Click( _
        ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles btnAdd.Click

        If Not (tbxValue.Text = String.Empty) Then
            myList.Add(tbxValue.Text)
            FillListBox()
        End If
    End Sub

    Private Sub btnRemove_Click( _
        ByVal sender As System.Object, _
        ByVal e As System.EventArgs) _
        Handles btnRemove.Click

        If lbxValues.SelectedIndex > -1 Then
            myList.RemoveAt(lbxValues.SelectedIndex)
            FillListBox()
        End If
    End Sub

End Class
```

Listing 5.1: SimpleArrayList.exe

Auch bei der Eigenschaft `Items` der `Listbox` handelt es sich eigentlich um ein `Collection`-Objekt, das wie eine `ArrayList` auch eine Methode `Add()` bzw. `AddRange()` implementiert. Zur Verwendung von `AddRange()` wird allerdings ein `Object`-Array benötigt. Die Klasse `ArrayList` beinhaltet eine Methode namens `ToArray()`, mit der die enthaltenen Objekte in ein `Array` überführt werden können. Mit diesen Werten wird die `Listbox` dann gefüllt. Das Beispielprogramm zur Laufzeit sehen Sie in Abbildung 5.2.

ToArray()

Abbildung 5.2: Das Programm zur Laufzeit [SimpleArrayList.sln]

Unterschiedliche Datentypen eintragen

Ein Vorteil einer `ArrayList` ist, dass darin Werte unterschiedlicher Art gespeichert werden können. Z.B. können `Integer`-Werte gemeinsam mit `String`-Werten gespeichert werden. Sollen diese Werte wieder ausgelesen werden, müssen sie mittels `Casting` wieder in den ursprünglichen Datentyp zurückkonvertiert werden. Den ursprünglichen Datentyp erhalten Sie über die Methode `GetType()` der enthaltenen Objekte. Diese zeigt nicht etwa `System.Object` als Datentyp an, sondern den Typ des enthaltenen Elements, also `System.String` für Zeichenketten und `System.Int32` für `Integer`-Werte.

Werte unterschiedlicher Datentypen

In diesem Beispiel werden der Liste Werte unterschiedlicher Herkunft hinzugefügt, `Strings` und `Integer`-Werte durcheinander, ganz wie Sie wollen. Gespeichert werden sie ohnehin alle als `Object`. Wenn eines der Elemente ausgewählt wird, wird der tatsächliche Datentyp in einer `Textbox` angezeigt.

```

'=====
'DifferentValues.sln
'Copyright 2002 Frank Eller
' Unterschiedliche Datentypen in ArrayList
'=====

Public Class Form1
    Inherits System.Windows.Forms.Form

#Region " Windows Form Designer generated code "

    'Der Code, der von VS.NET eingefügt wurde, wird
    'aus Übersichtsgründen nicht dargestellt.

#End Region

    Private myList As New ArrayList()

    Private Sub FillListBox()
        lbxValues.BeginUpdate()
        lbxValues.Items.Clear()
        lbxValues.Items.AddRange(myList.ToArray())
        lbxValues.EndUpdate()
    End Sub

    Private Sub tbxInt_KeyPress(ByVal sender As Object, _
        ByVal e As System.Windows.Forms.KeyPressEventArgs) _
        Handles tbxInt.KeyPress

        If Not Char.IsDigit(e.KeyChar) Then
            e.Handled = True
        End If

    End Sub

    Private Sub btnAddString_Click(ByVal sender As
System.Object, _
        ByVal e As System.EventArgs) Handles btnAddString.Click

        If Not (tbxString.Text = String.Empty) Then
            myList.Add(tbxString.Text)
            FillListBox()
        End If

    End Sub

    Private Sub btnAddInt_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles btnAddInt.Click

        If Not (tbxInt.Text = String.Empty) Then
            myList.Add(Convert.ToInt32(tbxInt.Text))

```

```

        FillListBox()
    End If

End Sub

Private Sub lbxValues_SelectedIndexChanged( _
    ByVal sender As Object, ByVal e As System.EventArgs) _
    Handles lbxValues.SelectedIndexChanged

    Dim o As Object = lbxValues.SelectedItem
    tbxType.Text = o.GetType().ToString()
End Sub

Private Sub btnRemove_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnRemove.Click

    If lbxValues.SelectedIndex > -1 Then
        myList.RemoveAt(lbxValues.SelectedIndex)
        FillListBox()
    End If

End Sub

End Class

```

Listing 5.2: Verschiedene Datentypen in einer ArrayList [DifferentValues.sln]

Die Methoden zum Hinzufügen und Entfernen kennen Sie bereits. Sie unterscheiden sich eigentlich nicht von den Methoden des vorherigen Beispiels, sieht man einmal von der Umwandlung des Werts aus `lbxInt` in einen `Integer` ab. In der Methode `SelectedIndexChanged()`, einer Ereignisbehandlungsroutine der Listbox, wird der enthaltene Datentyp angezeigt. Abbildung 5.3 zeigt das Programm zur Laufzeit.

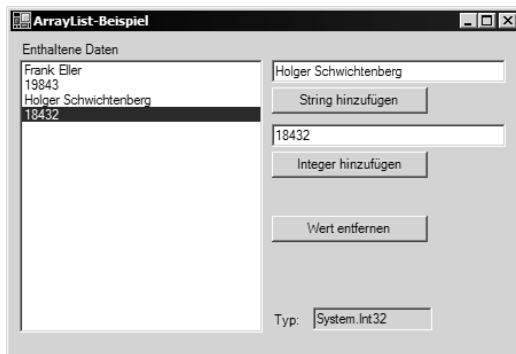


Abbildung 5.3: Strings und Integer-Werte gemischt [DifferentValues.sln]

ArrayList aufsteigend sortieren

Sort() Die Klasse `ArrayList` beinhaltet bereits eine Standardmethode zum Sortieren von Werten, die es aber auch erlaubt, eigene Sortierungen zu benutzen. So kann festgelegt werden, auf welche Art sortiert werden soll (standardmäßig wird vom kleinsten zum größten Wert sortiert). Allerdings erscheint die Vorgehensweise ein wenig umständlich. Die Sortierung auf die standardmäßige Art geschieht einfach durch einen Aufruf der Methode `Sort()` ohne weitere Parameter.

IComparer Soll die Art der Sortierung geändert werden, wird eine Implementierung der Schnittstelle `IComparer` benötigt. Diese fordert lediglich die Deklaration einer einzigen Methode, `Compare()`, die zwei Objekte vom Typ `Object` als Parameter erwartet und einen `Integer`-Wert zurückliefert. Dieser Wert ist

- ▶ gleich 0, wenn beide übergebenen Werte gleich sind,
- ▶ kleiner 0, wenn der erste Wert kleiner als der zweite ist,
- ▶ größer 0, wenn der zweite Wert größer als der erste ist.

Gemeint ist dabei allerdings, dass bei einem zurückgelieferten Wert kleiner 0 der erste Wert vor dem zweiten in der sortierten Liste angeordnet wird und dass bei einem Ergebniswert größer 0 der zweite Wert vor dem ersten eingeordnet wird. Damit kann man durch die Implementierung einer eigenen Klasse, die das Interface `IComparer` implementiert, die Sortierreihenfolge festlegen. Beispielsweise könnte man Strings nach der Länge sortieren oder die standardmäßig aufsteigende Sortierung ändern.

ArrayList absteigend sortieren

Das folgende Beispiel zeigt die Verwendung einer solchen Klasse mit dem Interface `IComparer`. Hier gilt, dass es sich um eine Liste mit `Integer`-Werten handeln muss; sortiert wird absteigend, also der größte Wert zuerst.

Zunächst die Methoden, die bereits aus den anderen Beispielen dieses Kapitels bekannt sind.

```
'=====
' SortDown.sln
' Copyright 2002 Frank Eller
' Sortiert eine ArrayList absteigend
'=====
```

```
Public Class Form1
    Inherits System.Windows.Forms.Form

    #Region " Windows Form Designer generated code "

        'Der Code, der von VS.NET eingefügt wurde, wird
        'aus Übersichtsgründen nicht dargestellt.

    #End Region

    Private myList As New ArrayList()

    Private Sub FillListBox()
        lbxValues.BeginUpdate()
        lbxValues.Items.Clear()
        lbxValues.Items.AddRange(myList.ToArray())
        lbxValues.EndUpdate()
    End Sub

    Private Sub tbxValue_KeyPress( _
        ByVal sender As Object, _
        ByVal e As System.Windows.Forms.KeyPressEventArgs) _
        Handles tbxValue.KeyPress

        If Not Char.IsDigit(e.KeyChar) Then
            e.Handled = True
        End If

    End Sub

    Private Sub btnAdd_Click( _
        ByVal sender As System.Object, _
        ByVal e As System.EventArgs) _
        Handles btnAdd.Click

        If Not (tbxValue.Text = String.Empty) Then
            myList.Add(Convert.ToInt32(tbxValue.Text))
            FillListBox()
        End If

    End Sub

    Private Sub btnRemove_Click( _
        ByVal sender As System.Object, _
        ByVal e As System.EventArgs) _
        Handles btnRemove.Click

        If lbxValues.SelectedIndex > -1 Then
            myList.RemoveAt(lbxValues.SelectedIndex)
            FillListBox()
        End If

    End Sub
```

```

End Sub

Private Sub btnSort_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnSort.Click

    myList.Sort(New MyComparer())
    FillListBox()

End Sub
End Class

```

Listing 5.3: Der Quellcode des Hauptformulars [SortDown.sln]

Der Unterschied liegt in der Ereignisbehandlungsroutine des Sortier-Buttons. Der Sort()-Methode der ArrayList wird ein Objekt vom Typ MyComparer übergeben. In der Klasse MyComparer ist eine eigene Compare()-Methode implementiert. Damit das funktioniert, muss das Interface IComparer implementiert werden, das sich ebenfalls im Namespace System.Collections befindet.

```

Public Class MyComparer
    Implements IComparer

    Public Function Compare(ByVal x As Object, _
        ByVal y As Object) As Integer _
        Implements IComparer.Compare

        Dim intX As Integer = CInt(x)
        Dim intY As Integer = CInt(y)

        If intX = intY Then
            Return 0
        Else
            Return IIf(intX < intY, 1, -1)
        End If
    End Function

End Class

```

Listing 5.4: Die Klasse MyComparer [SortDown.sln]

- IIf** Man könnte hier auch nur mit IIf arbeiten, die Routine würde dann ebenfalls funktionieren.

Weitere Methoden

Synchronized() Eine Collection, also auch eine ArrayList, ist von Haus aus nicht threadsicher. Es ist jedoch möglich, einen Wrapper um die Array

List zu legen und sie so threadsicher zu machen. Die dazu benötigte Methode heißt `Synchronized()` und ist eine statische Methode der Klasse `ArrayList`.

`Synchronized()` liefert wiederum als Ergebnis ein nun sicheres `ArrayList`-Objekt zurück. Die Zeilen

```
Dim ar1 As New ArrayList()  
Dim syncAr1 As ArrayList = ArrayList.Synchronized(ar1)
```

liefern `ar1` als nicht-synchronisierte `ArrayList` und `syncAr1` als synchronisierte `ArrayList` zurück. Über die Eigenschaft `IsSynchronized` können Sie kontrollieren, ob eine `ArrayList` synchronisiert, also threadsicher, ist.

Eine weitere Methode, die unter Umständen recht nützlich werden kann, ist `FixedSize()`. Auch hierbei handelt es sich um eine statische Methode der Klasse `ArrayList`. Nach dem Aufruf der Methode, der ebenso funktioniert wie bei `Synchronized()`, ist die `ArrayList` gegen das Hinzufügen von Elementen geschützt, die Größe der Liste ist festgelegt. Ob eine `ArrayList` eine feste Größe besitzt, können Sie über die Eigenschaft `IsFixedSize` kontrollieren.

Obwohl die Größe der `ArrayList` nun festgelegt ist, handelt es sich doch nicht um einen Schreibschutz. Sie können zwar keine weiteren Werte mehr hinzufügen, wohl aber bestehende Werte ändern.

Wollen Sie dagegen eine `ArrayList` gegen das Beschreiben schützen, verwenden Sie einfach die statische Methode `ReadOnly()`. Auch hier entspricht die Funktionsweise den Methoden `Synchronize()` und `FixedSize()`. Jetzt ist die `ArrayList` wirklich schreibgeschützt, die Werte können nur noch ausgelesen werden. Ob es sich so verhält, erfahren Sie durch Auswertung der Eigenschaft `IsReadOnly`.

FixedSize()



ReadOnly()

5.1.2 Die Klassen Stack und Queue

Zwei spezielle Listenarten werden durch die Klassen `Stack` und `Queue` implementiert. Erstere implementiert eine so genannte *Last-In-First-Out*-Struktur, d.h. der Wert, der als letzter hinzugefügt wurde, wird als erster wieder herausgenommen. Die zweite Klasse implementiert das Gegenteil, eine *First-In-First-Out*-Struktur, d.h. der Wert, der als erster eingefügt wurde, wird auch als erster wieder entnommen. Wie die beiden Strukturen arbeiten, können Sie in Abbildung 5.4 sehen.

FIFO und LIFO

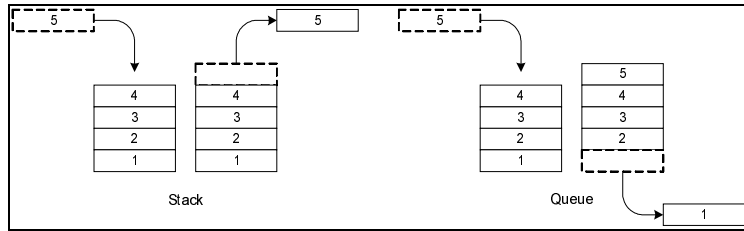


Abbildung 5.4: Stack und Queue

Queue und Stack Die Klasse `Queue` könnte z.B. Verwendung finden in einem Parser für Rechenoperationen, der die Zahlen und die Rechenoperationen in der Reihenfolge ablegen will, in der sie später auch wieder aufgerufen werden sollen. Die Klasse `Stack` könnte verwendet werden, um Zwischenergebnisse zu speichern, die dann später in entgegengesetzter Reihenfolge wieder ausgelesen werden.

Push() und Pop() Anders als bei einer `ArrayList` gibt es bei einem `Stack` und einer `Queue` keine Methoden, um beispielsweise Daten irgendwo in der Mitte einzufügen oder Daten über einen Index auszulesen. Für diese Klassen ist klar definiert, dass die einzelnen Elemente an einer bestimmten Stelle eingefügt werden und dass immer nur ein bestimmtes Element entfernt werden kann. Die Methode, mit der man einem `Stack`-Objekt ein Element hinzufügen kann, heißt `Push()`, die Methode zum Entfernen eines Elements `Pop()`.

Enqueue() und Dequeue() Entsprechende Methoden gibt es auch bei der Klasse `Queue`. Dort allerdings heißen sie `Enqueue()`, um ein Element in die Schlange hineinzustellen, und `Dequeue()`, um es zu entfernen. Beide Klassen speichern die Daten als Daten vom Typ `Object`, d.h. auch hier können alle Arten von Datentypen verwendet werden.

Peek() Falls Sie sich ansehen wollen, welches Element als nächstes aus der Liste entfernt würde, wenn Sie `Pop()` respektive `Dequeue()` aufrufen, können Sie bei beiden Klassen die Methode `Peek()` verwenden. Diese liefert das oberste Element der Liste, genauer gesagt, das nächste Element, das entfernt würde, allerdings ohne es zu entfernen.

Stack und Queue verwenden

Ein simples Beispiel soll die Verwendung von `Stack` und `Queue` illustrieren. Wir stellen einfach zwei Listboxen dar, wobei wir in einer den Inhalt eines `Stack`-Objekts anzeigen und in der anderen den Inhalt eines `Queue`-Objekts. Listing 5.5 zeigt den Programmcode.

```
'=====
' StackAndQueue.exe
' Copyright 2002 Frank Eller
' Zeigt die Verwendung von Stack und Queue
'=====
```

```
Public Class Form1
    Inherits System.Windows.Forms.Form

    #Region " Windows Form Designer generated code "

    'Der Code, der von VS.NET eingefügt wurde, wird
    'aus Übersichtsgründen nicht dargestellt.

    #End Region

    ' --- Felder
    Private aStack As New Stack()
    Private aQueue As New Queue()

    Private Sub FillListBoxes()

        ' --- Listboxen füllen
        lbxStack.BeginUpdate()
        lbxStack.Items.Clear()
        lbxStack.Items.AddRange(aStack.ToArray())
        lbxStack.EndUpdate()

        lbxQueue.BeginUpdate()
        lbxQueue.Items.Clear()
        lbxQueue.Items.AddRange(aQueue.ToArray())
        lbxQueue.EndUpdate()
    End Sub

    Private Sub btnInit_Click( _
        ByVal sender As System.Object, _
        ByVal e As System.EventArgs) _
        Handles btnInit.Click

        ' --- Stack und Queue leeren
        aStack.Clear()
        aQueue.Clear()
        FillListBoxes()
    End Sub

    Private Sub btnAddStack_Click( _
        ByVal sender As System.Object, _
        ByVal e As System.EventArgs) _
        Handles btnAddStack.Click
```

```
' --- Wert zu Stack hinzufügen
If Not (tbxAddStack.Text = String.Empty) Then
    aStack.Push(tbxAddStack.Text)
    FillListBoxes()
End If

End Sub

Private Sub btnAddQueue_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnAddQueue.Click

    ' --- Wert zu Queue hinzufügen
    If Not (tbxAddQueue.Text = String.Empty) Then
        aQueue.Enqueue(tbxAddQueue.Text)
        FillListBoxes()
    End If

End Sub

Private Sub btnGetStack_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnGetStack.Click

    ' --- Wert aus Stack auslesen
    If aStack.Count > 0 Then
        tbxStackValue.Text = aStack.Pop().ToString()
        FillListBoxes()
    End If

End Sub

Private Sub btnGetQueue_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnGetQueue.Click

    ' --- Wert aus Queue auslesen
    If aQueue.Count > 0 Then
        tbxQueueValue.Text = aQueue.Dequeue().ToString()
        FillListBoxes()
    End If

End Sub

Private Sub button3_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles button3.Click
```

```

' --- Beenden
Me.Close()
End Sub
End Class

```

Listing 5.5: StackAndQueue.exe

Wie in den anderen Listen auch können die Elemente von Stack und Queue mit Hilfe der Methode `Clear()` gelöscht werden. Auch das Füllen der Listboxen kennen Sie schon von den `ArrayList`-Beispielen.

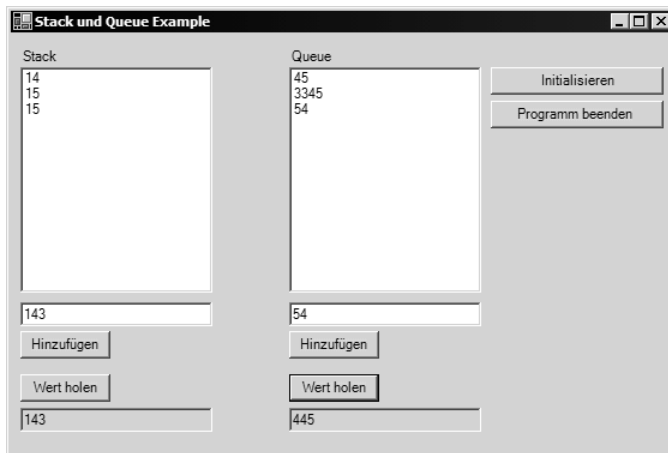


Abbildung 5.5: Stack und Queue zur Laufzeit (Example_13_04)

5.1.3 Die Klasse BitArray

Ein `BitArray` dient zum Speichern boolescher Werte. Wenn Sie ein neues Objekt vom Typ `BitArray` erstellen, müssen Sie im Konstruktor zumindest die Basislänge angeben. Allerdings ist es bei einem `BitArray` auch möglich, diese Größe nachträglich zu ändern. Die Eigenschaft `Length` dient hierbei der Größenänderung.

Setzen Sie `Length` auf einen Wert, der kleiner als die bisherige Länge ist, werden alle darüber liegenden Werte abgeschnitten. Stellen Sie einen Wert ein, der größer als die bisherige Länge ist, werden neue Elemente hinzugefügt und auf den Wert `False` gesetzt. Anders als bei der Redimensionierung eines normalen Arrays mittels `ReDim` werden bestehende Werte nicht gelöscht.

Das Auslesen eines Bits erfolgt über die Methode `Get()`, das Setzen eines Bits über die Methode `Set()`. Sie haben auch die Möglichkeit,

BitArray**Length****Bits setzen
und auslesen**

**Vergleichs-
methoden**

alle Bits des Arrays auf einmal auf einen bestimmten Wert zu setzen, indem Sie die Methode `SetAll()` verwenden.

`BitArray` besitzt weiterhin Methoden, die boolesche Operationen durchführen. Die Vergleiche werden immer mit den Bits eines anderen `BitArray`-Objekts durchgeführt. Die entsprechenden Methoden sind `And()`, `Or()`, `XOr()` und `Not()`. Zurückgeliefert wird ein weiteres `BitArray`-Objekt, das das Ergebnis beinhaltet.

BitArray verwenden

In diesem einfachen Beispiel wird ein `BitArray`-Objekt dazu benutzt, boolesche Werte zu speichern und wieder zu laden. Das Einstellen der Werte geschieht mit Hilfe von Checkboxes. Jeder Checkbox wurde ein Tag-Wert zugewiesen, in diesem Beispiel von 0 bis 9. Damit kann sehr einfach darauf zugegriffen werden, indem die Collection `Controls` des Formulars durchlaufen und jeweils der Wert des `BitArray`-Objekts, dessen Position dem eingestellten Tag-Wert entspricht, ausgelesen oder geschrieben wird.

Den Quellcode sehen Sie in Listing 5.1

```
'=====
' BitArrayExample.exe
' Copyright 2002 Frank Eller
' Zeigt die Verwendung der Klasse BitArray
'=====

Public Class Form1
    Inherits System.Windows.Forms.Form

    #Region " Windows Form Designer generated code "

        'Der Code, der von VS.NET eingefügt wurde, wird
        'aus Übersichtsgründen nicht dargestellt.

    #End Region

    ' --- Felder deklarieren
    Private Values As New BitArray(10)

    Private Sub btnWrite_Click( _
        ByVal sender As System.Object, _
        ByVal e As System.EventArgs) _
        Handles btnWrite.Click

        ' --- Deklarationen
        Dim ctrl As Control
```

```
' --- Alle Controls des Formulars prüfen
' --- Wenn Checkbox, dann Eigenschaft
' --- checked prüfen und entsprechenden
' --- Wert im BitArray setzen
For Each ctrl In Me.Controls
    If TypeOf ctrl Is CheckBox Then
        Dim cbx As CheckBox = CType(ctrl, CheckBox)
        Values.Set(cbx.Tag, cbx.Checked)
    End If
Next

End Sub

Private Sub btnRead_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnRead.Click

    ' --- Deklarationen
    Dim ctrl As Control

    ' --- Alle Controls des Formulars prüfen
    ' --- Wenn Checkbox, dann Eigenschaft
    ' --- checked setzen, wenn entsprechender
    ' --- Wert im BitArray True
    For Each ctrl In Me.Controls
        If TypeOf ctrl Is CheckBox Then
            Dim cbx As CheckBox = CType(ctrl, CheckBox)
            cbx.Checked = Values.Get(cbx.Tag)
        End If
    Next

End Sub

Private Sub btnExit_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnExit.Click

    ' --- Beenden
    Me.Close()

End Sub

End Class
```

Listing 5.6: Werte aus BitArray lesen und in BitArray schreiben [BitArrayExample.sln]

Abbildung 5.6 zeigt den Screenshot dieses kleinen Programms.

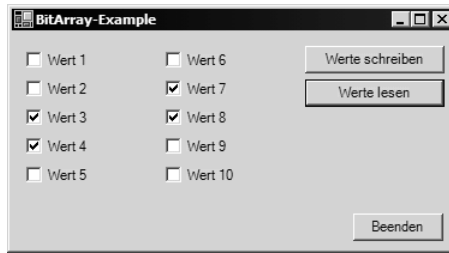


Abbildung 5.6: Der Screenshot von BitArrayExample

5.1.4 Die Klasse HashTable

HashTable Eine `HashTable` speichert Schlüssel-Wert-Paare, wobei ein Hashcode als Schlüssel verwendet wird. Genauer gesagt, wenn Sie ein Objekt als Schlüssel übergeben, wird aus diesem der Hashcode ermittelt und das Element basierend auf diesem Hashcode gespeichert. Ein Hashcode ist ein für jedes Objekt eindeutiger Integer-Wert, der unter anderem auch zur Verschlüsselung verwendet wird.

Hashcodes bereitstellen Die Verwendung des Hashcodes macht es notwendig, dass das Objekt, das als Schlüssel verwendet wird, auch einen Hashcode bereitstellen kann. Die Ermittlung dieses Codes kann auf zwei Arten geschehen: Entweder implementiert jedes Schlüsselobjekt eine eigene Methode, über die ein Hashcode ermittelt werden kann, oder man übergibt dem Konstruktor der `HashTable` ein Objekt, das die Schnittstelle `IHashCodeProvider` implementiert, und dessen Methode wird für alle Objekte zur Ermittlung des Hashcodes verwendet. Die Verwendung der Standardimplementation ist nicht zu empfehlen.

GetHashCode()-Standardimplementierung Zwar besitzen alle Klassen des .NET-Frameworks eine Methode `GetHashCode()`, die sie von `System.Object` erben. Diese Methode liefert aber in der Standardimplementierung nicht wirklich einen eindeutigen Hashcode, sondern vielmehr einen Index, der von der CLR für ein entsprechendes Objekt vergeben wird. Dieser kann wieder verwendet werden, nachdem die Garbage Collection das Objekt freigegeben hat.

Eigene GetHashCode()-Implementierung Dadurch können zwei unterschiedliche Objekte den gleichen Hashcode erhalten. Dies ist nicht wünschenswert, weshalb sie ihren eigenen Hash-Algorithmus bereitstellen können. Das funktioniert auf zwei Arten, nämlich:

- ▶ indem Sie die Methode `GetHashCode()` in eigenen Klassen überschreiben und so eine eigene Implementierung bereitstellen oder
- ▶ indem Sie `IHashCodeProvider` in einem Objekt implementieren und auf diese Weise den Hashcode ermitteln.

Auf die in der `HashTable` enthaltenen Schlüssel können Sie über die Eigenschaft `Keys` zugreifen, auf die Werte über die Eigenschaft `Values`. Dabei sind die Schlüssel zwar nicht sortiert, die Reihenfolge ist aber für `Keys` und `Values` die gleiche, so dass es keine Probleme beim Zugriff gibt. Der Zugriff auf die einzelnen Objekte geschieht über den Schlüssel, nicht über einen Index. Damit ist außerdem gewährleistet, dass die Schlüsselwerte für unterschiedliche Werte eindeutig sind, womit Sie z.B. auch kontrollieren können, ob ein bestimmtes Objekt bereits in der `HashTable` enthalten ist.

**Zugriff auf eine
HashTable**

Passwort-Test

Ein einfaches Beispiel soll zeigen, wie Sie die `HashTable` verwenden können. In einem Netzwerk ist der Benutzername üblicherweise eindeutig, wir können ihn also als Schlüssel für eine `HashTable` verwenden. Das dazugehörige Passwort wird zwar im Klartext in der `HashTable` abgelegt, ohne den passenden Namen nützt es aber nichts.

Für das Beispiel werden Namen und Passwörter im `FormLoad`-Ereignis der `HashTable` zugewiesen, normalerweise würden sie vermutlich aus einer Datei eingelesen. Der Name wird dann aus einer Combobox ausgelesen und mit dem eingegebenen Passwort verglichen. Stimmen beide überein, erhält der Benutzer Zugriff, andernfalls nicht.

Das Hauptformular besteht lediglich aus einer Combobox mit Namen `cbxNames` und einer Textbox mit Namen `txtPassword`, dazu zwei Labels und zwei Buttons. Der Quelltext sollte keine Probleme bereiten.

```
'=====
' AccessCheck.sln
' Copyright 2002 Frank Eller
' Verwenden eines Hashcodes
'=====
```

```

Public Class Form1
    Inherits System.Windows.Forms.Form

    #Region " Windows Form Designer generated code "

        'Der Code, der von VS.NET eingefügt wurde, wird
        'aus Übersichtsgründen nicht dargestellt.

    #End Region

    Private checkTable As New Hashtable()

    Private Sub Form1_Load( _
        ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles MyBase.Load

        ' --- Namen der Liste hinzufügen
        checkTable.Add("Frank Eller", "67FE24HOST")
        checkTable.Add("Holger Schwichtenberg", _
            "27HS43LOCAL")
        checkTable.Add("Klaus Kappler", "66KK39REMOTE")
        checkTable.Add("Sylvia Hasselbach", "43SH24HOST")

        FillComboBox()
    End Sub

    Private Sub FillComboBox()

        ' --- Deklarationen
        Dim oName As Object

        ' --- Combobox mit Daten füllen
        cbxNames.Items.Clear()
        For Each oName In checkTable.Values
            cbxNames.Items.Add(oName)
        Next
    End Sub

    Private Sub btnExit_Click( _
        ByVal sender As System.Object, _
        ByVal e As System.EventArgs) _
        Handles btnExit.Click

        ' --- Beenden
        Me.Close()

    End Sub

    Private Sub btnCheck_Click( _
        ByVal sender As System.Object, _
        ByVal e As System.EventArgs) _

```

```
Handles btnCheck.Click

' --- Deklarationen
Dim aPassword As String
Dim aName As String

' --- Überprüfen, ob Name und Passwort
' --- übereinstimmen
aPassword = txtPassword.Text
aName = cbxNames.SelectedItem

If checkTable(aName) = aPassword Then
    MessageBox.Show("Sie sind berechtigt", _
        "Berechtigung", MessageBoxButtons.OK, _
        MessageBoxIcon.Information)
Else
    MessageBox.Show("Sie haben keine Berechtigung", _
        "Berechtigung", MessageBoxButtons.OK, _
        MessageBoxIcon.Hand)
End If

End Sub
End Class
```

Listing 5.7: Zugriffskontrolle mit Namen und Passwort [Accesscheck.sln]

5.2 Eigene Collections erzeugen

Natürlich ist es möglich, mit den vom .NET-Framework vorgegebenen Collection-Klassen zu arbeiten. Weitaus angenehmer ist es allerdings, eine speziell auf die eigenen Bedürfnisse zugeschnittene Collection zu verwenden.

Collections sind eigentlich nur herkömmliche Klassen, die bestimmte Schnittstellen implementieren. Über die Methoden dieser Schnittstellen wird die gewünschte Funktionalität erreicht. Beispielsweise muss, um die `For Each`-Anweisung benutzen zu können, die Schnittstelle `IEnumerable` implementiert sein. Für die Methoden `Add()` und `Remove()`, die Sie bereits aus der `ArrayList` kennen, benötigt man die Schnittstelle `IList`. Alle Schnittstellen, die eine Rolle spielen, sind bereits im Namespace `System.Collections` deklariert.

Implementierte Schnittstellen

Einige dieser Schnittstellen benötigen die Funktionalität weiterer Schnittstellen, deren Methoden dann natürlich ebenfalls implementiert werden müssen. Tabelle 5.2 zeigt die verschiedenen Schnittstellen, ihre Abhängigkeiten und Funktionen im Überblick.

Schnittstelle (Interface)	Funktion
ICollection	Definiert Methoden bezüglich Größe und Synchronisation sowie den Enumerator für die Collection. Da ICollection von IEnumerable abgeleitet ist, muss auch dessen einzige Methode GetEnumerator implementiert werden, wenn ICollection benutzt wird.
IComparer	Definiert eine Methode Compare() zum Vergleichen zweier Werte bzw. Objekte. Diese Methode muss Folgendes liefern: 0, wenn beide Objekte gleich sind, -1, wenn das erste Objekt kleiner ist als das zweite, und +1, wenn das zweite Objekt kleiner ist als das erste.
IDictionary	Repräsentiert eine Collection von Name-Wert-Paaren. Bindet auch die Schnittstellen ICollection und IEnumerable ein.
IDictionary-Enumerator	Ist von IEnumerator abgeleitet und stellt einen Enumerator für Dictionary-Listen, also Listen mit Name-Wert-Paaren, dar. Auch die Methoden von IEnumerator müssen implementiert werden.
IEnumerable	Besitzt nur eine Methode, GetEnumerator(), die ein IEnumerator-Objekt zurückliefert (also eine beliebige Klasse, die IEnumerator implementiert)
IEnumerator	Bietet Methoden, mit denen über eine Collection iteriert werden kann, allerdings nur vorwärts. Eine Implementierung von IEnumerator wird für die For Each-Schleife benötigt.
IHashCodeProvider	Implementiert nur eine Methode, GetHashCode(), mit der der Hashcode eines Objekts (also ein eindeutiger Integer-Wert) ermittelt wird
IList	IList liefert Methoden für eine herkömmliche Collection, also Methoden zum Hinzufügen, Entfernen oder Auslesen von Daten. IList ist abgeleitet von ICollection und IEnumerable, d.h. deren Methoden müssen ebenfalls implementiert werden.

Tabelle 5.2: Die Schnittstellen in System.Collection

Je nach der Funktionalität, die Sie bereitstellen wollen, müssen Sie die entsprechenden Methoden implementieren. In den vordefinierten Collections sind diese Schnittstellen je nach Funktionalität ebenfalls implementiert. Abbildung 5.7 zeigt die Hierarchie der Schnittstellen.

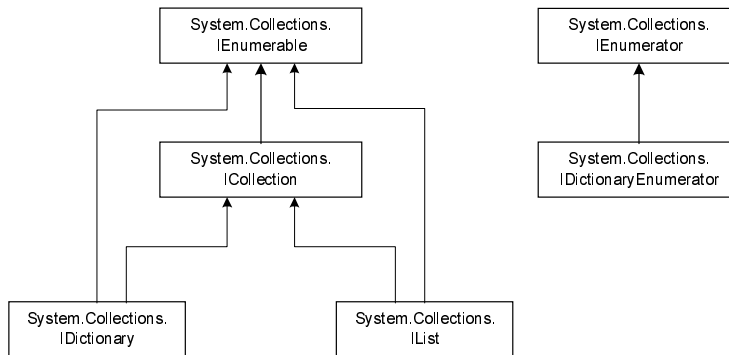


Abbildung 5.7: Hierarchie der Schnittstellen des Namespaces `System.Collections`

Eine Linkliste als Beispiel

Angenommen, Sie möchten eine Collection dazu benutzen, eine eigene Linkliste zu verwalten. Für jeden Link sollen sowohl eine Bezeichnung, der URL als auch eine Beschreibung angegeben werden. Sie könnten nun einerseits eine eigene Klasse erzeugen, die diese Daten zur Verfügung stellt, und Instanzen dieser Klasse in einer `ArrayList` speichern. Oder Sie können sich eine eigene Collection zusammenbauen, die genau die Funktionalität beinhaltet, die Sie sich wünschen.

In diesem Beispiel wird eine eigene Collection so angepasst, dass sie sinnvoll auf das Auslesen oder Setzen von Werten reagiert. Gespeichert werden Werte, die Links im Internet repräsentieren sollen. Wie angegeben sollen Bezeichnung, URL und Beschreibung gespeichert werden.

Zum Speichern der Werte wird eine Klasse namens `Link` benutzt. Den Aufbau dieser Klasse sehen Sie in Listing 5.8.

```

' =====
' CustomCollection.sln
' Copyright 2002 Frank Eller
' Klasse Link
' =====

Public Class Link

    ' --- Private Felder
    Private _name As String = String.Empty
    Private _url As String = String.Empty
    Private _description As String = String.Empty
  
```

```

' --- Eigenschaften (Properties)
Public Property Name() As String
    Get
        Return _name
    End Get
    Set(ByVal Value As String)
        _name = Value
    End Set
End Property

Public Property Url() As String
    Get
        Return _url
    End Get
    Set(ByVal Value As String)
        _url = Value
    End Set
End Property

Public Property Description() As String
    Get
        Return _description
    End Get
    Set(ByVal Value As String)
        _description = Value
    End Set
End Property

Public ReadOnly Property FullName() As String
    Get
        Return _name + "<" + _url + ">"
    End Get
End Property

' --- Methoden
Public Overrides Function ToString() As String

    ' --- Liefert einen String zurück
    Return _url
End Function

Public Overloads Function Equals( _
    ByVal Value As Object) As Boolean

    ' --- Vergleicht zwei Websites
    ' --- Zwei Websites sind gleich,
    ' wenn der URL gleich ist
    Return (Me.ToString() = Value.ToString)
End Function

' --- Konstruktoren

```

```

Public Sub New()
    Me._description = ""
    Me._name = ""
    Me._url = ""
End Sub

Public Sub New(ByVal n As String,
    ByVal u As String, ByVal d As String)
    Me._name = n
    Me._url = u
    Me._description = d
End Sub
End Class

```

Listing 5.8: Die Klasse Link [Links.vb]

In dieser Klasse wird die Funktion `Equals()` überladen, um zwei Webseiten korrekt zu vergleichen. Zwei Webseiten sind dann gleich, wenn es sich um den gleichen URL handelt. Weiterhin existiert eine Eigenschaft `FullName`, die nur Lesen erlaubt und Bezeichner sowie URL zurückliefert. Die Eigenschaft `Description` liefert die Beschreibung der Website zurück, die Standardmethode `ToString()` wird so überschrieben, dass nur der URL zurückgeliefert wird. Mit dieser Klasse als Basis wird nun eine Collection aufgebaut, die den Anforderungen besser gerecht wird als eine simple `ArrayList`.

**Equals()
überladen**

Die neue Collection soll die Schnittstelle `IList` implementieren. Damit wird zwangsläufig auch die Implementierung der Schnittstellen `ICollection` und `IEnumerable` notwendig, da `IList` von beiden abgeleitet ist. Zum Speichern der Elemente wird intern eine `ArrayList` benutzt, was den Vorteil hat, dass deren bereits vorhandene Implementationen benutzt werden können. Die neue Collection-Klasse soll den Namen `LinkList` tragen.

**IList imple-
mentieren**

Da die `LinkList`-Klasse nur für Objekte vom Typ `Link` geeignet sein soll, wird zunächst eine Methode implementiert, die kontrolliert, ob es sich um eine Website handelt:

```

' --- Kontrolle auf richtigen Datentyp
Private Function IsWebsite(ByVal value As Object) _
    As Boolean

    Dim checkLink As New Link()
    Return (value.GetType() Is checkLink.GetType())
End Function

```

Die Standard-Eigenschaft `Item` der Liste (in C# handelt es sich dabei um den Indexer) wird so implementiert, dass auch hier zwar eine Zuweisung erfolgen kann, aber kontrolliert wird, ob es sich auch wirklich um einen Link (also ein Objekt vom Typ `Link`) handelt. Dazu wird die eben erwähnte Methode `IsWebsite()` verwendet.

```
Default Public Property item( _
    ByVal index As Integer) As Object _
    Implements IList.Item
Get
    If index < Count Then
        Return _websites.Item(index)
    End If
    Return -1
End Get
Set(ByVal Value As Object)
    If index < Count Then
        If IsWebsite(Value) Then
            _websites(index) = Value
        End If
    End If
End Set
End Property
```

Alle weiteren Methoden werden implementiert, indem die entsprechende Methode der internen `ArrayList` aufgerufen wird, in der die Objekte gespeichert sind. Die gesamte Deklaration der Klasse `LinkList` zeigt Listing 5.9.

```
'=====
' CustomCollection.sln
' Copyright 2002 Frank Eller
' Klasse LinkList
'=====

Public Class LinkList
    Implements IList

    ' --- Die Liste zum Speichern der Elemente
    Private _websites As ArrayList

    ' --- Kontrolle auf richtigen Datentyp
    Private Function IsWebsite(ByVal value As Object) _
        As Boolean
        Dim checkLink As New Link()
        Return (value.GetType() Is checkLink.GetType())
    End Function

    ' --- IList implementieren
```



```
Public ReadOnly Property IsFixedSize() As Boolean _
    Implements IList.IsFixedSize
    Get
        Return False
    End Get
End Property
```

```
Public ReadOnly Property IsReadOnly() As Boolean _
    Implements IList.IsReadOnly
    Get
        Return False
    End Get
End Property
```

```
Public ReadOnly Property IsSynchronized() _
    As Boolean Implements IList.IsSynchronized
    Get
        Return _websites.IsSynchronized
    End Get
End Property
```

```
Default Public Property item( _
    ByVal index As Integer) As Object _
    Implements IList.Item
    Get
        If index < Count Then
            Return _websites.Item(index)
        End If
        Return -1
    End Get
    Set(ByVal Value As Object)
        If index < Count Then
            If IsWebsite(Value) Then
                _websites(index) = Value
            End If
        End If
    End Set
End Property
```

```
Public Function Contains(ByVal value As Object) _
    As Boolean Implements IList.Contains
    Return _websites.Contains(value)
End Function
```

```
Public Sub Remove(ByVal value As Object) _
    Implements IList.Remove
    If IsWebsite(value) Then
        _websites.Remove(value)
    End If
End Sub
```

```

Public Sub Insert(ByVal index As Integer, _
    ByVal value As Object) Implements IList.Insert
    If IsWebsite(value) Then
        _websites.Insert(index, value)
    End If
End Sub

Public Sub RemoveAt(ByVal index As Integer) _
    Implements IList.RemoveAt
    _websites.RemoveAt(index)
End Sub

Public Function Add(ByVal value As Object) _
    As Integer Implements IList.Add
    If IsWebsite(value) Then
        _websites.Add(value)
    End If
End Function

Public Sub Clear() Implements IList.Clear
    _websites.Clear()
End Sub

Public Function IndexOf(ByVal value As Object) _
    As Integer Implements IList.IndexOf
    If IsWebsite(value) Then
        Return _websites.IndexOf(value)
    End If
    Return -1
End Function

' --- ICollection implementieren
Public ReadOnly Property Count() As Integer _
    Implements ICollection.Count
    Get
        Return _websites.Count
    End Get
End Property

Public ReadOnly Property SyncRoot() As Object _
    Implements ICollection.SyncRoot
    Get
        Return _websites.SyncRoot
    End Get
End Property

Public Sub CopyTo(ByVal value As Array, _
    ByVal index As Integer) _
    Implements ICollection.CopyTo
    _websites.CopyTo(value, index)
End Sub

```

```

' --- IEnumerable implementieren
Public Function GetEnumerator() As IEnumerator _
Implements IEnumerable.GetEnumerator
    Return _websites.GetEnumerator()
' Return CType(New WebEnumerator(_websites), _
'     IEnumerator)
End Function

' --- Konstruktor
Public Sub New()
    _websites = New ArrayList()
End Sub
End Class

```

Listing 5.9: Die Klasse *LinkList* [*CustomCollection.sln*]

Da die *IEnumerator*-Implementierung aus der internen *ArrayList* absolut ausreichend ist, wurde hier keine eigene Implementierung programmiert. Eine solche ist jedoch auch sehr schnell zu erstellen. Im Beispielprojekt ist in der Methode *GetEnumerator()* die Zeile

IEnumerator

```

Return CType(New WebEnumerator(_websites), _
    IEnumerator)

```

auskommentiert. Wenn Sie eine eigene Implementierung von *IEnumerator* verwenden wollen, können Sie diese Zeile aktivieren und dafür die darüber stehende auskommentieren. Die entsprechende Klasse *WebEnumerator* ist selbstverständlich auch im Projekt enthalten und wie in Listing 5.10 deklariert.

```

' =====
' CustomCollection.sln
' Copyright 2002 Frank Eller
' Klasse WebEnumerator
' =====

Public Class WebEnumerator
    Implements IEnumerator

    Private _websites As ArrayList
    Private _currentValue As Integer = -1

    Public ReadOnly Property Current() As Object _
        Implements IEnumerator.Current
        Get
            Return _websites.Item(_currentValue)
        End Get
    End Property

    Public Function MoveNext() As Boolean _
        Implements IEnumerator.MoveNext

```

```

' --- Liefert True, wenn erfolgreich, und
' --- False, wenn die Anzahl der Elemente
' --- überschritten ist
_currentValue += 1
If _currentValue >= _websites.Count Then
    Return False
Else
    Return True
End If

End Function

Public Sub reset() Implements IEnumerator.Reset
    _currentValue = -1
End Sub

Public Sub New(ByVal websites As ArrayList)
    _websites = websites
End Sub

End Class

```

Listing 5.10: Die Klasse WebEnumerator (aus CustomCollection.sln)

Eigene Implementierung von IEnumerator

WebEnumerator implementiert eine einfache Zählvariable, die mit jedem Zugriff auf `MoveNext()` hochgezählt wird. Es muss lediglich kontrolliert werden, ob die Anzahl der tatsächlich in der `ArrayList` enthaltenen Elemente überschritten ist oder nicht.

Beim Zurücksetzen muss darauf geachtet werden, dass bei der Verwendung der `For Each`-Schleife zuerst `Reset()` und dann gleich `MoveNext()` aufgerufen werden. Erst danach wird das betreffende Element angefordert. Daher muss der Initialwert für die Zählvariable -1 sein, nicht 0.

Die grundsätzliche Funktion der neuen Collection ist damit bereits fertig. Zur Anzeige der Daten, zum Hinzufügen und Entfernen von Werten benötigen wir noch zwei Formulare: einmal das Hauptformular zur Anzeige und ein weiteres zum Eingeben oder Bearbeiten der Daten. Letzteres besteht eigentlich nur aus drei Textboxen mit den Bezeichnungen `txtName`, `txtUrl` und `txtDesc`. Weiterhin brauchen wir natürlich je einen Button zum Abbrechen und Bestätigen der Eingabe. Abbildung 5.8 zeigt den Aufbau des Eingabeformulars.

Die Eigenschaft `DialogResult` der beiden Buttons wurde für den OK-Button auf `Ok`, für den ABBRUCH-Button auf `Cancel` eingestellt. Damit wird das Formular beim Betätigen eines der Buttons auto-

matisch geschlossen und in der aufrufenden Methode muss nur noch kontrolliert werden, welcher `DialogResult`-Wert zurückgeliefert wurde.

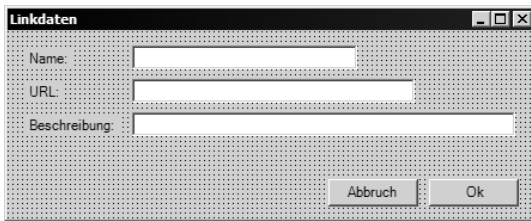


Abbildung 5.8: Das Formular für die Eingabe der Linkdaten

Das Hauptformular beinhaltet Methoden zum Hinzufügen, zum Ändern und zum Löschen von Daten. Angezeigt werden die Einträge in einer Listbox namens `lbnWebSites`. Wenn ein Eintrag ausgewählt wird, sollen die Details in drei Labels angezeigt werden, die ebenfalls selbstbeschreibende Namen haben: `lblName`, `lblLink` und `lblDescription`. Damit man sieht, dass es sich bei dem URL auch wirklich um eine Internet-Adresse handelt, wurde hier eine Komponente des Typs `LinkLabel` verwendet. Den Aufbau des Hauptformulars zeigt Abbildung 5.9.

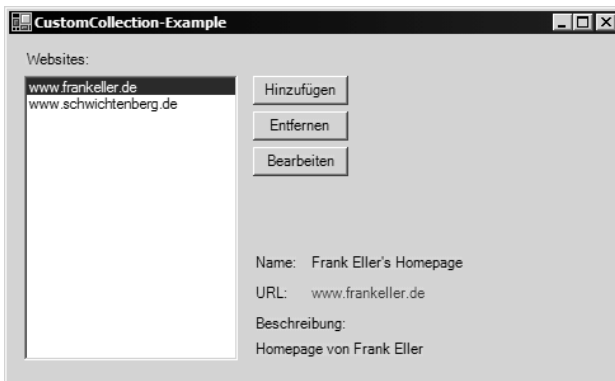


Abbildung 5.9: Der Aufbau des Hauptformulars

Zum Speichern der Werte wird natürlich eine Instanz der neuen Klasse `LinkList` benutzt, in diesem Fall heißt sie `WebSites`. Eine Methode namens `FillListBox()` füllt die Listbox mit den Werten. Diese Methode ist denkbar einfach aufgebaut:

```

Private Sub FillListBox()
    Dim currLink As Link

    lbxWebSites.Items.Clear()

    For Each currLink In WebSites
        lbxWebSites.Items.Add(currLink)
    Next
End Sub

```

Zum Hinzufügen wird das Eingabeformular aufgerufen und eine neue Link-Instanz erzeugt, wenn der Benutzer mit OK bestätigt hat.

```

Private Sub btnAdd_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnAdd.Click

    Dim LinksFrm As New FrmLinks()
    With LinksFrm
        If .ShowDialog() = DialogResult.OK Then
            Dim name As String = .txtName.Text
            Dim url As String = .txtUrl.Text
            Dim desc As String = .txtDesc.Text
            WebSites.Add(New Link(name, url, desc))
            FillListBox()
        End If
    End With
End Sub

```

Die Methode zum Ändern eines Werts funktioniert ähnlich, nur wird hier das Eingabefenster vor der Anzeige noch mit den Werten des aktuellen Links gefüllt.

```

Private Sub btnEdit_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnEdit.Click

    If lbxWebSites.SelectedIndex > -1 Then
        Dim i As Integer = lbxWebSites.SelectedIndex
        Dim currentLink As Link
        Dim LinksFrm As New FrmLinks()

        currentLink = CType(WebSites(i), Link)
        With LinksFrm
            .txtName.Text = currentLink.Name
            .txtUrl.Text = currentLink.Url
            .txtDesc.Text = currentLink.Description

            If LinksFrm.ShowDialog() = DialogResult.OK Then
                currentLink.Name = .txtName.Text
            End If
        End With
    End If
End Sub

```

```

        currentLink.Url = .txtUrl.Text
        currentLink.Description = .txtDesc.Text
        WebSites(i) = currentLink
    End If

    End With
End If
End Sub

```

Die Ereignisbehandlungsroutine für den Button zum Löschen eines Eintrags kontrolliert, ob in der Listbox etwas markiert ist, und löscht diesen Eintrag dann aus der Liste.

```

Private Sub btnRemove_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnRemove.Click

    If lbxWebSites.SelectedIndex > -1 Then
        WebSites.RemoveAt(lbxWebSites.SelectedIndex)
        FillListBox()
    End If
End Sub

```

Jetzt fehlt nur noch die Routine zum Anzeigen der Details, sobald ein Link aus der Listbox ausgewählt wurde. Sinnigerweise wird diese Funktionalität im Ereignis `SelectedIndexChanged()` der Listbox programmiert. Der Aufbau der Routine ist aber wieder denkbar einfach, da ja nur Werte ausgelesen werden. Der in der Listbox enthaltene Wert wird mittels `CType`-Anweisung in ein Objekt vom Typ `Link` umgewandelt und dann entsprechend angezeigt.

```

Private Sub lbxWebSites_SelectedIndexChanged( _
    ByVal sender As Object, _
    ByVal e As System.EventArgs) _
    Handles lbxWebSites.SelectedIndexChanged

    If lbxWebSites.SelectedIndex > -1 Then
        Dim i As Integer = lbxWebSites.SelectedIndex
        Dim currentLink As Link
        currentLink = CType(WebSites(i), Link)
        lblLink.Text = currentLink.Url
        lblDescription.Text = currentLink.Description
        lblName.Text = currentLink.Name
    End If

End Sub

```

Damit ist die Funktionalität hergestellt. Abbildung 5.10 zeigt das Programm im Einsatz. Wenn Sie möchten, können Sie natürlich

noch für das Link-Label eine Methode programmieren, die beim Klick darauf ein Browserfenster öffnet und zu dem angezeigten Link navigiert.

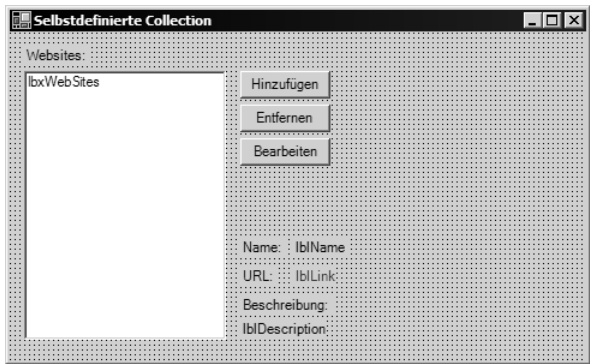


Abbildung 5.10: Die eigene Collection im Einsatz

5.3 Spezialisierte Collections

Alle Collection-Klassen in `System.Collections` sind mehr oder weniger allgemein gehalten, d.h. es werden Objekte vom Typ `System.Object` gespeichert – somit können alle Arten von Objekten gespeichert werden. Ein wenig spezieller wird es bei der Verwendung des Namespaces `System.Collections.Specialized`. Dort sind Collections enthalten, die für einen bestimmten Einsatz optimiert sind.

Collection-Klasse	Einsatzgebiet
<code>CollectionsUtil</code>	Wird benutzt für Collections, die Groß-/Kleinschreibung in Strings ignorieren
<code>HybridDictionary</code>	Repräsentiert eine Liste, die das Interface <code>IDictionary</code> implementiert. Solange die Liste klein ist, wird eine Liste vom Typ <code>ListDictionary</code> benutzt, die eine einfach verkettete Liste darstellt. Sobald die Liste größer wird, wird automatisch eine <code>HashTable</code> benutzt.
<code>ListDictionary</code>	Repräsentiert eine einfach verkettete Liste. Diese Liste kann verwendet werden, wenn nicht mehr als zehn Einträge verwaltet werden sollen, bei größeren Listen wirkt sich das negativ auf die Performance aus.

Tabelle 5.3: Die spezialisierten Collections aus `System.Collections.Specialized`

Collection-Klasse	Einsatzgebiet
NameObject CollectionBase	Eine abstrakte Klasse, von der abgeleitet werden muss. Sie repräsentiert eine Liste aus String- und Object-Werten. Der Zugriff ist entweder über den Schlüssel oder den Index möglich.
NameValueCollection	Repräsentiert eine sortierte Liste aus String-Schlüsselwerten und String-Werten. Der Zugriff erfolgt entweder über den Schlüssel oder den Index.
StringCollection	Repräsentiert eine einfache Collection für String-Werte
StringDictionary	Repräsentiert eine HashTable. Der Key-Wert muss ein String sein.

Tabelle 5.3: Die spezialisierten Collections aus *System.Collections.Specialized* (Fortsetzung)

Der Einsatz dieser spezialisierten Collections gestaltet sich nicht anders als bei den bereits besprochenen, lediglich die Art der Werte oder das Verhalten der Liste ist anders. Aus diesem Grund erübrigt sich eine genauere Betrachtung. Allerdings ist es sinnvoll, z.B. zum Speichern von Strings ein Objekt vom Typ `StringCollection` zu verwenden, da der Zugriff auf die enthaltenen Elemente dann ohne eine Umwandlung von `Object` in `String` möglich ist.

6 Namespace System.IO

Der Namespace `System.IO` enthält im Wesentlichen die Funktionen, die bisher durch die »klassische« COM-Bibliothek `File System Objects (FSO)`, die in der Scripting Runtime Library (`scrrun.dll`) implementiert war, bereitgestellt wurden, also

**Dateisystem-
operationen**

- ▶ den Zugriff auf Ordner und Dateien des Dateisystems
- ▶ das Lesen und Schreiben von Textdateien.

Microsoft hat sowohl Klassen-, Attribut- und Methodennamen als auch die grundsätzliche Vorgehensweise gegenüber FSO stark geändert.



Darüber hinaus bietet `System.IO` aber auch folgende Funktionen, die in FSO fehlten:

- ▶ das Lesen und Schreiben von Binärdateien
- ▶ die Überwachung von Veränderungen im Dateisystem

6.1 Zugriff auf die Struktur des Dateisystems

Microsoft bietet in der FCL zwei jeweils verschiedene Klassen für den Zugriff auf Dateien und Ordner in einem Dateisystem an. Der Dateizugriff ist möglich mit den Klassen `FileInfo` und `File`. Der Zugriff auf Ordner ist möglich mit den Klassen `DirectoryInfo` und `Directory`.

Der grundsätzliche Unterschied besteht darin, dass `File` und `Directory` nur statische Methoden (Shared Members) und keine Attribute bieten. Von diesen beiden Klassen können überhaupt keine Instanzen gebildet werden, da es keinen öffentlichen Konstruktor in diesen Klassen gibt. Sie sind also rein statisch. Sie bieten Aktionen wie Anlegen, Löschen und Verschieben und sind daher vergleichbar mit der Funktionalität der Klasse `FileSystemObject` im klassischen FSO-Objektmodell.

File und Directory

FileInfo und DirectoryInfo

FileInfo und DirectoryInfo sind nicht-statische Klassen, von denen Instanzen gebildet werden können. Sie bieten Attribute und Methoden. Bei der Instanziierung muss der Name der Datei bzw. des Verzeichnisses angegeben werden.

Klasse	Instanzierbar	Methoden	Attribute
Directory	Nein	Ja	Nein
DirectoryInfo	Ja	Ja	Ja
File	Nein	Ja	Nein
FileInfo	Ja	Ja	Ja

Tabelle 6.1: Gegenüberstellung der verschiedenen Klassen

6.1.1 Objektmodell

Objekthierarchie

Die folgende Abbildung zeigt den Ausschnitt aus dem Objektmodell von System.IO, der sich mit Dateien und Verzeichnissen beschäftigt.

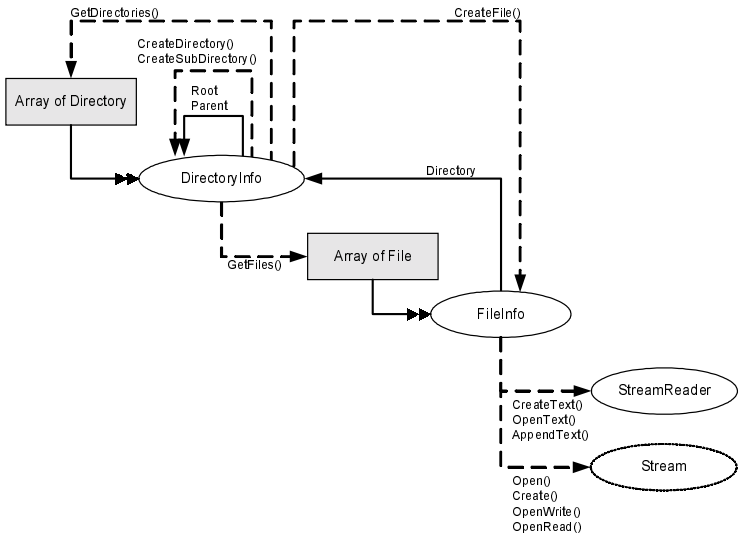


Abbildung 6.1: Objektmodell für System.IO.FileInfo und System.IO.DirectoryInfo

6.1.2 Verzeichnisse auflisten

Bei der Instanziierung der Klasse `DirectoryInfo` muss das zu kapselnde Verzeichnis als Zeichenkette an den Konstruktor übergeben werden. Ein parameterloser Konstruktor ist nicht definiert.

DirectoryInfo

Die Instanziierung ist auch dann erfolgreich, wenn das Verzeichnis gar nicht existiert. Mit dem Attribut `Exists` kann geprüft werden, ob das Verzeichnis auch wirklich vorhanden ist. Es folgt die Ausgabe von Informationen über den Ordner.

Exists

`GetFiles()` liefert eine Menge der in dem Ordner enthaltenen Dateien, wobei mit Hilfe eines DOS-Platzhalters ([*] und [?]) eine Einschränkung der Dateimenge möglich ist. Ergebnis dieser Methode ist ein Array von Objekten des Typs `FileInfo`. Über dieses Array kann genauso wie über eine Objektmenge mit `For Each` iteriert werden.

GetFiles()

Beispiel

Das folgende Beispiel gibt Informationen über einen Dateiordner aus und listet alle in dem Ordner enthaltenen Dateien mit der Dateierweiterung `.txt` auf.

**Dateiordner
ausgeben**

```
' =====
' Beispiel aus FCL-Buch.sln
' Liste der Dateien in einem bestimmten Ordner
' (C) Holger@Schwichtenberg.de
' =====
Sub Datei_Liste()
    Const VERZEICHNIS = "d:\data\dateisystem\"

    ' --- Zugriff auf ein Dateiverzeichnis
    Dim d As DirectoryInfo
    d = New DirectoryInfo(VERZEICHNIS)

    ' --- Prüfung auf Existenz
    If Not d.Exists Then
        out("Verzeichnis nicht vorhanden!")
        Exit Sub
    End If

    ' --- Ausgabe von Informationen über den Ordner
    out("Erzeugt am: " & d.CreationTime)
    out("Zuletzt gelesen am : " & d.LastAccessTime)
    out("Zuletzt geändert am : " & d.LastWriteTime)
    out("Wurzelordner: " & d.Root.Name)
    out("Name des übergeordneten Ordners: " & _
        d.Parent.Name)
```

```

out("Pfad des übergeordneten Ordners: " & _
    d.Parent.FullName)

' --- Liste bestimmter Dateien
Dim f As FileInfo
out("Alle Text-Dateien In Ordner: " & _
    d.FullName)
For Each f In d.GetFiles("*.txt")
    out(f.Name & ";" & f.Length() & ";" & _
        f.CreationTime())
Next

End Sub

```

Listing 6.1: Liste der Dateien in einem bestimmten Ordner [Dateisystem.vb]

6.1.3 Verzeichnisse und Dateien suchen

Die FCL bietet leider keine eingebaute Klasse, um Dateien oder Verzeichnisse zu suchen. Sie haben daher keine andere Möglichkeit, als selbst rekursiv in einem Verzeichnisbaum hinabzusteigen, um eine Datei oder ein Verzeichnis zu finden.

Ihnen wird die nachfolgend dokumentierte Klasse `FileSearcher` eine große Hilfe sein. `FileSearcher` sucht rekursiv von einem Startordner ausgehend nach einem oder mehreren Objekten im Dateisystem. Die Methode `Search()` erwartet im ersten Parameter einen String mit dem Startordner und im zweiten Parameter entweder einen String oder eine `ArrayList` mit mehreren Strings. Als Strings sind DOS-Suchmuster mit den Platzhaltern `?` und `*` erlaubt.

`FileSearcher` liefert die Suchergebnisse einerseits in Form einer `ArrayList` (Liste der vollständigen Pfade der Fundstellen) am Ende der Ausführung zurück. Andererseits feuert die Klasse bei jeder Fundstelle sofort das Ereignis `FileFound()`.

```

Imports System.IO

' =====
' Beispiel aus FCL-Buch.sln
' Ergebnis einer Suche im Dateisystem
' (C) Holger@Schwichtenberg.de
' =====

Public Class FileSearchResult
    Public name As String
    Public size As Long

    Sub New()

```

```

End Sub

Sub New(ByVal nName As String, _
    ByVal nSize As Long)
    name = nName
    size = nSize
End Sub

Public Overrides Function toString() As String
    toString = name & " (" & size & ")"
End Function

End Class

' =====
' Beispiel aus FCL-Buch.sln
' Klasse für Suchanfrage im Dateisystem
' (C) Holger@Schwichtenberg.de
' =====
Public Class FileSearcher

    ' --- Member
    Public Event FileFound( _
        ByVal sr As FileSearchResult)

    Public Function Search( _
        ByVal startdir As String, _
        ByVal name As String) As ArrayList

        Dim ergebnis As New ArrayList()
        Dim names As New ArrayList()
        names.Add(name)
        Search(startdir, names, ergebnis)
        Search = ergebnis
    End Function

    Public Function Search( _
        ByVal startdir As String, _
        ByVal names As ArrayList) As ArrayList

        Dim ergebnis As New ArrayList()
        _Search(startdir, names, ergebnis)
        Search = ergebnis
    End Function

    ' --- Interne Routinen
    Private Sub _Search( _
        ByVal startdir As String, _
        ByVal names As ArrayList, _
        ByVal ergebnis As ArrayList)

```

```

Dim d As New DirectoryInfo(startdir)
Dim f As FileInfo
Dim sd As DirectoryInfo
Dim sr As FileSearchResult
Dim name As String
'out("Suche In: " & startdir)

Try
    For Each name In names

        For Each f In d.GetFiles(name)
            sr = New FileSearchResult( _
                f.FullName, f.Length)
            RaiseEvent FileFound(sr)
            ergebnis.Add(sr)
        Next
        For Each sd In d.GetDirectories(name)
            sr = New FileSearchResult( _
                sd.FullName, 0)
            RaiseEvent FileFound(sr)
            ergebnis.Add(sr)
        Next
        Next
        ' --- Rekursion
        For Each sd In d.GetDirectories()
            _Search(sd.FullName, names, ergebnis)
        Next
    Catch
    End Try

End Sub

End Class

```

Listing 6.2: Klasse zur Suche im Dateisystem [FileSearch.vb]

Die folgende Routine sucht mit der Klasse `FileSearcher` innerhalb des Projekts »FCL-Buch« nach Dateien mit der Extension `.doc` und `.txt`.

```

Imports System.IO

' =====
' Beispiel aus FCL-Buch.sln
' Beispiel zur Anwendung der Klasse FileSearcher
' (C) Holger@Schwichtenberg.de
' =====

Module Dateisuche
    Public groesse As Long
    Public anz As Long
    Dim File_ergebnis As ArrayList

```



```

Dim Dir_ergebnis As ArrayList

Public Sub datei_suche()
    Dim s As New FileSearcher()

    Dim suchliste As ArrayList
    anz = 0
    groesse = 0
    AddHandler s.FileFound, _
        AddressOf datei_gefunden

    suchliste = New ArrayList()
    suchliste.Add("*.doc")
    suchliste.Add("*.txt")

    File_ergebnis = s.Search("..", suchliste)

    say("Suchergebnisse:" & File_ergebnis.Count)
    say("Größe: " & groesse)
End Sub

Sub datei_gefunden(ByVal sr As FileSearchResult)
    out("Datei gefunden:" & sr.name & "(" & _
        sr.size & ")")
End Sub

End Module

```

Listing 6.3: Beispiel zur Anwendung der Klasse FileSearcher [Dateisuche.vb]

Ausgabe

Die Suche sollte mindestens die folgenden beiden Dateien finden:

```

1 Datei gefunden:
H:\Code\FCL-Buch_Klassen\_Daten\Dateisystem\protokoll.txt (86)
2 Datei gefunden:
H:\Code\FCL-Buch_Klassen\_Daten\Dateisystem\ Buch.doc (10752)

```

6.1.4 Verzeichnisse anlegen und löschen

Die Klasse `DirectoryInfo` ermöglicht auch das Anlegen von Dateien. Dabei muss zunächst eine Instanz der `DirectoryInfo`-Klasse erzeugt werden, wobei dem Konstruktor der zu erzeugende Pfad übergeben wird. Danach ruft man einfach `Create()` auf. Diese Methode kann Verzeichnisse in mehreren Ebenen gleichzeitig erzeugen; das übergeordnete Verzeichnis muss nicht existieren. Die Methode ist auch tolerant gegenüber dem Fall, dass die Ver-

Create()

zeichnisse bereits existieren: Dann werden die bestehenden Verzeichnisse unangetastet beibehalten.

Delete() Die Methode `Delete()` auf einem `DirectoryInfo`-Objekt löscht das instanziierte Verzeichnis aus dem Dateisystem. `Delete()` erwartet einen optionalen booleschen Parameter, der spezifiziert, ob das Löschen rekursiv erfolgen soll. Wenn dieser Parameter nicht auf `True` gesetzt wird, führt eine Löschoperation zum Fehler, wenn das Verzeichnis Dateien oder Unterverzeichnisse enthält. Im Fall der Angabe von `True` werden auch alle Unterverzeichnisse rekursiv gelöscht.

Beispiel

Anlegen und Löschen von Verzeichnissen

Das Beispiel zeigt das Anlegen eines neuen Verzeichnisses mit zwei Unterverzeichnissen. Das Verzeichnis wird abschließend wieder gelöscht. Dabei wird allerdings – natürlich – nur ab der Ebene gelöscht, die der Pfad des aktuellen `DirectoryInfo`-Objekts festlegt. Durch `Create()` angelegte übergeordnete Ebenen werden nicht gelöscht, wie das folgende Beispiel beweist.

```
' =====
' Beispiel aus FCL-Buch.sln
' Verzeichnisse anlegen und löschen
' (C) Holger@Schwichtenberg.de
' =====

Sub datei_makedirl()

    Const VERZEICHNIS = _
        "d:\data\dateisystem\hs\Dokumente\"

    ' --- Zugriff auf ein Dateiverzeichnis
    Dim d As DirectoryInfo
    Dim sd As DirectoryInfo

    d = New DirectoryInfo(VERZEICHNIS)

    ' --- Verzeichnis erzeugen
    d.Create()

    ' --- Unterverzeichnis erzeugen
    sd = d.CreateSubdirectory("1. Halbjahr")
    out(sd.FullName & " angelegt!")

    ' --- Unterverzeichnis erzeugen
    sd = d.CreateSubdirectory("2. Halbjahr")
    out(sd.FullName & " angelegt!")
```

```

    MsgBox("Dateien angelegt!")

    ' --- Verzeichnis wieder löschen
    d.Delete(True)
End Sub

```

Listing 6.4: Verzeichnisse anlegen und löschen [Dateisystem.vb]

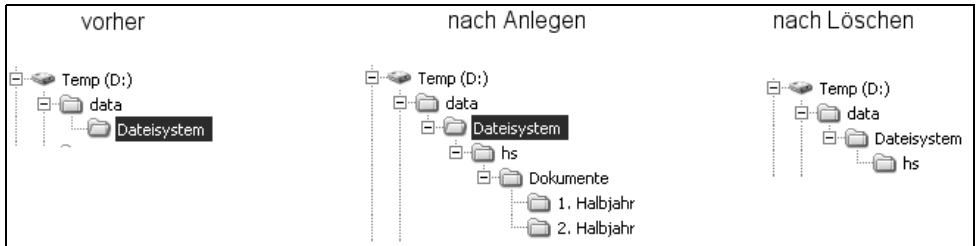


Abbildung 6.2: Dateisystemstruktur vor und nach der Ausführung des Beispiels

Variante

Eine Variante des obigen Beispiels besteht darin, die entsprechenden statischen Methoden aus der Klasse `Directory` zu nutzen. `CreateDirectory("PFAD")` legt einen Ordner an, während `Directory.Delete("PFAD", True)` den Inhalt des angegebenen Ordners rekursiv löscht.

CreateDirectory()

```

' =====
' Beispiel aus FCL-Buch.sln
' Verzeichnis anlegen (VARIANTE)
' (C) Holger@Schwichtenberg.de
' =====
Sub datei_makedir2()
    Const VERZEICHNIS = _
        "d:\data\dateisystem\hs\Dokumente\"

    ' --- Verzeichnis anlegen
    Directory.CreateDirectory(VERZEICHNIS)
    MsgBox("Ordner angelegt!")
    ' --- Verzeichnis wieder löschen
    Directory.Delete(VERZEICHNIS, True)
End Sub

```

Listing 6.5: Verzeichnis anlegen (VARIANTE) [Dateisystem.vb]

6.1.5 Dateien anlegen und löschen

CreateFile(), CreateDirectory()

Eine Datei wird mit `CreateFile()` auf einem `DirectoryInfo`-Objekt erzeugt. Analog gibt es ein `CreateDirectory()`. Ein Eintrag wird über die Methode `Delete()` entfernt, die es sowohl im `Directory`- als auch im `File`-Objekt gibt.

Auch für ein `FileInfo`-Objekt gilt analog zur Klasse `DirectoryInfo`:

- ▶ Bei der Instanziierung muss der Pfad angegeben werden.
- ▶ Die Instanziierung ist auch dann erfolgreich, wenn die Datei gar nicht existiert.
- ▶ Mit dem Attribut `Exists` kann geprüft werden, ob die Datei vorhanden ist.

Beispiel

Dateien erzeugen

Im folgenden Beispiel werden zehn leere Textdateien erzeugt. Wenn es die Dateien in dem angegebenen Verzeichnis schon gibt, werden sie vor dem Anlegen gelöscht.

```
' =====
' Beispiel aus FCL-Buch.sln
' Erzeugen und Löschen von Textdateien
' (C) Holger@Schwichtenberg.de
' =====

Sub dateien_erzeugen()

    Dim d As DirectoryInfo
    Dim a As Integer
    Dim dateiname As String
    Dim f As FileInfo

    Const VERZEICHNIS = "d:\data\dateisystem\"

    ' --- Zugriff auf ein Dateiverzeichnis
    d = New DirectoryInfo(VERZEICHNIS)

    ' --- 10 neue Dateien erzeugen
    For a = 1 To 10
        dateiname = "Datei" & a.ToString() & ".txt"
        f = New FileInfo(d.FullName & "\" & dateiname)
        If f.Exists Then
            ' --- Wenn Datei vorhanden, dann löschen!
            f.Delete()
            out("Datei " & dateiname & " gelöscht!")
        End If
        f.Create()
    Next a
End Sub
```

```

        out("Datei " & f.Name & " erzeugt!")
    Next
End Sub

```

Listing 6.6: Erzeugen und Löschen von Textdateien [Dateisystem.vb]

6.1.6 Kopieren und Verschieben

Die Klassen `Directory` und `File` bieten beide die Methode `Move()` an, mit zwei Parametern: im ersten der Pfad des Ausgangsobjekts, im zweiten der Pfad des Zielobjekts. Die Klassen `DirectoryInfo` und `FileInfo` bieten die gleiche Funktionalität mit der Methode `MoveTo()`. Hier wird nur ein Parameter erwartet: der Zielpfad. Das Ausgangsobjekt ist bereits durch die aktuelle Bindung der Klassen `DirectoryInfo` bzw. `FileInfo` festgelegt.

Verschieben

Das Kopieren ist leider nur auf Dateiebene definiert durch `Copy()` in `File` bzw. `CopyTo()` in `FileInfo`. Es gibt kurioserweise keine Möglichkeit in der FCL, ein ganzes Verzeichnis zu kopieren. Sie müssen dies selbst durch rekursives Kopieren der Dateien erledigen.

Kopieren

Beispiel

Im folgenden Beispiel wird ein Ordner angelegt, eine Datei dorthin verschoben, die Ursprungsdatei dann gelöscht und die Kopie an den Ursprungsordner verschoben. Damit die »Operation« rückstands-frei bleibt, wird der angelegte Ordner am Ende wieder gelöscht.

```

' =====
' Beispiel aus FCL-Buch.sln
' Datei kopieren und verschieben
' (C) Holger@Schwichtenberg.de
' =====
Sub datei_bewegen()
    Const DOCORDNER = _
        "_daten\dateisystem\Dokumente\"

    If Directory.Exists(DOCORDNER) Then
        Directory.Delete(DOCORDNER, True)
    End If
    Directory.CreateDirectory(DOCORDNER)
    out("Ordner angelegt!")
    File.Copy("../_daten\dateisystem\buch.doc", _
        DOCORDNER & "\buch.doc")
    out("Datei kopiert!")
    File.Delete("../_daten\dateisystem\buch.doc")
    out("Ausgangsdatei gelöscht")
    File.Move(DOCORDNER & "\buch.doc", _

```

```
        "..\_daten\dateisystem\buch.doc")
    out("Kopie verschoben")
    Directory.Delete(DOCDORNER, True)
    out("Verzeichnis gelöscht")
End Sub
```

Listing 6.7: Kopieren und Verschieben einer Datei [Dateisystem.vb]

6.1.7 Informationen über das Dateisystem

System.IO.Path Die Klasse `System.IO.Path` bietet einige statische Attribute, die Informationen über die Trennzeichen enthalten, die auf der jeweiligen Plattform dazu dienen, Dateisystempfade aufzubauen. Diese Funktionalität ist konzipiert in Hinblick auf die Plattformunabhängigkeit von .NET, da nicht alle Betriebssysteme die gleichen Trennzeichen für Dateisystempfade verwenden.

Attribut	Bedeutung
DirectorySeparatorChar	DirectorySeparatorChar ist das Zeichen, das die Namen von Verzeichnissen und Dateien voneinander trennt (liefert unter Windows den Backslash »\«).
AltDirectorySeparatorChar	Alternatives Pfadtrennzeichen (liefert unter Windows den Slash "/")
PathSeparator	PathSeparator ist das Zeichen, das zwei Pfadangaben voneinander trennt (z.B. in der Umgebungsvariablen <i>Path</i>). Dies ist unter Windows das Semikolon.
VolumeSeparatorChar	VolumeSeparatorChar ist das Zeichen, das einem Laufwerksbuchstaben folgen muss (unter Windows der Doppelpunkt).
InvalidPathChars	InvalidPathChars liefert ein Array mit Zeichen, die in Pfadangaben nicht erlaubt sind.
NewLine	Informationen über das Trennzeichen, das auf der jeweiligen Plattform dazu dienen, Zeilenumbrüche in Zeichenketten zu definieren

Tabelle 6.2: Attribute, die allgemeine Informationen über den Aufbau von Dateisystempfaden liefern

Beispiel

Informationen
über den Aufbau
von Pfaden

Das folgende Beispiel gibt Informationen über den Aufbau von Dateisystempfaden aus. Die verbotenen Zeichen werden dabei als

eine Liste von ASCII-Codes ausgegeben, da nicht alle dieser Zeichen dargestellt werden können.

```
' =====
' Beispiel aus FCL-Buch.sln
' Ausgabe von Informationen über den Aufbau von
' Dateisystempfaden
' (C) Holger@Schwichtenberg.de
' =====
Sub datei_pfade()
    Dim p As System.IO.Path
    out("-- Allgemeine Informationen")
    out("Ordertrenner: " & _
        p.DirectorySeparatorChar)
    out("Alternativer Ordertrenner: " & _
        p.AltDirectorySeparatorChar)
    out("Laufwerkstrenner: " & _
        p.VolumeSeparatorChar)
    out("Pfadtrenner: " & p.PathSeparator)
    out("Verbotene Zeichen:")
    Dim s As String
    Dim ausgabe As String
    For Each s In p.InvalidPathChars
        ausgabe &= Asc(s) & " "
    Next
    out(ausgabe)
End Sub
```

Listing 6.8: Ausgabe von Informationen über den Aufbau von Dateisystempfaden [Infos.vb]

Ausgabe

Das obige Listing liefert unter Windows folgende Ausgabe:

```
1 -- Allgemeine Informationen
2 Ordertrenner: \
3 Alternativer Ordertrenner: /
4 Laufwerkstrenner: :
5 Pfadtrenner: ;
6 Verbotene Zeichen:
7 34 60 62 124 0 8 16 17 18 20 21 22 23 24 25
```

6.1.8 Arbeit mit Pfadangaben

Die Klasse `System.IO.Path` erfüllt neben der Ausgabe von allgemeinen Informationen über den Aufbau von Pfadangaben zwei weitere Aufgaben:

- Extraktion von Teilinformationen aus Pfadangaben
- Manipulation von Pfadangaben

Beispiel

Pfadangaben extrahieren und ändern

Das Beispiel zeigt zunächst die Extraktion von einzelnen Teilen einer Pfadangabe. Anschließend wird die Dateiextension ausgetauscht und dann ein neues Unterverzeichnis in die Pfadangabe eingefügt.

```
' =====
' Beispiel aus FCL-Buch.sln
' Arbeit mit Pfadangaben
' (C) Holger@Schwichtenberg.de
' =====
Sub datei_pfad2()

    Const PFAD = _
"d:\data\dateisystem\hs\Dokumente\dotnet-buch.doc"

    Dim p As System.IO.Path

    ' --- Infos ausgeben
    out("Kompletter Pfad:" & p.GetFullPath(PFAD))
    out("Dateiname: " & p.GetFileName(PFAD))
    out("Extension: " & p.GetExtension(PFAD))
    out("Verzeichnis: " & p.GetDirectoryName(PFAD))
    out("Laufwerk: " & p.GetPathRoot(PFAD))
    out("Dateiname ohne Extension: " & _
        p.GetFileNameWithoutExtension(PFAD))

    ' --- Änderungen
    out("Datei mit neuer Extension: " & _
        p.ChangeExtension(PFAD, ".rtf"))

    Dim verzeichnis As String
    Dim datei As String
    verzeichnis = p.GetDirectoryName(PFAD)
    datei = p.GetFileName(PFAD)
    verzeichnis = p.Combine(PFAD, "Neu")
    out("Neuer Pfad: " & p.Combine(verzeichnis, _
        datei))

End Sub
```

Listing 6.9: Arbeit mit Pfadangaben [Infos.vb]

Ausgabe

Das obige Listing liefert folgende Ausgabe:

```
1 Kompletter Pfad:d:\data\dateisystem\hs\Dokumente
  \dotnet-buch.doc
2 Dateiname: dotnet-buch.doc
3 Extension: .doc
4 Verzeichnis: d:\data\dateisystem\hs\Dokumente
5 Laufwerk: d:\
6 Dateiname ohne Extension: dotnet-buch
7 Datei mit neuer Extension: d:\data\dateisystem
  \hs\Dokumente\dotnet-buch.rtf
8 Neuer Pfad: d:\data\dateisystem\hs\Dokumente\
  dotnet-buch.doc\Neu\dotnet-buch.doc
```

6.2 Zugriff auf Dateiinhalte

Auch die Arbeit mit (Text-)Dateien hat sich stark verändert. Eine große Verbesserung gegenüber FSO ist, dass nun auch Binärdateien gelesen und geschrieben werden können.

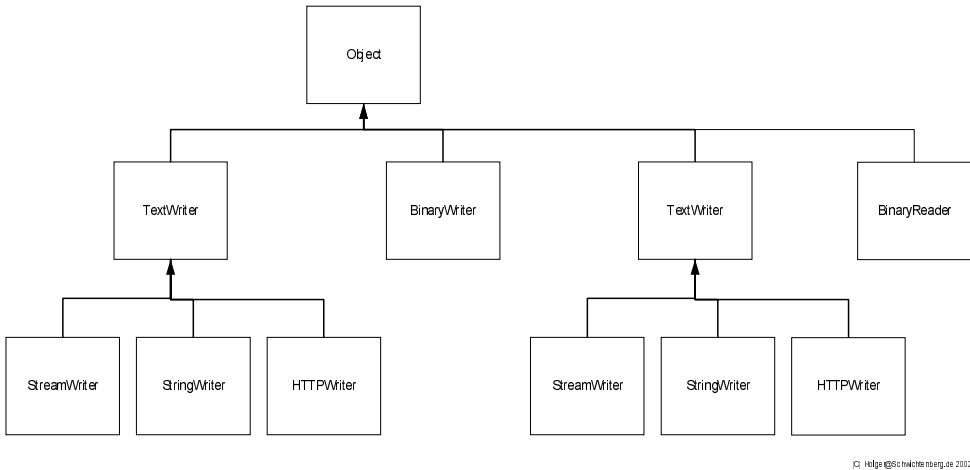
6.2.1 Readers, Writers und Streams

Die Verwendung von *Streams* (Datenströmen) ist ein zentrales Konzept in .NET. Ein *Stream* ist eine beliebige Byte-Folge in einem beliebigen Medium, z.B. eine Textdatei, eine Binärdatei, ein Datenstrom im Netzwerk oder auch eine Byte-Folge im Speicher. Es gibt für Streams in der FCL eine abstrakte Basisklasse `System.IO.Stream`, von der zahlreiche Klassen in verschiedenen Namespaces erben. Es gibt zahlreiche Attribute und Methoden in FCL-Klassen, die mit Streams arbeiten.

Streams

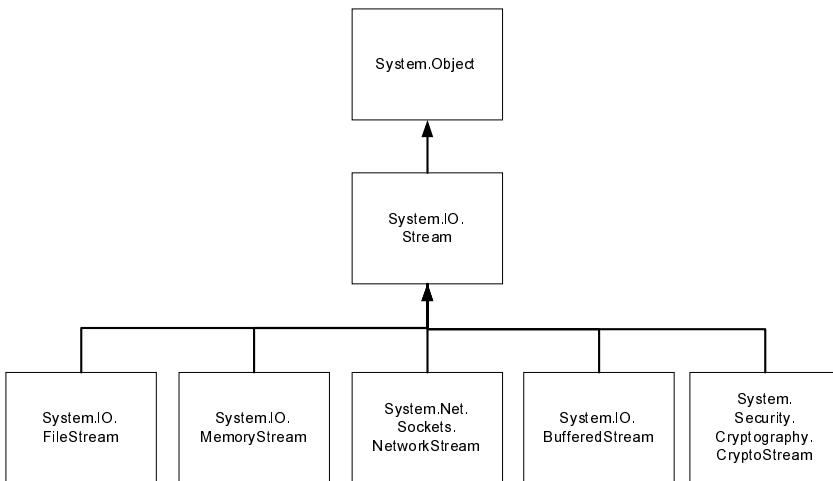
Auf einen Stream greift man mit Hilfe anderer Klassen zu: `StreamReader` und `StreamWriter` für Streams mit ASCII-Zeichen und `BinaryReader` und `BinaryWriter` für Streams mit beliebigen Byte-Folgen. `StreamReader` und `StreamWriter` sind Unterklassen der Klassen `TextReader` bzw. `TextWriter`. `BinaryReader` und `BinaryWriter` sind dagegen direkt von `System.Object` abgeleitet.

Readers und Writers



(C) Holger@Schichtenberg.de 2002

Abbildung 6.3: Vererbung der Readers und Writers



(C) Holger@Schichtenberg.de 2002

Abbildung 6.4: Vererbung der Streams



Während ein `FileInfo`-Objekt das Äußere einer Datei darstellt, ist ein `FileStream`-Objekt der Inhalt einer Datei.

6.2.2 Textdatei schreiben

Das grundsätzliche Vorgehen beim Schreiben in eine Datei ist:

- ▶ Erzeugen eines `FileStream`-Objekts unter Angabe der Parameter `FileMode` und `FileAccess`
- ▶ Erzeugen eines `StreamWriter`-Objekts auf Basis des `FileStream`-Objekts
- ▶ Setzen des Zeigers auf Anfang oder Ende des Streams
- ▶ Danach können Inhalte mit `Write()` und `WriteLine()` geschrieben werden.
- ▶ Aufruf von `Close()` im `StreamWriter`-Objekt
- ▶ Aufruf von `Close()` im `FileStream`-Objekt

**Vorgehensweise
beim Schreiben
einer Textdatei**

Ohne Aufruf von `Close()` im `StreamWriter` wird nichts in die Datei geschrieben. `Close()` darf für den `FileStream` erst nach dem Schließen des `StreamWriter`-Objekts aufgerufen werden.



Für `FileAccess` und `FileMode` gibt es Enumerationen in `System.IO` (siehe folgende Tabellen).

Symbolische Konstante	Wenn Datei vorhanden	Wenn Datei nicht vorhanden
Append	Anhängen	Wird erzeugt
Create	Wird überschrieben	Wird erzeugt
CreateNew	Fehler	Wird erzeugt
Open	Öffnen	Fehler
OpenOrCreate	Öffnen	Wird erzeugt
Truncate	Öffnen + Löschen	Fehler

Tabelle 6.3: Enumeration »FileMode«

Symbolische Konstante	Lesen	Schreiben
Read	Ja	Nein
ReadWrite	Ja	Ja
Write	Nein	Ja

Tabelle 6.4: Enumeration »FileMode«

Beispiel

Protokolldatei erzeugen

Die folgende Routine hängt eine Protokollierungsinformation an eine Textdatei an.

```
' =====
' Beispiel aus FCL-Buch.sln
' Schreiben einer Protokolldatei
' (C) Holger@Schwichtenberg.de
' =====

Public Sub datei_schreiben()
    Const DATEINAME As String = _
        "d:\buch\dateisystem\protokoll.txt"

    ' --- Datei öffnen
    Dim fs As FileStream = New FileStream( _
        DATEINAME, FileMode.OpenOrCreate, _
        FileAccess.Write)

    ' --- Stream öffnen
    Dim w As StreamWriter = New StreamWriter(fs)
    ' --- Anfügen am Ende
    w.BaseStream.Seek(0, SeekOrigin.End)
    ' --- Zeilen schreiben
    w.WriteLine("Start des Programms: " & now)
    ' --- Einzelne Zeichen schreiben
    Dim a As Integer
    For a = 1 To 26
        w.Write(chr(96 + a))
    Next
    w.WriteLine()
    ' --- Zeilen schreiben
    w.WriteLine("Ende des Programms: " & now)
    ' --- Writer und Stream schließen
    w.Close()
    fs.Close()
End Sub
```

Listing 6.10: Schreiben einer Protokolldatei [Dateiinhalte.vb]

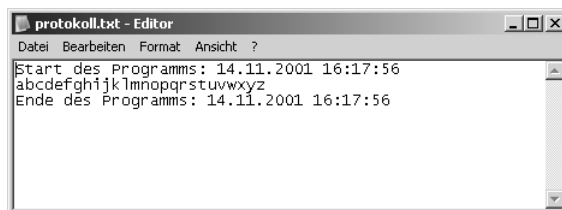


Abbildung 6.5: Ergebnis des obigen Beispiels

6.2.3 Textdatei lesen

Das grundsätzliche Vorgehen beim Lesezugriff auf den Inhalt einer Textdatei ist:

**Vorgehensweise
beim Lesen einer
Textdatei**

- ▶ Erzeugen eines `FileStream`-Objekts
- ▶ Erzeugen eines `StreamReader`-Objekts auf Basis des `FileStream`-Objekts
- ▶ Setzen des Zeigers auf den Anfang des Streams
- ▶ Danach können Inhalte mit `Read()` und `ReadLine()` gelesen werden.
- ▶ Während des Lesens kann mit `while r.Peek() > -1` auf das Erreichen des Dateiendes geprüft werden.
- ▶ Aufruf von `Close()` im `StreamReader`-Objekt
- ▶ Aufruf von `Close()` im `FileStream`-Objekt

Beispiel

In diesem Beispiel wird die im vorherigen Unterkapitel erzeugte Protokolldatei wieder eingelesen und ausgegeben.

**Protokolldatei
einlesen**

```
' =====
' Beispiel aus FCL-Buch.sln
' Zeilenweises Lesen aus einer Textdatei
' (C) Holger@Schwichtenberg.de
' =====
Sub Datei_lesen()
    Const DATEINAME As String = _
        "d:\buch\dateisystem\protokoll.txt"
    ' --- Datei öffnen
    Dim fs As FileStream = New FileStream( _
        DATEINAME, FileMode.OpenOrCreate, _
        FileAccess.ReadWrite)
    ' --- Stream öffnen
    Dim r As StreamReader = New StreamReader(fs)
    ' --- Zeiger auf den Anfang
    r.BaseStream.Seek(0, SeekOrigin.Begin)
    ' --- Alle Zeilen lesen und an Console ausgeben
    While r.Peek() > -1
        say(r.ReadLine())
    End While
    ' --- Reader und Stream schließen
    r.Close()
    fs.Close()
End Sub
```

Listing 6.11: Zeilenweises Lesen aus einer Textdatei [Dateiinhalte.vb]

Ausgabe

Die obige Routine liefert für die in Abbildung 1.5 dargestellte Datei folgende Ausgabe:

```
1 Start des Programms: 14.11.2001 16:21:45
2 abcdefghijklmnopqrstuvwxyz
3 Ende des Programms: 14.11.2001 16:21:45
```

6.2.4 Binärdatei schreiben

Das Schreiben in eine Binärdatei unterscheidet sich von dem Schreiben in eine Textdatei nur dadurch, dass ein `BinaryWriter`-Objekt statt eines `StreamWriter`-Objekts verwendet wird. Die Klasse `BinaryWriter` bietet zum Schreiben nur `Write()`, nicht aber `WriteLine()` an.

Beispiel

Byte-Werte in Datei schreiben

Das Beispiel schreibt die Byte-Werte 1 bis 255 nacheinander in eine Datei mit Namen *daten.bin*.

```
' =====
' Beispiel aus FCL-Buch.sln
' Schreiben einer Binärdatei
' (C) Holger@Schwichtenberg.de
' =====
Public Sub datei_binaer_schreiben()
    Const DATEINAME = _
        "d:\data\dateisystem\daten.bin"
    ' --- Datei öffnen
    Dim fs As FileStream = New FileStream( _
        DATEINAME, FileMode.OpenOrCreate, _
        FileAccess.Write)
    ' --- Stream öffnen
    Dim w As BinaryWriter = New BinaryWriter(fs)
    ' --- Anfügen am Ende
    w.BaseStream.Seek(0, SeekOrigin.End)
    ' --- Einzelne Zeichen schreiben
    Dim a As Integer
    For a = 1 To 255
        w.Write(a)
    Next
    ' --- Stream schließen
    w.Close()
    fs.Close()
End Sub
```

Listing 6.12: Schreiben einer Binärdatei [Dateiinhalte.vb]

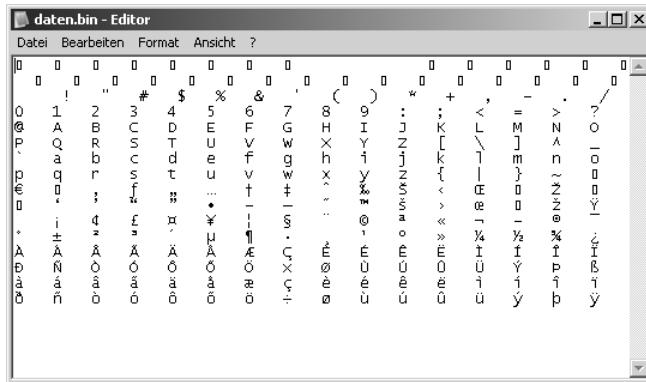


Abbildung 6.6: Ergebnis des obigen Beispiels, dargestellt im Windows Editor

6.2.5 Binärdatei lesen

Auch das binäre Lesen einer Datei entspricht dem Lesen einer Textdatei mit dem Unterschied, dass ein `BinaryReader` statt eines `StreamReader` verwendet werden muss.

Beispiel

Das Beispiel liest die bereits zuvor verwendete Textdatei in binärer Form ein.

Datei binär lesen

```
' =====
' Beispiel aus FCL-Buch.sln
' Zeichenweises binäres Lesen aus einer Datei
' (C) Holger@Schwichtenberg.de
' =====

Sub Datei_binaer_lesen()
    Const DATEINAME As String = _
        "d:\buch\dateisystem\protokoll.txt"
    ' --- Datei öffnen
    Dim fs As New FileStream( _
        DATEINAME, FileMode.OpenOrCreate, _
        FileAccess.ReadWrite)
    ' --- Stream öffnen
    Dim r As BinaryReader = New BinaryReader(fs)
    ' --- Zeiger auf den Anfang
    r.BaseStream.Seek(0, SeekOrigin.Begin)
    ' --- Alle Zeilen lesen und an Console ausgeben
    Dim ausgabe As String
    While r.PeekChar() > -1
        ausgabe &= r.ReadByte()
    End While
    say(ausgabe)
```

```
r.Close()
```

```
End Sub
```

Listing 6.13: Zeichenweises binäres Lesen aus einer Datei [Dateiinhalte.vb]

Ausgabe

Die obige Routine liefert für die in Abbildung 1.5 dargestellte Datei folgende Ausgabe:

```
83,116,97,114,116,32,100,101,115,32,80,114,111,103,114,97,109,
109,115,58,32,49,52,46,49,49,46,50,48,48,49,32,49,54,58,50,49,
58,52,53,13,10,97,98,99,100,101,102,103,104,105,106,107,108,10
9,110,111,112,113,114,115,116,117,118,119,120,121,122,13,10,69
,110,100,101,32,100,101,115,32,80,114,111,103,114,97,109,109,1
15,58,32,49,52,46,49,49,46,50,48,48,49,32,49,54,58,50,49,58,52
,53,13,10
```

6.3 Dateisystem überwachen

Änderungen im Dateisystem verfolgen

Ein sehr schönes Feature, das es in FSO nicht gab, ist die Überwachung des Dateisystems auf Veränderungen. Überwacht werden können Dateien und Verzeichnisse. Überwachte Veränderungen sind

- ▶ Anlegen
- ▶ Löschen
- ▶ Umbenennen
- ▶ Änderung der Dateiattribute
- ▶ Änderung der Datumsangaben (Erstellung, Lesezugriff, Schreibzugriff)
- ▶ Änderung der Sicherheitseinstellungen (nur unter NTFS)

Unterstützte Plattformen

Diese Dateisystemüberwachung ist nur mit Windows NT 4.0 und höher verfügbar. Die Überwachung funktioniert unter bestimmten Konstellationen auch für entfernte Laufwerke auf anderen Computern (siehe Tabelle). Die Überwachung ist natürlich nicht verfügbar für CD- und DVD-Laufwerke.

Client	Festplattentyp	Überwachung möglich?
Windows 2000/XP/.NET	Lokale Festplatten	Ja
Windows NT 4.0	Lokale Festplatten	Ja
Windows 2000/XP/.NET	Entfernte Festplatten unter Windows 2000/XP/.NET	Ja
Windows 2000/XP/.NET	Entfernte Festplatten unter Windows NT 4.0	Ja
Windows NT 4.0	Entfernte Festplatten unter Windows NT 4.0	Nein
Windows NT 4.0	Entfernte Festplatten unter Windows 2000/XP/.NET	Ja
Windows 9x/ME	Beliebig	Nein

Tabelle 6.5: Unterstützte Plattformen für die Dateisystemüberwachung

Klasse `FileSystemWatcher`

Die Überwachung wird realisiert durch eine einzige Klasse: `System.IO.FileSystemWatcher`. Bei der Instanziierung sind keine Parameter notwendig. Optional kann aber im ersten Parameter das zu überwachende Verzeichnis und im zweiten Parameter ein DOS-Muster für die zu überwachenden Dateien eingegeben werden.

**FileSystem-
Watcher**

Die Klasse `FileSystemWatcher` definiert vier Ereignisse, die beim Eintritt von Änderungen im Dateisystem ausgelöst werden können. Diese Ereignisse sind:

- `Changed()`
- `Created()`
- `Deleted()`
- `Renamed()`

Die Klasse bietet einige Einstellungsmöglichkeiten, um genau zu spezifizieren, was überwacht werden soll.

Eine einzelne Änderung im Dateisystem kann mehrere Ereignisse im `FileSystemWatcher` auslösen, z.B. führt das Ändern des Dateinamens nacheinander zu den Ereignissen `Renamed` und `Changed`. Das Verschieben einer Datei löst insgesamt fünf verschiedene Ereignisse auf der Datei und dem Zielordner aus.



Attribute

- Path

Die Klasse `FileSystemWatcher` erlaubt im Attribut `Path` die Spezifizierung, welcher Pfad überwacht werden soll.

```
watcher.Path = PFAD
```
- Filter

Im Attribut `Filter` kann durch ein DOS-Muster (Verwendung von `*` und `?` möglich) angegeben werden, welche Dateien überwacht werden sollen.

```
watcher.Filter = "*.*)"
```
- IncludeSubdirectories

Mit `IncludeSubdirectories` kann festgelegt werden, dass die Überwachung auch die Unterverzeichnisse des genannten Verzeichnisses mit einschließt.

```
watcher.IncludeSubdirectories = True
```
- NotifyFilter

Für das `Changed`-Ereignis kann näher spezifiziert werden, welche Änderungen zu beachten sind. Die folgende Tabelle zeigt die Möglichkeiten, die über die Enumeration `NotifyFilters` als Konstanten zur Zuweisung an das Attribut `NotifyFilter` zur Verfügung stehen.

Konstante	Erläuterung
<code>Attributes</code>	Änderung der Datei-/Verzeichnis-Attribute
<code>CreationTime</code>	Änderung des Erstellungsdatums
<code>DirectoryName</code>	Änderung des Verzeichnisnamens
<code>FileName</code>	Änderung des Dateinamens
<code>LastAccess</code>	Änderung des Datums des letzten Zugriffs
<code>LastWrite</code>	Änderung des Datums des letzten Schreibzugriffs
<code>Security</code>	Änderung der Sicherheitseinstellungen
<code>Size</code>	Änderung der Dateigröße

Tabelle 6.6: Überwachbare Veränderungen in einem Dateisystemobjekt

Bindung der Ereignisse

- AddHandler

Mit Hilfe des VB.NET-Befehls `AddHandler` müssen dann noch die Ereignisbehandlungsroutinen mit den Ereignissen verbunden werden. Es kann pro Ereignis eine Ereignisbehandlungsroutine definiert werden. Da `Changed`, `Created` und `Deleted` die gleichen Ereignis-Parameter besitzen, können diese drei Ereignisse auch in einer Ereignisbehandlungsroutine bearbeitet werden.

```
Ereignisbehandlung(ByVal source As Object, _
ByVal e As FileSystemEventArgs)
```

Im ersten Parameter wird ein Verweis auf das `FileSystemWatcher`-Objekt, das Auslöser war, geliefert. Der zweite Parameter, ein Objekt vom Typ `FileSystemEventArgs`, liefert den Pfad und den Namen der Datei bzw. des Verzeichnisses. Durch das Attribut `ChangeType` aus dem übergebenen `FileSystemEventArgs`-Objekt kann dann zwischen den drei Ereignistypen unterschieden werden.

Beim `Renamed`-Ereignis ist der zweite Parameter dagegen ein anderer. In der Klasse `RenamedEventArgs` gibt es zusätzlich die Attribute `OldName` und `OldFullPath`.

Renamed

```
Renamed(ByVal source As Object, _
ByVal e As RenamedEventArgs)
```

Bei der Verwendung von spätem Binden können auch alle vier Ereignisse in einer Ereignisbehandlungsroutine behandelt werden. Dazu ist der zweite Parameter vom Typ `System.EventArgs` zu spezifizieren. Dies ist die gemeinsame Oberklasse über `FileSystemEventArgs` und `RenamedEventArgs`.

System.EventArgs

Beim `Changed`-Ereignis wird leider nicht übermittelt, welche Änderung ausgeführt wurde. Wenn Sie auf unterschiedliche Veränderungen anders reagieren wollen, müssen Sie dafür jeweils ein eigenes `FileSystemWatcher`-Objekt anlegen.



Start der Überwachung

Die Überwachung wird gestartet mit

```
watcher.EnableRaisingEvents = True
```

**EnableRaising
Events**

Beispiel

Im folgenden Beispiel werden alle Ereignisse für beliebige Dateien in einem bestimmten Verzeichnis (und in dessen Unterverzeichnissen) überwacht und per Ausgabe dokumentiert.

**Verzeichnis
überwachen**

```
' =====
' Beispiel aus FCL-Buch.sln
' Überwachung der Veränderungen In einem Verzeichnis
' (C) Holger@Schwichtenberg.de
' =====
Imports System.IO

Module Dateisystem_Ueberwachung
```

```

Public Sub Dateisystem_ueberwachen()

    Const PFAD As String = "D:\data\Dateisystem"

    ' --- Instanziierung
    Dim watcher As New FileSystemWatcher()

    ' --- zu überwachendes Verzeichnis
    watcher.Path = PFAD
    ' --- zu überwachende Dateitypen
    watcher.Filter = "*.*)"
    ' --- Unterverzeichnisse einschließen
    watcher.IncludeSubdirectories = True

    ' --- zu überwachende Änderungen
    watcher.NotifyFilter = _
        NotifyFilters.Attributes Or _
        NotifyFilters.LastAccess Or _
        NotifyFilters.LastWrite Or _
        NotifyFilters.Security Or _
        NotifyFilters.Size Or _
        NotifyFilters.FileName Or _
        NotifyFilters.DirectoryName

    ' --- Registrierung der
    ' --- Ereignisbehandlungsroutinen
    AddHandler watcher.Changed, _
        AddressOf DS_aenderung
    AddHandler watcher.Created, AddressOf DS_neu
    AddHandler watcher.Deleted, _
        AddressOf DS_loeschen
    AddHandler watcher.Renamed, AddressOf DS_name
    ' --- Starten der Überwachung
    out("Überwachen von " & PFAD & " beginnt...")
    watcher.EnableRaisingEvents = True
    ' --- Warten auf Benutzeraktion
    MsgBox( _
        "Klicken Sie hier, um die Überwachung " & _
        "zu beenden!", , "Überwachung")

End Sub

' ### Ereignisbehandlungsroutine für
' ### Änderungsereignisse
Public Sub DS_aenderung( _
    ByVal source As Object, _
    ByVal e As FileSystemEventArgs)

    out("Datei " & e.FullPath & _
        " wurde geändert!")

End Sub

```

```

' ### Ereignisbehandlungsroutine für
' ### Erstellungsereignisse
Public Sub DS_neu(ByVal source As Object, _
ByVal e As FileSystemEventArgs)

    out("Datei " & e.FullPath & _
        " wurde neu angelegt!")
End Sub

' ### Ereignisbehandlungsroutine für
' ### Löschereignisse
Public Sub DS_loeschen(ByVal source As Object, _
ByVal e As FileSystemEventArgs)

    out("Datei " & e.FullPath & _
        " wurde gelöscht!")
End Sub

' ### Ereignisbehandlungsroutine für
' ### Namensänderungen im Dateisystem
Public Sub DS_name(ByVal source As Object, _
ByVal e As RenamedEventArgs)

    out("Datei " & e.OldFullPath & _
        " heißt nun " & e.FullPath & "!")
End Sub

End Module

```

Listing 6.14: Überwachung der Veränderungen in einem Verzeichnis [watcher.vb]

Ausgabe

Das folgende Listing zeigt ein Beispiel für eine Ausgabe und die zugehörigen Aktionen im Windows Explorer. **Ausgabe**

Ausgeführte Aktion	Ausgabe
Anlegen eines Ordners	1 Überwachen von D:\data\Dateisystem beginnt... 2 D:\data\Dateisystem\neuer Ordner wurde neu angelegt!
Umbenennen des Ordners	3 neuer Ordner heißt nun hs!
Anlegen einer Textdatei mit dem Kontextmenü des Explorers	4 D:\data\Dateisystem\neu textdokument.txt wurde neu angelegt!

Tabelle 6.7: Beispiel für eine Ausgabe des Dateisystemüberwachungsprogramms

Ausgeführte Aktion	Ausgabe
Umbenennen der Datei	5 neu textdokument.txt heißt nun notizen.txt!
	6 D:\data\Dateisystem\notizen.txt wurde geändert!
Inhalt der Textdatei ändern	7 D:\data\Dateisystem\notizen.txt wurde geändert!
	8 D:\data\Dateisystem\notizen.txt wurde geändert!
Anlegen einer Word-Datei mit dem Kontextmenü des Explorers	9 D:\data\Dateisystem\neu microsoft word-dokument.doc wurde neu angelegt!
	10 D:\data\Dateisystem\neu microsoft word-dokument.doc wurde geändert!
Umbenennen der Word-Datei	11 neu microsoft word-dokument.doc heißt nun buch.doc!
Verschieben der Word-Datei in ein Unterverzeichnis	12 D:\data\Dateisystem\buch.doc wurde geändert!
	13 D:\data\Dateisystem\buch.doc wurde gelöscht!
	14 D:\data\Dateisystem\hs\buch.doc wurde neu angelegt!
	15 D:\data\Dateisystem\hs wurde geändert!
	16 D:\data\Dateisystem\hs\buch.doc wurde geändert!
Schreibschutz auf Datei setzen	17 D:\data\Dateisystem\notizen.txt wurde geändert!
Datei mit Schreibschutz nach Anfrage gelöscht	18 D:\data\Dateisystem\notizen.txt wurde geändert!
	19 D:\data\Dateisystem\notizen.txt wurde geändert!
	20 D:\data\Dateisystem\notizen.txt wurde gelöscht!

Tabelle 6.7: Beispiel für eine Ausgabe des Dateisystemüberwachungsprogramms (Fortsetzung)

7 Namespaces System.Text und System.Text.RegularExpressions

Die Arbeit mit Zeichenketten, oder vielmehr das Parsen eingegebener Zeichenketten, ist eine Aufgabe, die in vielen Programmiersprachen einen verhältnismäßig hohen Aufwand erfordert. Das .NET-Framework bietet hierfür verschiedene Klassen im Namespace `System.Text` und in dem ihm untergeordneten Namespace `System.Text.RegularExpressions` an. Wie man schon am Namen sieht, geht es darin um reguläre Ausdrücke, eine effektive Form des Durchsuchens von Zeichenketten mittels so genannter *Patterns*.

Der Namespace `System.Text` selbst enthält nur eine interessante Klasse, nämlich `StringBuilder`.

7.1 Die Klasse `StringBuilder`

`StringBuilder` ist eine versiegelte Klasse (sealed in C#, `NotInheritable` in Visual Basic .NET – eine Klasse, von der man nicht erben kann). Sie dient vom Grundprinzip her dazu, eine Zeichenkette aus vielen verschiedenen Bestandteilen zusammenzubauen, wobei Sie ähnlich wie bei einer Collection Methoden zur Verfügung haben, um einzelne Zeichen oder ganze Zeichenketten anzuhängen, zu löschen oder an einer bestimmten Stelle einzufügen. Ein `StringBuilder`-Objekt ist damit so etwas wie ein »Zeichenketten-Baukasten«

StringBuilder

`StringBuilder` bietet ähnlich wie Collections Methoden zum Einfügen, Löschen oder Anhängen neuer einzelner Zeichen oder auch ganzer Zeichenketten. Weiterhin können einzelne Zeichen auch geändert werden. Über die Methode `ToString()` wird der Inhalt eines `StringBuilder`-Objekts als `System.String` zurückgeliefert. Sie haben Recht, wenn Sie sagen, dass man all das auch mit der `String`-Klasse erreichen kann; die Methoden von `StringBuilder` sind allerdings wesentlich schneller.

7.1.1 Eigenschaften und Methoden

Die Klasse `StringBuilder` implementiert vier Eigenschaften:

- ▶ `Capacity` enthält die Kapazität der `StringBuilder`-Instanz. Da ein `StringBuilder`-Objekt dynamisch wächst, können Sie an `Capacity` stets ablesen, wie viel Speicher aktuell vom Objekt belegt wird. `Capacity` wird dynamisch verwaltet und darf nicht kleiner als der Wert von `Length` werden.
- ▶ `Chars` ist der Indexer der Klasse `StringBuilder`. Über diese Eigenschaft können Sie auf die einzelnen enthaltenen Zeichen zugreifen.
- ▶ `Length` enthält die aktuelle Länge der im `StringBuilder` enthaltenen Zeichenkette. Die Länge ist immer kleiner als der Wert der Eigenschaft `Capacity`. (***)
- ▶ `MaxCapacity` ist ein konstanter Wert, der die größtmögliche Anzahl an Zeichen angibt, die ein `StringBuilder`-Objekt beinhalten kann. In der aktuellen Implementation entspricht dieser Wert dem Wert von `Int32.MaxValue`, also 2147483647 Zeichen oder 2 GByte. In zukünftigen Implementationen kann sich dieser Wert allerdings auch noch ändern.

Capacity Interessant ist, dass der Wert von `Capacity` zwar dynamisch wächst, Sie aber dennoch die Möglichkeit haben, diesen Wert zu ändern. Der Grund dafür ist, dass das .NET-Framework aus Gründen der Performance manchmal dafür sorgt, dass `Capacity` größer wird als eigentlich nötig. In diesem Fall können Sie den Wert auch herabsetzen.

EnsureCapacity() Den umgekehrten Weg gehen Sie über die Methode `EnsureCapacity()`. Hier können Sie festlegen, dass das `StringBuilder`-Objekt in jedem Fall Platz für eine bestimmte Anzahl an Zeichen vorsieht. `Capacity` wird daraufhin nicht mehr kleiner werden als der übergebene Wert.



Wenn Sie Änderungen an `Capacity` vornehmen, achten Sie immer darauf, dass der Wert von `Length` nicht unterschritten wird. Wenn der Wert von `Capacity` kleiner ist als der Wert von `Length`, wird eine `ArgumentOutOfRangeException` ausgelöst.

Die Methoden der Klasse `StringBuilder` erinnern ein wenig an Methoden einer Collection. Eigentlich handelt es sich ja auch um nichts anderes, nämlich um eine Collection aus Zeichen. Die

Methoden sind alle mehrfach überladen und recht einfach zu benutzen. Tabelle 7.1 listet sie auf.

Methode	Bedeutung
<code>Append()</code>	Fügt ein Zeichen oder eine Zeichenkette am Ende des <code>StringBuilder</code> -Objekts an
<code>AppendFormat()</code>	Fügt einen formatierten String am Ende des <code>StringBuilder</code> -Objekts an
<code>Insert()</code>	Fügt ein Zeichen oder eine Zeichenkette bzw. die String-Repräsentation eines Zahlenwerts an einer vorgegebenen Position ein
<code>Remove()</code>	Entfernt einen Bestandteil des im <code>StringBuilder</code> -Objekt enthaltenen Strings
<code>Replace()</code>	Ersetzt alle Vorkommen eines Zeichens oder eines Strings entweder im gesamten enthaltenen String oder in einem Bereich des enthaltenen Strings
<code>ToString()</code>	Liefert den Inhalt des <code>StringBuilder</code> -Objekts als String zurück

Tabelle 7.1: Wichtige Methoden der Klasse `StringBuilder`

Sie werden bemerkt haben, dass keine Methode `Clear()` existiert, mit der Sie den Inhalt des `StringBuilder`-Objekts löschen können. Eine solche Methode wird auch nicht benötigt – setzen Sie einfach `Length` auf 0. Wenn Sie `Length` einen Wert zuweisen, der kleiner ist als die aktuelle Länge des enthaltenen Strings, wird dieser auf die neu angegebene Länge gekürzt und `Capacity` entsprechend angepasst.



7.1.2 Erzeugen eines Zufalls-Strings

Der folgende recht einfache Code baut einen String aus einer Anzahl zufällig ausgewählter Zeichen auf. Der resultierende String könnte z.B. als Schlüssel für ein einfaches Chiffriersystem genutzt werden.

```
'=====
' RandomString.sln
' Copyright 2002 Frank Eller
' Erzeugen einer zufälligen Zeichenfolge
'=====
```

```
Imports System.Text
```

```

Public Class Form1
    Inherits System.Windows.Forms.Form

    #Region " Windows Form Designer generated code "

        'Der Code, der von VS.NET eingefügt wurde, wird
        'aus Übersichtsgründen nicht dargestellt.

    #End Region

    Private Sub btnCreate_Click( _
        ByVal sender As System.Object, _
        ByVal e As System.EventArgs) _
        Handles btnCreate.Click

        ' --- StringBuilder erzeugen
        Dim builder As New StringBuilder()
        ' --- Zufallsgenerator initialisieren
        Dim rand As New Random(DateTime.Now.Millisecond)
        Dim resultLength As Integer
        Dim i As Integer
        Dim randValue As Integer

        resultLength = Convert.ToInt32(tbxLength.Text)

        ' --- String erzeugen
        For i = 0 To resultLength
            randValue = rand.Next(26)
            ' --- +65 für Großbuchstaben
            builder.Append(ChrW(randValue + 65))
        Next

        tbxResult.Text = builder.ToString()

    End Sub

    Private Sub btnExit_Click( _
        ByVal sender As System.Object, _
        ByVal e As System.EventArgs) _
        Handles btnExit.Click

        ' --- Beenden
        Me.Close()
    End Sub
End Class

```

Listing 7.1: RandomString.exe

tbxResult ist, wie aus dem Programmtext ersichtlich, die Textbox, in der das Ergebnis angezeigt wird. tbxLength ist die Textbox, in der die Länge des zu erzeugenden Strings angezeigt wird. Da in

diesem Code keine Fehlerverwaltung enthalten ist, muss man darauf achten, wirklich nur Zahlen einzugeben.

Im Beispiel wird die Methode `Append()` benutzt, um jeweils ein Zeichen an den String anzuhängen. Ebenso einfach ist es, Zeichen oder einen ganzen String einzufügen (über die Methode `Insert()`) oder zu entfernen (über die Methode `Remove()`).

Abbildung 7.1 zeigt das Programm zur Laufzeit mit dem Resultat.



Abbildung 7.1: Erzeugen eines zufälligen Strings (*RandomString.exe*)

7.1.3 Mehrfaches Einfügen von Strings

`Insert()` fügt einen String, ein Zeichen oder aber die String-Repräsentation eines Zahlenwerts an der angegebenen Position ein. Bei der Verwendung von `Insert()` muss man lediglich darauf achten, mit der Positionsangabe nicht über die aktuelle Länge des im `StringBuilder` enthaltenen Strings zu kommen, was eine `ArgumentOutOfRangeException` zur Folge hätte.

Insert()

`Insert()` ist vielfach überladen und arbeitet mit unterschiedlichen Formaten. Beispielsweise können Sie einem String auch Zahlenwerte hinzufügen, wobei natürlich die jeweilige Entsprechung der Zahl als String eingefügt wird. Eine weitere Möglichkeit besteht z.B. darin, einen Wert mehrfach einzufügen, wie im folgenden Beispiel.

```
'=====
' InsertMulti.sln
' Copyright 2002 Frank Eller
' Mehrfaches Einfügen eines Strings
'=====
```

```
Imports System.Text
```

```
Public Class Form1
```

```
    Inherits System.Windows.Forms.Form
```

```
#Region " Windows Form Designer generated code "
```

*'Der Code, der von VS.NET eingefügt wurde, wird
'aus Übersichtsgründen nicht dargestellt.*

#End Region

```
Private builder As New StringBuilder()

Private Sub btnInsert_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnInsert.Click

    ' --- Deklarationen
    Dim insertPos As Integer
    Dim insertCount As Integer
    Dim insertStr As String

    ' --- Initialisieren der Werte
    insertPos = Convert.ToInt32(tbxPosition.Text)
    insertCount = Convert.ToInt32(tbxCount.Text)
    insertStr = tbxInsert.Text

    ' --- Einfügeposition muss kleiner Length sein
    If insertPos > builder.Length Then
        insertPos = builder.Length
    End If

    ' --- Anzahl Einfügen muss größer 0 sein
    If insertCount < 1 Then
        insertCount = 1
    End If

    ' --- Einfügen
    builder.Insert(insertPos, insertStr, insertCount)
    tbxResult.Text = builder.ToString()
End Sub

Private Sub btnClear_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnClear.Click

    builder.Length = 0
    tbxResult.Text = builder.ToString()
End Sub

Private Sub btnExit_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnExit.Click
```

```
' --- Beenden  
Me.Close()  
End Sub  
End Class
```

Listing 7.2: InsertMulti.exe

Abbildung 7.2 zeigt das Programm zur Laufzeit.



Abbildung 7.2: Zeichen mehrfach einfügen (InsertMulti.exe)

7.2 Reguläre Ausdrücke

7.2.1 Übersicht

Das .NET-Framework bietet mit dem Namespace `System.Text.RegularExpressions` eine kleine Anzahl von Klassen, die dazu dienen, Textpassagen effizient und flexibel durchsuchen zu können. Schlüssel hierzu sind die so genannten *regulären Ausdrücke*, eine Abfragesprache für die Suche innerhalb eines Textes.

Bei regulären Ausdrücken handelt es sich um einen Mustervergleich, wie man ihn auch in Sprachen wie Perl oder PHP wiederfindet. Über ein so genanntes *Pattern* wird der Suchausdruck festgelegt. Dabei kann mit Hilfe von Sonderzeichen und Escape-Sequenzen das Verhalten des Suchausdrucks festgelegt werden. Andere Zeichen als diese Sonderzeichen werden als sie selbst behandelt. Teilweise lassen sich die Sonderzeichen mit den Platzhaltern unter DOS vergleichen, wo das Zeichen »*« für jedes beliebige Zeichen steht oder »?« für genau ein Zeichen. Diese »Platzhalter« können Sie auch in regulären Ausdrücken verwenden, zusammen mit weiteren, sie haben dort jedoch eine andere Bedeutung.

Mustervergleich

Ein Beispiel für einen Mustervergleich wäre das Zählen aller Worte innerhalb eines Textes, die mit dem Buchstaben »E« beginnen. Mit herkömmlichen Methoden wäre das eine recht umfangreiche Arbeit. Zunächst müssten alle Wörter erst einmal gefunden und dann für jedes Wort kontrolliert werden, ob es mit »E« beginnt. Noch umfangreicher wird es, wenn dazu nur die Wörter ausgefiltert werden sollen, die nicht nur mit »E« beginnen, sondern auch noch mit »d« enden – dann müsste das Wortende auch noch kontrolliert werden.

Regex Mit Hilfe regulärer Ausdrücke können Sie ein solches Verhalten mit wenig Aufwand realisieren. Den Kern der Funktionalität bildet die Klasse `Regex`, die für die eigentliche Auswertung zuständig ist. Das `Pattern` liefert das Muster für den Vergleich. Heraus kommt eine Gruppe gefundener Textstellen, die später ausgewertet werden können. Das Suchmuster für die Suche nach allen Wörtern, die mit »E« beginnen, sähe so aus:

```
\bE\S+\b
```

Pattern Zunächst wird nach einer Wortbegrenzung gesucht. Dafür steht die Escape-Sequenz `\b`. Danach soll ein »E« folgen und nach dem »E« eine beliebige Anzahl an Zeichen, die keine speziellen Zeichen (z.B. Tabulatoren) oder Leerzeichen darstellen. Solche Zeichen nennt man *Whitespaces*. Die entsprechende Escape-Sequenz für einen *Whitespace* ist `\S`; das `+` dahinter bedeutet, dass beliebig viele dieser Zeichen folgen dürfen. Und danach folgt natürlich das Wortende, also eine Wortbegrenzung dargestellt durch `\b`.

Eine Liste der Klassen, die für Mustervergleiche relevant sind, finden Sie in Tabelle 7.2.

Klasse	Funktion
Capture	Capture repräsentiert einen gefundenen Suchstring innerhalb eines Strings. Capture selbst kann nicht instanziiert werden, die Elemente einer CaptureCollection liefern allerdings Instanzen von Capture zurück.
CaptureCollection	CaptureCollection liefert mehrere Instanzen von Capture zurück.

Tabelle 7.2: Die Klassen des Namespaces `System.Text.RegularExpressions`

Klasse	Funktion
Group	Group liefert eine Gruppe von Capture-Objekten zurück. In einem Objekt vom Typ Group können eines oder mehrere Capture-Elemente enthalten sein. Mit Hilfe von Gruppen ist es möglich, eine einzelne Abfrage in verschiedene Bestandteile zu unterteilen, die später zusammen oder getrennt ausgewertet werden können.
GroupCollection	GroupCollection ist eine Sammlung von Group-Objekten.
Match	Match repräsentiert eine komplette Abfrage eines oder mehrerer regulärer Ausdrücke. Innerhalb einer solchen Abfrage ist auch eine GroupCollection vorhanden, wobei die erste Gruppe immer alle gefundenen Textstellen des kompletten Ausdrucks repräsentiert und die weiteren Gruppen die einzelnen Unterbestandteile des Suchausdrucks.
MatchCollection	MatchCollection repräsentiert eine Sammlung von Match-Objekten.
Regex	Regex ist die zentrale Klasse dieses Namespaces. Mit dieser Klasse erfolgt die Auswertung der regulären Ausdrücke.
RegexCompilationInfo	Reguläre Ausdrücke können kompiliert und somit schneller gemacht werden. Das ist vor allem bei komplexen Ausdrücken eine sinnvolle Vorgehensweise. RegexCompilationInfo liefert Informationen für den Compiler.

Tabelle 7.2: Die Klassen des Namespaces `System.Text.RegularExpressions` (Fortsetzung)

Es mag ein wenig unklar sein, was es mit Groups, Captures und Matches auf sich hat. Reguläre Ausdrücke sind wie schon angesprochen eine enorm mächtige Möglichkeit der Textsuche, und zu dieser Mächtigkeit gehört unter anderem auch, dass ein Suchausdruck verschiedene Suchgruppen beinhalten kann. Die Ergebnisse dieser Suchgruppen werden dann in Group-Objekten gespeichert. Das erste Group-Objekt beinhaltet immer den gesamten Suchausdruck, also alle gefundenen Textstellen.

Jede gefundene Textstelle wird in einem Capture-Objekt festgehalten. Alle gefundenen Textstellen einer Gruppe können Sie dann über die Eigenschaft `Captures` der Klasse `Group` ermitteln. Wenn Sie alle Suchergebnisse ermitteln wollen, liefert auch `Match` eine Eigenschaft `Captures`, die dann alle Capture-Objekte einer Suche zurückliefert. Das entspricht auch den Ergebnissen der ersten Group, die ebenfalls alle Suchergebnisse liefert.

Capture

Bevor es allerdings mit der Anwendung regulärer Ausdrücke losgehen kann, erst noch ein paar Informationen darüber, wie solche Suchausdrücke aussehen und was man mit ihnen machen kann. Reguläre Ausdrücke können sehr komplex sein, dabei aber umfangreiche Abfragen in einem einzigen String zusammenfassen.

7.2.2 Suchausdrücke (Patterns)

Innerhalb regulärer Ausdrücke (oder Patterns) gelten alle Zeichen als sie selbst, ausgenommen einige Sonderzeichen und Escape-Sequenzen. Wie ein solches Pattern aufgebaut ist, haben Sie weiter oben schon gesehen. Tabelle 7.3 listet einige der Sonderzeichen für reguläre Ausdrücke auf, Tabelle 7.4 einige der Escape-Sequenzen. Für weitere Informationen suchen Sie in der Hilfe des Visual Studio .NET nach »Pattern«.

Sonderzeichen	Bedeutung bei der Suche
^	Anfang des Textes oder einer Zeile im Multiline-Modus. Der Ausdruck ^B kontrolliert, ob das erste Wort des Textes bzw. der Zeile mit B beginnt.
\$	Ende des Textes. Der Ausdruck \$n kontrolliert, ob der letzte Buchstabe des Textes ein n ist.
.	Jedes beliebige einzelne Zeichen
*	Das vorhergehende Zeichen wird nicht oder mehrfach wiederholt. Der Ausdruck Wa*I findet z.B. WI, Wal, Waal, Waaal usw.
+	Wie *, das Zeichen muss aber mindestens einmal vorkommen. Wa+I würde also ebenfalls Wal, Waal, Waaal finden, aber nicht WI.
?	Das vorhergehende Zeichen wird kein- oder einmal wiederholt. Wa?I würde also WI und Wal finden, aber nicht Waal usw.
	Sucht nach verschiedenen Ausdrücken, die mit getrennt sind (verhält sich also wie ein »oder«). Der Ausdruck Katze Tiger Hund würde also eines der angegebenen Tiere im Text finden.

Tabelle 7.3: Sonderzeichen für Suchausdrücke

Sequenz	Bedeutung bei der Suche
\A	Anfang des Textes. Anders als bei der Verwendung von ^ wird die Multiline-Option nicht beachtet.

Tabelle 7.4: Escape-Sequenzen für reguläre Ausdrücke

Sequenz	Bedeutung bei der Suche
\Z	Ende des Textes. Ein Zeilenumbruch am Ende des Textes wird ignoriert.
\z	Ende des Textes. Ein Zeilenumbruch am Ende des Textes wird nicht ignoriert.
\B	Das gesuchte Zeichen darf sich nicht am Anfang oder am Ende eines Wortes befinden.
\b	Bezeichnet Anfang und Ende eines Wortes
\S	Sucht nach allen Zeichen, die keine Whitespaces sind. Whitespaces sind Sonderzeichen (Escape-Charakter) und das Leerzeichen.
\s	Sucht nach Whitespace-Zeichen

Tabelle 7.4: Escape-Sequenzen für reguläre Ausdrücke (Fortsetzung)

7.2.3 Die Klasse Regex

Die Klasse `Regex` ist der Kern der Suche mit regulären Ausdrücken. Die Basis ist der eigentliche Suchausdruck, der im Allgemeinen als Pattern bezeichnet wird. Sie können den Suchausdruck entweder bereits bei der Instanziierung des `Regex`-Objekts angeben oder aber erst bei der Anwendung der Suchmethode `Match` bzw. `Matches`. Die zweite Vorgehensweise hat den Vorteil, dass Sie für jede Suche einfach ein anderes Pattern angeben können, anstatt ein neues `Regex`-Objekt erzeugen zu müssen.

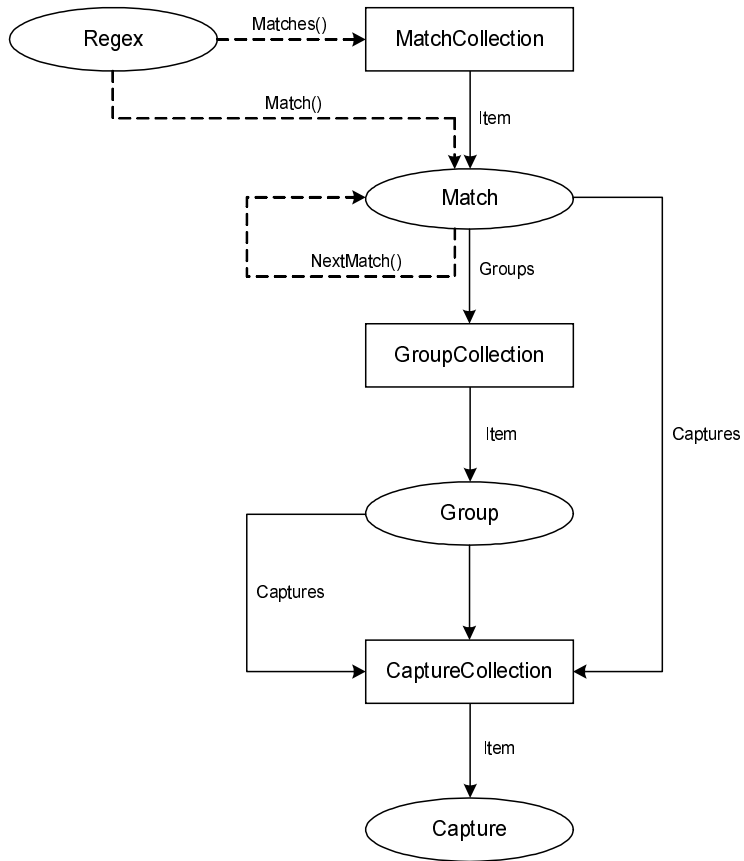
Patterns

Jedes Objekt vom Typ `Match` repräsentiert mehrere, in Gruppen (Groups) geordnete Suchergebnisse (Captures). Dabei ist die erste Gruppe immer das Suchergebnis über den ganzen Ausdruck. Ist der Ausdruck selbst gruppiert, enthalten weitere Gruppen die entsprechenden Suchergebnisse. Der Zusammenhang zwischen den Klassen `Regex`, `Match`, `Group` und `Capture` wird im Diagramm in Abbildung 7.3 etwas klarer dargestellt.

Matches, Captures und Groups

Die Klasse `Capture` repräsentiert jeweils ein Vorkommen gemäß einem Suchausdruck innerhalb eines Strings. Über die Eigenschaft `Index` lässt sich herausfinden, an welcher Stelle der Textbestandteil gefunden wurde. Über die Eigenschaft `Length` lässt sich die Länge des Suchstrings ermitteln und die Eigenschaft `Value` liefert den Suchausdruck im Klartext zurück.

Capture



© by Frank Ellert

Abbildung 7.3: Zusammenhang zwischen den Klassen aus *System.Text.RegularExpressions*

Suchen des ersten Auftretens

Suchen mit Regex

Für eine einfache Suche ist es nicht notwendig, gleich alle Klassen zu verwenden. Die Klasse `Match` beinhaltet ja bereits eine Eigenschaft `Captures`, die die dem Suchausdruck entsprechenden Vorkommen beinhaltet. Das folgende Beispiel benutzt die `Regex`-Klasse, um das erste Vorkommen eines Ausdrucks zu ermitteln. Das erste Vorkommen entspricht der Eigenschaft `Captures(0)` bzw. `Groups(0).Captures(0)` des `Match`-Objekts. Verwendet wird hier die Methode `Match()` des `Regex`-Objekts, weil ja nur das erste Vorkommen relevant sein soll.

Das Hauptformular besteht aus einer Textbox mit Namen `txtText`, die den zu durchsuchenden Text enthält. Weiterhin gibt es eine Textbox `txtPattern`, die den Suchausdruck aufnimmt, und je einen Button zum Suchen und zum Beenden des Programms.

```
'=====
' CaptureText.sln
' Copyright 2002 Frank Eller
' Suchen nach erstem Vorkommen im Text
'=====

Imports System.Text.RegularExpressions

Public Class Form1
    Inherits System.Windows.Forms.Form

    #Region " Windows Form Designer generated code "

        'Der Code, der von VS.NET eingefügt wurde, wird
        'aus Übersichtsgründen nicht dargestellt.

    #End Region

    Private Function FindPattern( _
        ByVal pattern As String) As Integer

        ' --- Deklarationen
        Dim searchText As String
        Dim findpos As Integer
        Dim r As Regex
        Dim m As Match

        ' --- Suchen
        r = New Regex(pattern)
        m = r.Match(txtText.Text)

        Return m.Captures(0).Index

    End Function

    Private Sub btnCapture_Click( _
        ByVal sender As System.Object, _
        ByVal e As System.EventArgs) _
        Handles btnCapture.Click

        lblResult.Text = "Bei Index " + _
            FindPattern(txtPattern.Text).ToString()

    End Sub
```

```

Private Sub btnExit_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnExit.Click

    Me.Close()

End Sub
End Class

```

Listing 7.3: Suche nach erstem Vorkommen [CaptureText.sln]

Die eigentliche Suche wird von der Methode `FindPattern()` erledigt, die den Index des ersten Suchergebnisses als `Integer`-Wert zurückliefert. Angegeben werden kann jedes beliebige Pattern, zurückgeliefert wird jedoch nur der Index des ersten Auftretens.

Zählen von Wörtern

Eine Sache, die bereits kurz angesprochen wurde und mit herkömmlichen Mitteln nicht so leicht zu realisieren ist, ist das Zählen der Wörter in einem Text oder auch das Zählen aller Zeichen, die keine *Whitespaces*, also Leer- oder Sonderzeichen sind. Mit Hilfe regulärer Ausdrücke ist dieses Problem trivial.

Kern der Abfrage ist eine Methode, die auf Basis eines Patterns die Vorkommen zählt. Danach muss lediglich noch das Pattern festgelegt werden, das für ganze Wörter oder für alle Zeichen gilt. Die Methode zur Auswertung ist recht einfach.

```

Private Function doCount(ByVal pattern As String) _
    As Integer

    ' --- Deklarationen
    Dim aRegex As Regex
    Dim aMatchColl As MatchCollection

    ' --- Zählen
    aRegex = New Regex(pattern)
    aMatchColl = aRegex.Matches(txtText.Text)

    Return aMatchColl.Count

End Function

```

Das Formular besteht aus einer Textbox für den zu durchsuchenden Text, Buttons zum Laden eines beliebigen Textes und Labels zur Ausgabe des Ergebnisses. Das gesamte Programm zeigt Listing 7.4.

```

'=====
' CountWords.sln
' Copyright 2002 Frank Eller
' Zählen von Wörtern in einem Text
'=====

Imports System.IO
Imports System.Text.RegularExpressions

Public Class Form1
    Inherits System.Windows.Forms.Form

#Region " Windows Form Designer generated code "

    'Der Code, der von VS.NET eingefügt wurde, wird
    'aus Übersichtsgründen nicht dargestellt.

#End Region

    Private Function doCount(ByVal pattern As String) As Integer

        ' --- Deklarationen
        Dim aRegex As Regex
        Dim aMatchColl As MatchCollection

        ' --- Zählen
        aRegex = New Regex(pattern)
        aMatchColl = aRegex.Matches(txtText.Text)

        Return aMatchColl.Count

    End Function

    Private Sub btnLoad_Click( _
        ByVal sender As System.Object, _
        ByVal e As System.EventArgs) _
        Handles btnLoad.Click

        ' --- Deklarationen
        Dim sReader As StreamReader

        ' --- Dialog anzeigen
        If dlgOpenFile.ShowDialog() = _
            DialogResult.OK Then
            sReader = New StreamReader( _
                dlgOpenFile.FileName)
            txtText.Text = sReader.ReadToEnd()
        End If
    End Sub

    Private Sub btnCountWords_Click( _

```

```

ByVal sender As System.Object, _
ByVal e As System.EventArgs) _
Handles btnCountWords.Click

' --- Deklarationen
Dim pattern As String = "\b(\S+)\b"

lblWords.Text = doCount(pattern).ToString()

End Sub

Private Sub btnCountChars_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnCountChars.Click

' --- Deklarationen
Dim pattern As String = "\S"

lblChars.Text = doCount(pattern).ToString()

End Sub

Private Sub btnExit_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnExit.Click

Me.Close()

End Sub
End Class

```

Listing 7.4: Durchsuchen nach Wörtern und Zeichen [CountWords.sln]

**Pattern für die
Wortsuche**

Das Pattern für die Suche nach ganzen Wörtern lautet `\b(\S+)\b`. Zunächst das `\b` für den Beginn eines Wortes, dann `\S+` für eine beliebige Anzahl Zeichen, die allerdings keine Whitespaces sein dürfen, und dann wieder die Wortbegrenzung mit `\b`. In diesem Beispiel wurde zum ersten Mal eine Gruppierung verwendet, die man an den runden Klammern erkennt. Ein weiteres Beispiel wird noch zeigen, wie man mit diesen Gruppen arbeiten kann.

**Pattern für
Einzelzeichen**

Das Pattern zum Herausfinden der einzelnen Zeichen ist noch einfacher, nämlich nur `\S` – damit wird jedes Zeichen gezählt, das kein Whitespace ist. Wollten Sie z.B. alle Zeichen zählen, die Whitespaces sind, könnten Sie das Pattern `\s` verwenden. Abbildung 7.4 zeigt das Programm zur Laufzeit.

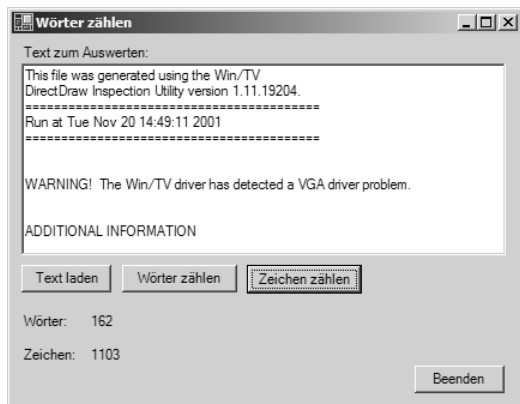


Abbildung 7.4: Zählen von Wörtern mit Hilfe regulärer Ausdrücke [CountWords.sln]

Gruppen

Suchausdrücke können gruppiert werden. Diese Gruppierung funktioniert genauso wie in der Mathematik, nämlich mit Hilfe von Klammern, die um die jeweilige Gruppe gelegt werden. Dadurch werden die regulären Ausdrücke recht flexibel.

**Gruppieren
durch Klammern**

In dem folgenden Beispiel werden alle Gruppen und die darin enthaltenen `Captures` aufgelistet. Die Oberfläche des Programms besteht wieder aus einer Textbox mit dem zu durchsuchenden Text, einer Textbox für die Eingabe des Patterns und einer Listbox namens `lbxResult`, in der die Ergebnisse angezeigt werden. Der Quelltext selbst ist nicht sonderlich kompliziert. Wenn Sie ein Pattern mit einer Gruppierung eingeben, sehen Sie, wie die Ergebnisse sich auf die Gruppen verteilen.

```
'=====
' CaptureGroups.sln
' Copyright 2002 Frank Eller
' Beispiel für Gruppen in Mustervergleichen
'=====
```

```
Imports System.Text.RegularExpressions
```

```
Public Class Form1
    Inherits System.Windows.Forms.Form
```

```
#Region " Windows Form Designer generated code "
```

```
'Der Code, der von VS.NET eingefügt wurde, wird
'aus Übersichtsgründen nicht dargestellt.
```

```

#End Region

Private Sub btnExit_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnExit.Click

    Me.Close()
End Sub

Private Function FindMatches( _
    ByVal pattern As String) As ArrayList

    ' --- Deklarationen
    Dim aRegex As Regex
    Dim result As New ArrayList()
    Dim foundMatches As MatchCollection
    Dim aMatch As Match
    Dim aGroup As Group
    Dim aCapture As Capture
    Dim groupCount As Integer = 0
    Dim matchCount As Integer = 0
    Dim captureCount As Integer = 0

    ' --- Suchen
    aRegex = New Regex(pattern)
    foundMatches = aRegex.Matches(txtText.Text)

    ' --- Ergebnisse aufzeichnen
    For Each aMatch In foundMatches
        matchCount += 1
        groupCount = 0
        captureCount = 0
        result.Add("Match: " + matchCount.ToString())
        For Each aGroup In aMatch.Groups
            groupCount += 1
            captureCount = 0
            result.Add("--- Group: " + _
                groupCount.ToString())
            For Each aCapture In aGroup.Captures
                captureCount += 1
                result.Add("    --- Capture " + _
                    captureCount.ToString() + " " + _
                    " Index: " + aCapture.Index.ToString())
            Next
        Next
    Next

    Return result
End Function

```



```
Private Sub btnFind_Click( _  
    ByVal sender As System.Object, _  
    ByVal e As System.EventArgs) _  
    Handles btnFind.Click  
  
    ' --- Deklarationen  
    Dim resultList As ArrayList  
  
    ' --- Suche starten  
    lbxResult.Items.Clear()  
    resultList = FindMatches(txtPattern.Text)  
    lbxResult.Items.AddRange(resultList.ToArray())  
  
End Sub  
  
End Class
```

Listing 7.5: Anzeige der Gruppen bei regulären Ausdrücken [CaptureGroups.sln]

8 Namespaces

System.Globalization und System.Resources

Zumeist wird ein Programm, das auf dem Markt präsentiert werden soll, nicht nur in einer Sprache veröffentlicht. Für kommerzielle Programme ist das sogar ein absolutes Muss, will man konkurrenzfähig bleiben. Leider ist der Aufwand, alle Ressourcen eines Programms zu übersetzen, relativ hoch, und auch beim Design einer Applikation muss darauf geachtet werden, dass Lokalisierung möglich ist.

Die zwei Namespaces, die Lokalisierung ermöglichen und auch relativ einfach machen, sind `System.Globalization` und `System.Resources`. In beiden sind einige Klassen enthalten, die die Lokalisierung einer Applikation stark vereinfachen.

8.1 Lokalisierung

Zur Umstellung eines Programms auf eine andere Sprache gehört nicht nur, alle Zeichenketten zu übersetzen, die das Programm enthält. Vielmehr ist es so, dass eine komplette Anpassung auf die Gegebenheiten des Landes erfolgen muss, in dem das Programm später laufen soll. Dazu kann es auch nötig sein, die Position von Komponenten auf einem Formular zu ändern oder möglicherweise eine Grafik anzupassen.

Sprachumstellung

Wichtig wird dies vor allem, wenn ein Programm für ein Land angepasst werden soll, in dem nicht von links nach rechts gelesen wird wie bei uns, sondern von rechts nach links. Die Positionen der Buttons sind zwangsläufig vertauscht. Natürlich muss auch der Text in den verschiedenen Komponenten geändert werden, die Anzeige von Datum und Zeit ist eine andere (es gibt einen anderen Kalender mit einer komplett unterschiedlichen Datums-

messung) und möglicherweise muss auch die gesamte Benutzerschnittstelle auf die Gegebenheiten dieses Landes umgestellt werden.

8.1.1 Ressourcendateien

Ressourcen-dateien

Alle Daten, die bei einer solchen Lokalisierung geändert werden müssen, werden dazu in einer so genannten *Ressourcendatei* gespeichert. Eine solche Datei kann alle Arten von Daten enthalten, von Strings über Grafiken und Maße bis hin zu Sound-Daten. Die Verwendung solcher Dateien gestaltete sich aber vor .NET etwas schwierig, was unter anderem mit dem Aufbau dieser Dateien zu tun hatte. Im Grunde waren es unter Windows immer kompilierte Textdateien, bei denen die unterschiedlichen Ressourcen ähnlich wie in einer *.ini*-Datei nacheinander als Klartext aufgeführt und mit einer ID versehen wurden. Wollte man eine Ressource laden, war es – mit VB6 – nötig, diese ID anzugeben. Damit man sich nicht alle diese Zahlen merken musste, griff man in der Regel auf aussagekräftige Konstanten zurück, die irgendwo im Programm hinterlegt wurden.

Übersetzung

Der Vorteil einer solchen Ressourcendatei war natürlich, dass sie einfach übersetzt werden konnte – immerhin handelte es sich um Text, d.h. man gab einfach die Textdatei weiter, ließ sie übersetzen und kompilierte sie unter einem anderen Namen. Innerhalb dieses Programms wurde dann kontrolliert, welche Ländereinstellungen das Betriebssystem hatte, und dementsprechend wurde die Ressourcendatei geladen.

Lokalisierung in .NET

Grundsätzlich hat sich daran nichts geändert, allerdings nimmt das .NET-Framework einem eine Menge Arbeit ab. Ressourcendateien können recht einfach erzeugt werden, indem man ein Formular als Lokalisierbar kennzeichnet (über eine Eigenschaft) und dann einfach das Land einstellt, für das man eine Ressourcendatei erzeugen will. Welche Daten darin gespeichert werden und wie man diese Daten ändern kann, wird später noch angesprochen werden.

8.1.2 Fehler vermeiden

Wichtig ist auch, dass ein Programm bereits bei der Planung darauf vorbereitet werden muss, lokalisierbar zu sein. Wann Sie die Umstellung vornehmen, ist dabei egal, das können Sie zu jeder Zeit tun. Die folgenden Dinge allerdings gilt es zu vermeiden:

- ▶ Vermeiden Sie String-Konstanten in Ihrem Programmcode. Alle Zeichenketten, jeder Text der irgendwo erscheinen soll, sollte aus der Ressourcendatei kommen. Hartcodierte Daten würden eine Neukompilierung erforderlich machen, was unnötiger Aufwand wäre. Wenn Sie gut planen, muss lediglich die Ressourcendatei übersetzt werden.
- ▶ Vermeiden Sie lange Zeichenketten. Es ist gut möglich, dass eine Zeichenkette, die in Ihrem Land noch gut auf den Bildschirm passt, in einem anderen Land eben zu lang wird. In der Regel ist die Sprache mit den kürzesten Sätzen Englisch, in anderen Sprachen kann ein kurzer String mit 20 Zeichen auch mal gerne 70-80% länger werden.
- ▶ Vermeiden Sie es, verschiedene Ressourcen-Strings zusammenzusetzen. Das Zusammensetzen in einer Sprache mag funktionieren, in einer anderen Sprache kommt dabei etwas Unverständliches heraus, das kein Mensch versteht. Benutzen Sie dazu lieber zwei Ressourcen-Strings, auch wenn sie sich sehr gleichen.
- ▶ Vermeiden Sie feste Formatierungen von Zahlen und Datumsangaben. Im Deutschen ist das Datumsformat »TT.MM.JJJJ«, wobei »TT« für den Tag, »MM« für den Monat und »JJJJ« für die vierstellige Jahreszahl steht. Im Englischen ist die Datumsangabe »MM/TT/JJJJ«. Wenn Sie eine solche Angabe hart formatieren, haben Sie ein Problem.
- ▶ Beschäftigen Sie einen Übersetzer, der etwas von der Materie versteht und das Programm kennt. Diese Regel mag den einen oder anderen etwas verwundern, Aber auch Sie haben sicherlich schon mit einem schlecht lokalisierten Programm gearbeitet haben, bei dem Sie über die Übersetzung gelacht (oder geflucht) haben. Damit das nicht passiert, übersetzen Sie entweder selbst (falls möglich) oder suchen Sie einen wirklich guten Übersetzer.

Es gibt noch weitere Verhaltensregeln, die sich aber im Großen und Ganzen auf die genannten zurückführen lassen. Beispielsweise sollten Sie auch in Menüs nie zu lange Wörter verwenden, die in einer anderen Sprache ggf. einen Umbruch des Menüs zur Folge haben.

8.2 Kalenderklassen

Calendar Verschiedene Länder benutzen unter Umständen verschiedene Kalender. Die verschiedenen Kalenderklassen sind möglicherweise nicht unbedingt die wichtigsten Klassen, was Lokalisierung angeht, es ist aber dennoch interessant, welche Möglichkeiten Ihnen diese Kalender bieten. Alle sind von der abstrakten Basisklasse `Calendar` abgeleitet, was eine Erweiterung dieser Klassen, sollte eine weitere wichtige Kalenderform auftauchen, erleichtert. Tabelle 8.1 zeigt eine Übersicht über die Kalenderarten mit ihren Eigenschaften.

Klasse	Bedeutung
<code>GregorianCalendar</code>	Repräsentiert den gregorianischen Kalender. Dies ist der Kalender, der bei uns Anwendung findet. Schaltjahre sind Jahre, die ohne Rest durch vier teilbar sind, es sei denn, sie sind durch 100 teilbar. Jahreszahlen, die durch 400 teilbar sind, stehen ebenfalls für ein Schaltjahr.
<code>HebrewCalendar</code>	Repräsentiert den hebräischen Kalender. Dieser Kalender hat ebenfalls zwölf Monate, außer in einem Schaltjahr – dann sind es 13 Monate. Schaltjahre werden in Bezug auf einen 19-Jahre-Zyklus bestimmt.
<code>HijriCalendar</code>	Die Klasse <code>HijriCalendar</code> repräsentiert einen muslimischen Kalender. Schaltjahre werden in Bezug auf einen 30-Jahre-Zyklus festgelegt. Die Anzahl der Tage eines Jahres variiert zwischen 354 und 355 Tagen. Der Monat Ramadan ist der Fastenmonat; durch die geringere Anzahl der Tage eines Jahres variiert dessen Lage im Bezug auf den gregorianischen Kalender.
<code>JapaneseCalendar</code>	Der japanische Kalender funktioniert wie der gregorianische Kalender, Jahreszahl und Ära sind allerdings verschieden. Der 1.1.2001 (gregorianisch) entspricht dem ersten Tag des Monats Ichigatsu im 13. Jahr der Ära Heisei.
<code>JulianCalendar</code>	Der julianische Kalender ist der Vorgänger des gregorianischen Kalenders und wird heute nicht mehr offiziell benutzt. Dadurch eignet sich diese Klasse nur noch für Berechnungen, die den julianischen Kalender zur Basis haben. Der Unterschied zum gregorianischen Kalender besteht in der Behandlung der Schaltjahre; diese sind beim julianischen Kalender alle Jahre, deren Jahreszahl durch vier teilbar ist, ohne Ausnahme.

Tabelle 8.1: Kalenderklassen in *System.Globalization*

Klasse	Bedeutung
KoreanCalendar	Der koreanische Kalender funktioniert ebenfalls wie der gregorianische, wobei auch hier Ära und Jahreszahl unterschiedlich sind. Zum Vergleich: Der 1.1.2001 (gregorianisch) entspricht dem 1.1.4334 (koreanisch).
TaiwanCalendar	Der taiwanesishe Kalender funktioniert wiederum wie der gregorianische. Allerdings sind Ära und Jahreszahl erneut verschieden. Der 1.1.2001 entspricht dem 1.1.90 des taiwanesischen Kalenders.
ThaiBuddhistCalendar	Auch der buddhistische Kalender funktioniert wie der gregorianische, allerdings sind hier wieder einmal Jahreszahl und Ära verschieden. Der 1.1.2001 entspricht dem 1.1.2544 des buddhistischen Kalenders.

Tabelle 8.1: Kalenderklassen in System.Globalization (Fortsetzung)

Kalenderklassen eignen sich selbstverständlich hauptsächlich für Berechnungen oder Ermittlungen, die mit einem Datum zu tun haben. Alle Kalenderklassen besitzen zwei gemeinsame Eigenschaften, die sie von der Basisklasse `Calendar` geerbt haben:

- ▶ `Eras` liefert alle Ären eines Kalenders zurück, sofern dieser mehrere verwendet. Der gregorianische Kalender z.B. berücksichtigt nur die aktuelle Ära.
- ▶ `TwoDigitYearMax` steht für die letzten zwei Ziffern einer Jahreszahl. Der Wert kann gelesen oder geschrieben werden.

Weitere Methoden der Kalenderklassen finden Sie in Tabelle 8.2.

Methode	Bedeutung
<code>AddDays()</code>	Liefert einen <code>DateTime</code> -Wert, der ein Datum und eine Zeit repräsentiert, die die angegebene Anzahl Tage von einem vorgegebenen <code>DateTime</code> -Wert entfernt ist/sind. Das Gleiche funktioniert auch mit Stunden, Millisekunden, Minuten, Monaten, Sekunden, Wochen oder Jahren. Die entsprechenden Methoden heißen dann <code>AddHours()</code> , <code>AddMilliseconds()</code> , <code>AddMinutes()</code> , <code>AddMonths()</code> , <code>AddSeconds()</code> , <code>AddWeeks()</code> oder <code>AddYears()</code> .
<code>GetDayOfMonth()</code>	Liefert die Nummer des aktuellen Tages innerhalb eines Monats zurück. Der <code>DateTime</code> -Wert mit dem aktuellen Datum muss angegeben werden.

Tabelle 8.2: Die Methoden der Kalenderklassen

Methoden	Bedeutung
GetDayOfWeek()	Liefert den zum angegebenen Datum passenden Wochentag zurück. Der Wert ist vom Typ <code>DayOfWeek</code> , einem Aufzählungstyp mit den Werten <code>Sunday</code> bis <code>Saturday</code> .
GetDayOfYear()	Liefert wie <code>GetDayOfMonth</code> einen Zahlenwert zurück, der die angegebene Position des Tages innerhalb eines Jahres zurückliefert. Ausgangspunkt ist der übergebene <code>DateTime</code> -Wert.
GetDaysInMonth()	Liefert die Anzahl der Tage in dem angegebenen Monat. Es müssen Monat und Jahr, jeweils durch einen <code>Integer</code> -Wert, angegeben werden.
GetDaysInYear()	Liefert die Anzahl der Tage in dem angegebenen Jahr. Das Jahr wird durch einen <code>Integer</code> angegeben.
GetMilliseconds()	Liefert die aktuelle Anzahl der Millisekunden des angegebenen <code>DateTime</code> -Werts
GetMinutes()	Liefert die aktuelle Anzahl der Minuten des angegebenen <code>DateTime</code> -Werts
GetMonthsInYear()	Liefert die Anzahl der Monate in dem Jahr zurück, das durch einen <code>Integer</code> -Wert angegeben wurde
GetSecond()	Liefert die Sekunden des angegebenen <code>DateTime</code> -Werts zurück
GetWeekOfYear()	Liefert die Nummer der Woche innerhalb des Jahres zurück. Bezugspunkt ist auch hier das Jahr als <code>Integer</code> -Wert.
IsLeapDay()	Liefert <code>True</code> zurück, wenn das übergebene Datum einem Schalttag entspricht. Im gregorianischen Kalender wäre das der 29. Februar. Übergeben werden Jahr, Monat und Tag als <code>Integer</code> -Werte.
IsLeapMonth()	Liefert <code>True</code> zurück, wenn es sich bei dem übergebenen Monat um einen Schaltmonat handelt. Im gregorianischen Kalender wäre das der Februar. Übergeben werden Jahr und Monat als <code>Integer</code> -Werte.
IsLeapYear()	Liefert <code>True</code> zurück, wenn es sich bei dem angegebenen Jahr um ein Schaltjahr handelt. Das Jahr wird als <code>Integer</code> -Wert übergeben.
ToDateTime()	Liefert einen <code>DateTime</code> -Wert zurück. Hier können Sie Einzelangaben in einen <code>DateTime</code> -Wert umwandeln. Übergeben werden Jahr, Monat, Tag, Stunde, Minute, Sekunde und Millisekunde als <code>Integer</code> -Werte.
ToFourDigitYear()	Wandelt eine zweistellige Jahreszahl in eine vierstellige Jahreszahl um.

Tabelle 8.2: Die Methoden der Kalenderklassen (Fortsetzung)

Alle Methoden sind Instanzmethoden, da die jeweiligen Besonderheiten des benutzten Kalenders berücksichtigt werden müssen.

8.2.1 Umrechnen von Datumsangaben

Die verwendeten Kalender ermöglichen es nun, das Datum eines fremden Landes (bzw. für Moslems auch das Datum beispielsweise des muslimischen Kalenders) herauszufinden. Dazu wird nur ein `DateTime`-Wert benötigt, der den Methoden der Kalenderklasse übergeben wird, die verwendet werden soll.

In diesem Beispiel wird über eine `DateTimePicker`-Komponente ein Datum ausgewählt und dann in hebräisch und muslimisch angezeigt (nur die Datumswerte, natürlich nicht die Beschriftung). Der Quelltext ist verhältnismäßig einfach.

```
'=====
' CalendarExample.sln
' Copyright 2002 Frank Eller
' Umrechnung von Datumsangaben
'=====

Imports System.Globalization

#Region " Windows Form Designer generated code "

    'Der Code, der von VS.NET eingefügt wurde, wird
    'aus Übersichtsgründen nicht dargestellt.

#End Region

Private Sub CheckDates()

    ' --- Deklarationen
    Dim actDate As DateTime
    Dim hiCalendar As New HijriCalendar()
    Dim heCalendar As New HebrewCalendar()
    Dim day As Integer
    Dim month As Integer
    Dim year As Integer

    actDate = datePicker.Value

    ' --- Hebräischer Kalender
    month = heCalendar.GetMonth(actDate)
    day = heCalendar.GetDayOfMonth(actDate)
    year = heCalendar.GetYear(actDate)

    lblHebrewDate.Text = day.ToString() + " - " & _
```

```

        + month.ToString() + " - " + year.ToString()
lblHebrewDays.Text = heCalendar.GetDaysInMonth( _
    year, month).ToString()
If heCalendar.IsLeapYear(year) Then
    lblHebrewLeapYear.Text = "Ja"
Else
    lblHebrewLeapYear.Text = "Nein"
End If

' --- Hijri-Kalender
month = hiCalendar.GetMonth(actDate)
day = hiCalendar.GetDayOfMonth(actDate)
year = hiCalendar.GetYear(actDate)

lblHijriDate.Text = day.ToString() + " - " _
    + month.ToString() + " - " + year.ToString()
lblHijriDays.Text = hiCalendar.GetDaysInMonth( _
    year, month).ToString()
If hiCalendar.IsLeapYear(year) Then
    lblHijriLeapYear.Text = "Ja"
Else
    lblHijriLeapYear.Text = "Nein"
End If

End Sub

Private Sub datePicker_ValueChanged( _
    ByVal sender As Object, _
    ByVal e As System.EventArgs) _
    Handles datePicker.ValueChanged

    CheckDates()
End Sub

Private Sub btnExit_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnExit.Click

    Me.Close()
End Sub

Private Sub Form1_Load( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles MyBase.Load

    CheckDates()
End Sub
End Class

```

Listing 8.1: CalendarExample.exe

Einen Screenshot des fertigen Programms sehen Sie in Abbildung 8.1.

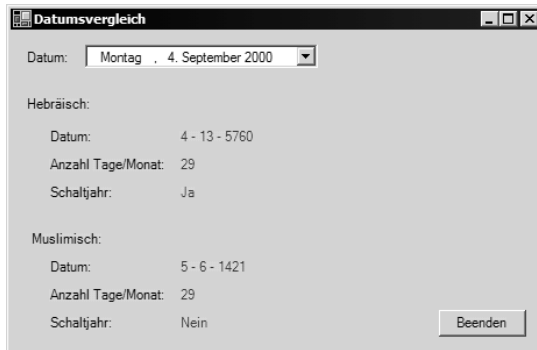


Abbildung 8.1: Datumsanzeige mit verschiedenen Kalendern [CalendarExample.sln]

8.3 Lokalisierung von Programmen

Um ein Programm auf ein bestimmtes Land einstellen zu können, muss dieses erst einmal identifiziert werden. Dazu gibt es zwei Möglichkeiten: über eine *LCID* (Local Country ID, Länderkennung) oder über einen String mit bestimmtem Aufbau, der sowohl Sprache als auch Landesinformationen enthält.

Die LCID besteht aus drei Teilen, nämlich einer primären und einer sekundären Sprachidentifikation sowie einer ID zur Sortierung. Repräsentiert wird dieser Wert durch eine Zahl, Deutsch z.B. durch 1031, US-Englisch durch 1033 usw. Diese Werte tauchen auch in Ihrem Dateisystem auf. Office XP beispielsweise legt für jede unterstützte Sprache ein Unterverzeichnis an, dessen Name diesem Wert entspricht. Darin befinden sich dann länderspezifische Dateien.

LCID

Eine weitaus besser lesbare Methode, das Land anzugeben, ist der String nach RFC1766. Diese Norm gibt vor, wie ein solcher länderspezifischer String aufgebaut sein muss. An erster Stelle steht die eigentliche Sprache, z.B. »de« für Deutsch oder »en« für Englisch, dann folgen ein Bindestrich und die Regionsinformation. Beispielsweise wird Deutsch auch in Österreich gesprochen, daher müssen diese Länder unterschieden werden. Der Wert für Standard-Deutsch (wie es in Deutschland gesprochen wird, wäre dann »de-DE«. Deutsch in Österreich entspräche »de-AT«.

Länder-Strings

**Ressourcen-
dateien**

Visual Studio macht sich diese Möglichkeiten in mehrerer Hinsicht zu Nutze. Wenn Sie ein Programm lokalisieren, d.h. unterschiedliche Ressourcendateien für die verschiedenen Sprachen anlegen, wird für jede Sprache ein Unterverzeichnis mit dem oben beschriebenen Aufbau erzeugt und die Ressourcendatei darin abgelegt.

Die Einstellung der Sprache bzw. des Landes in Visual Studio geschieht über die Klasse `CultureInfo`.

8.3.1 Die Klasse `CultureInfo`

`CultureInfo` enthält alle Daten bzgl. eines Landes, z.B. Formatierungsdaten, Zahlenformate, Datumsformate, den benutzten Kalender usw. Der Konstruktor der Klasse erwartet entweder einen `Integer`-Wert, der das Land repräsentiert, das eingestellt werden soll, oder einen String im RFC1766-Format. Der angesprochene `Integer`-Wert entspricht oben angegebener LCID.

Es existieren drei verschiedene Arten von Kulturinformationen:

- InvariantCulture** ► Die Eigenschaft `InvariantCulture` liefert ein `CultureInfo`-Objekt, das mit der Sprache Englisch assoziiert ist, ansonsten aber keine Informationen beinhaltet. Ein solches Objekt kann für Methoden verwendet werden, die keine besondere Kulturinformation benötigen, denen aber dennoch ein Objekt vom Typ `CultureInfo` übergeben werden muss.
- Neutrale Kultur** ► Eine neutrale Kultur ist ein `CultureInfo`-Objekt, bei dem nur die Sprache, aber keine spezifischen Informationen zugewiesen sind. Um ein neutrales `CultureInfo`-Objekt zu erzeugen, lassen Sie beim RFC1766-Code einfach die spezifische Landesinformation weg. »de« bezeichnet damit ein neutrales `CultureInfo`-Objekt, »de-DE« ein spezifisches.
- Spezifische Kultur** ► Spezifische `CultureInfo`-Objekte sind sowohl einer Sprache als auch einem Land bzw. einer Region zugewiesen. Die entsprechenden Ländercodes bestehen aus beiden Bestandteilen, z.B. »en-US« für US-Englisch.

`CultureInfo` besitzt sowohl statische (shared) als auch objektbezogene Eigenschaften und Methoden. Einige der wichtigsten sind in Tabelle 8.3 aufgeführt.

Eigenschaft	Bedeutung
CurrentCulture	Eine statische (shared) Eigenschaft. Sie liefert ein <code>CultureInfo</code> -Objekt mit den Einstellungen des aktuellen Threads.
CurrentUICulture	Ebenfalls statisch. Diese Eigenschaft liefert ein <code>CultureInfo</code> -Objekt mit den Einstellungen, die der Ressourcenmanager benutzt.
InstalledUICulture	Wiederum eine statische Eigenschaft. Sie liefert ein <code>CultureInfo</code> -Objekt mit den Einstellungen des aktuellen Systems.
InvariantCulture	Liefert ein <code>CultureInfo</code> -Objekt zurück, das sprachenunabhängig ist. Auch diese Eigenschaft ist statisch.
Calendar	Liefert ein <code>Calendar</code> -Objekt mit den Einstellungen des aktuellen Kalenders
CompareInfo	Liefert Informationen über die Art und Weise, auf die Strings verglichen werden
DateTimeFormat	In dieser Eigenschaft können Sie einstellen bzw. auslesen, auf welche Art und Weise Datum und Zeit formatiert werden sollen. Das zurückgelieferte Objekt ist vom Typ <code>DateTimeFormatInfo</code> .
DisplayName	Liefert die Bezeichnung des eingestellten Landes in der Sprache, auf die Visual Studio eingestellt ist
EnglishName	Liefert die Bezeichnung des eingestellten Landes in englischer Sprache
IsNeutralCulture	Liefert <code>True</code> , wenn es sich bei der eingestellten Sprache um eine landesunabhängige Sprache handelt
LCID	Liefert die LCID des aktuell eingestellten Landes
Name	Liefert den Namen des eingestellten Landes im RFC1766-Format
NumberFormat	In dieser Eigenschaft können Sie einstellen bzw. auslesen, auf welche Art Zahlen formatiert werden sollen. Das zurückgelieferte Objekt ist vom Typ <code>NumberFormatInfo</code> .
OptionalCalendars	Liefert eine Liste von Kalenderarten, die optional für dieses Land verwendet werden können
TextInfo	Liefert ein <code>TextInfo</code> -Objekt mit den Schreibeinstellungen für die eingestellte Sprache zurück

Tabelle 8.3: Eigenschaften von `CultureInfo`

Die Methoden von `CultureInfo` finden Sie in Tabelle 8.4.

Methode	Funktion
<code>Clone()</code>	Erstellt eine Kopie des aktuellen <code>CultureInfo</code> -Objekts
<code>CreateSpecificCulture()</code>	Erzeugt ein <code>CultureInfo</code> -Objekt mit den Einstellungen, die dem im Konstruktor angegebenen Namen entsprechen. Diese Methode ist statisch (<code>shared</code>).
<code>GetCultures()</code>	Liefert eine Liste unterstützter Ländereinstellungen in Form von <code>CultureInfo</code> -Objekten. Auch diese Methode ist statisch. Als Parameter wird ein Wert vom Typ <code>CultureTypes</code> verlangt, mit dem Sie angeben können, welcher Typ von Kulturliste zurückgeliefert werden soll.
<code>ReadOnly()</code>	Liefert das aktuelle <code>CultureInfo</code> -Objekt als unveränderliches (nur lesbares) Objekt zurück
<code>ToString()</code>	Liefert den Namen des aktuellen <code>CultureInfo</code> -Objekts zurück

Tabelle 8.4: Methoden von `CultureInfo`

Kulturen anzeigen

Über die Methode `GetCultures()` können die verfügbaren `CultureInfo`-Objekte angezeigt werden. Das folgende Beispiel verwendet diese Methode, um alle einstellbaren Länder aufzulisten. Benutzt wird für die Liste die Eigenschaft `DisplayName` des `CultureInfo`-Objekts. Der Formularaufbau besteht aus einer Combobox, in der eingestellt werden kann, welche Art von `CultureInfo`-Objekten angezeigt werden soll, und einer Listbox, in der diese Objekte dann mit vollem Namen angezeigt werden.

```
'=====
' CultureList.sln
' Copyright 2002 Frank Eller
' Zeigt Kulturen an
'=====

Imports System.Globalization

Public Class Form1
    Inherits System.Windows.Forms.Form

    #Region " Windows Form Designer generated code "
```

*'Der Code, der von VS.NET eingefügt wurde, wird
'aus Übersichtsgründen nicht dargestellt.*

#End Region

```
Private Sub FillListBox(ByVal list As ArrayList)
```

```
    lbxInformation.Items.Clear()
```

```
    lbxInformation.Items.AddRange(list.ToArray())
```

```
End Sub
```

```
Private Function GetCultureInfos( _  
    ByVal typ As CultureTypes)
```

```
    ' --- Deklarationen
```

```
    Dim result As New ArrayList()
```

```
    Dim cInfo As CultureInfo
```

```
    For Each cInfo In CultureInfo.GetCultures(typ)  
        result.Add(cInfo.DisplayName)
```

```
    Next
```

```
    Return result
```

```
End Function
```

```
Private Sub cbxCultureType_SelectedIndexChanged( _  
    ByVal sender As Object, _  
    ByVal e As System.EventArgs) _  
    Handles cbxCultureType.SelectedIndexChanged
```

```
    Select Case cbxCultureType.SelectedIndex
```

```
        Case 0 : FillListBox( _  
            GetCultureInfos(CultureTypes.AllCultures))
```

```
        Case 1 : FillListBox(GetCultureInfos( _  
            CultureTypes.InstalledWin32Cultures))
```

```
        Case 2 : FillListBox(GetCultureInfos( _  
            CultureTypes.NeutralCultures))
```

```
        Case 3 : FillListBox(GetCultureInfos( _  
            CultureTypes.SpecificCultures))
```

```
    End Select
```

```
End Sub
```

```
Private Sub Form1_Load(_  
    ByVal sender As System.Object, _  
    ByVal e As System.EventArgs) Handles MyBase.Load
```

```
    cbxCultureType.SelectedIndex = 0
```

```

End Sub

Private Sub btnExit_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnExit.Click

    Me.Close()

End Sub

End Class

```

Listing 8.2: CultureInfo-Objekte anzeigen [CultureList.sln]

Über `GetCultures()` werden die einzelnen `CultureInfo`-Objekte ermittelt und dann entsprechend in der Listbox angezeigt. Das Listing selbst sollte recht leicht verständlich sein. Abbildung 8.2 zeigt das Programm zur Laufzeit.



Abbildung 8.2: Alle einstellbaren Kulturen [CultureList.exe]

Infos über eine Kultur anzeigen

Nach Ermittlung eines `CultureInfo`-Objekts können natürlich auch dessen Einstellungen angezeigt werden. Das folgende Beispiel tut genau dies. In einer Combobox werden die spezifischen `CultureInfo`-Objekte angezeigt, eine Listbox dient zur Anzeige der Details. Auch dieses Beispiel ist recht leicht verständlich.

Der Aufbau des Formulars entspricht dem vorherigen Beispiel, diesmal sind allerdings die Rollen etwas anders verteilt. Die Combobox listet alle Kulturen auf, die Listbox detaillierte Eigenschaften. Wenn Sie z.B. Arabisch anzeigen lassen, werden Sie feststellen, dass sogar der Zeichensatz der Sprache berücksichtigt wird.

```
'=====
' CultureInformation.sln
' Copyright 2002 Frank Eller
' Zeigt Informationen über eine Kultur an
'=====

Imports System.Globalization

Public Class Form1
    Inherits System.Windows.Forms.Form

    #Region " Windows Form Designer generated code "

        'Der Code, der von VS.NET eingefügt wurde, wird
        'aus Übersichtsgründen nicht dargestellt.

    #End Region

    Private cultures As New ArrayList()

    Private Sub Form1_Load( _
        ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles MyBase.Load

        ' --- Deklarationen
        Dim country As CultureInfo

        ' --- Combobox füllen
        cbxCountry.Items.Clear()

        For Each country In CultureInfo.GetCultures( _
            CultureTypes.SpecificCultures)

            cbxCountry.Items.Add(country.DisplayName)
            cultures.Add(country)
        Next

        cbxCountry.SelectedIndex = 0

    End Sub

    Private Sub FillListBox(ByVal list As ArrayList)
```

```

lbxInformation.Items.Clear()
lbxInformation.Items.AddRange(list.ToArray())

```

End Sub

```

Private Function GetCountryInfo( _
    ByVal country As String) As ArrayList

    ' --- Deklarationen
    Dim cInfo As CultureInfo
    Dim dateInfo As DateTimeFormatInfo
    Dim result As New ArrayList()
    Dim helpStr As String

    ' --- Zuweisungen
    cInfo = New CultureInfo(country)
    dateInfo = cInfo.DateTimeFormat

    ' --- Auswertung
    result.Add("Informationen:")
    result.Add("-----")
    result.Add("Name: " + cInfo.Name)
    result.Add("DisplayName: " + cInfo.DisplayName)
    result.Add("Englisch: " + cInfo.EnglishName)
    result.Add(String.Empty)
    result.Add("Neutrale Kultur: " + _
        cInfo.IsNeutralCulture.ToString())
    result.Add(String.Empty)
    result.Add("LCID: " + cInfo.LCID.ToString())
    result.Add(String.Empty)
    result.Add("Datumspattern: " + _
        dateInfo.LongDatePattern)
    result.Add("Zeitpattern: " + _
        dateInfo.LongTimePattern)
    result.Add("Monate: ")
    For Each helpStr In dateInfo.MonthNames
        result.Add("    " + helpStr)
    Next
    result.Add("Tage: ")
    For Each helpStr In dateInfo.DayNames
        result.Add("    " + helpStr)
    Next

    Return result

```

End Function

```

Private Sub cbxCountry_SelectedIndexChanged( _
    ByVal sender As Object, _
    ByVal e As System.EventArgs) _
    Handles cbxCountry.SelectedIndexChanged

```

```

FillListBox( _
    GetCountryInfo( _
        CType(cultures(cbxCountry.SelectedIndex), _
            CultureInfo).ToString()))

End Sub

Private Sub btnExit_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnExit.Click

    Me.Close()
End Sub
End Class

```

Listing 8.3: Kulturinformationen anzeigen [CultureInformation.sln]

Das Ergebnis des Programms sehen Sie in Abbildung 8.3. Beachten Sie, dass in diesem Falle auch die Schriftart angepasst wurde – es handelt sich ja um Unicode.

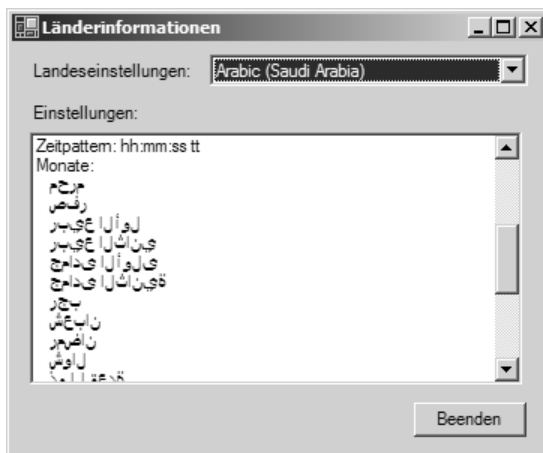


Abbildung 8.3: Einstellungen für Saudi-Arabien

8.3.2 Lokalisierung mit Visual Studio .NET

Wie bereits angesprochen bietet Visual Studio einige Hilfsmittel, die die Lokalisierung von Programmen erheblich erleichtern. Anhand einer kleineren Applikation soll aufgezeigt werden, wie schnell ein Programm an unterschiedliche Sprachen angepasst werden kann.

Einstellungen festlegen

Über eine Combobox soll die jeweilige Sprache ausgewählt werden, in diesem Fall eine der Sprachen Deutsch, Englisch oder Französisch. Weitere Sprachen sind natürlich möglich und ebenso schnell implementiert. Der Aufbau des Basisformulars ist recht einfach, Sie sehen ihn in Abbildung 8.4.



Abbildung 8.4: Das Basisformular in Deutsch

- Localizable** Nun muss das Formular zunächst einmal lokalisierbar gemacht werden. Im Eigenschaftsfenster finden Sie die Eigenschaft `Localizable`, die standardmäßig auf `False` eingestellt ist. Sobald Sie diese Eigenschaft auf `True` setzen, wird das Formular automatisch mehrsprachig.
- Language** Über die Eigenschaft `Language` kann nun die gewünschte Sprache ausgewählt werden. Augenscheinlich ändert sich nichts, im Hintergrund allerdings hat sich einiges getan. Visual Studio .NET hat nun für die neue Version eine Ressourcendatei erzeugt, die alle Einstellungen des Formulars beinhaltet. Berücksichtigt werden allerdings nur Einstellungen, die geändert wurden, unveränderte werden aus dem ursprünglichen Formular übernommen. Abbildung 8.5 zeigt das Eigenschaftsfenster mit Englisch als gewählter Sprache.

Das Gleiche können Sie nun auch noch einmal mit der Sprache Französisch machen. Wenn Sie in die Eigenschaft `Language` umschalten, werden Sie immer die gerade aktuelle Version sehen. Bauen Sie das Formular ein wenig um – Sie können z. B. die Sprache ändern, das Aussehen, Position der Komponenten, Größe der Komponenten usw. Visual Studio .NET sorgt automatisch dafür, dass alle Einstellungen in der jeweiligen Ressourcendatei gespeichert werden. Abbildung 8.6 zeigt die englische Version des Formulars, Abbildung 8.7 die französische. Allerdings ist Französisch nicht gerade meine Lieblingssprache, wie unschwer zu erkennen ist.

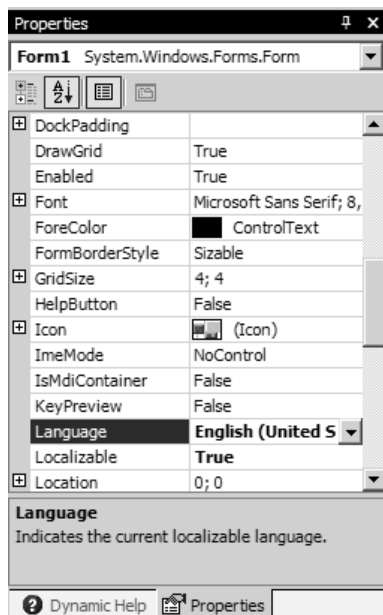


Abbildung 8.5: Das Eigenschaftsfenster mit Englisch als gewählter Sprache

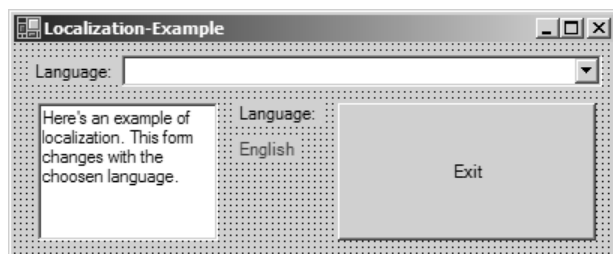


Abbildung 8.6: Die englische Version des Formulars

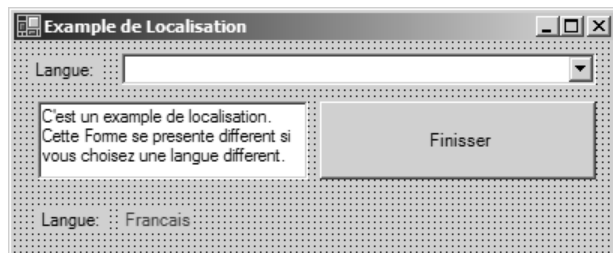


Abbildung 8.7: Die französische Version des Formulars

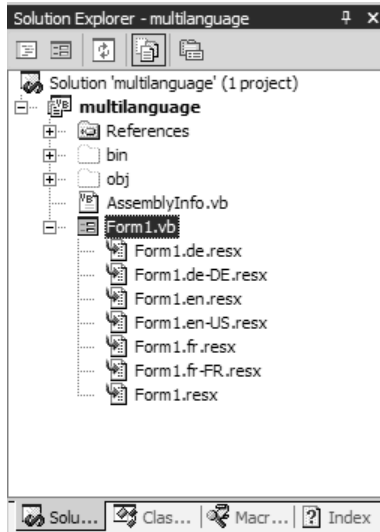


Abbildung 8.8: Die erzeugten Ressourcendateien

Damit ist das Projekt im Prinzip bereits lokalisiert – und das allein durch Änderung der Texte im Formular und durch Umstellen der Komponenten. Sie können auch Größenänderungen vornehmen, alles wird gespeichert. Wenn Sie im Projektfenster alle Dateien anzeigen lassen, sehen Sie als Unterpunkte des Formulars die erzeugten Ressourcendateien.

Anwendungen lokalisieren

Standardkultur Wenn Sie die Anwendung starten, wird sie immer als die deutsche Anwendung starten. Das liegt ganz einfach daran, dass jede Applikation standardmäßig die Kulturinformationen des Betriebssystems nutzt, im Fall dieses Beispiels also die deutsche Version. Wollen Sie eine der lokalisierten Versionen aufrufen, müssen Sie dafür sorgen, dass das entsprechende `CultureInfo`-Objekt benutzt wird.

Kultur zuweisen An dieser Stelle muss also in das Programm eingegriffen werden. Die Einstellung muss erfolgen, bevor die Komponenten initialisiert werden, da bei der Initialisierung bereits die lokalisierten Werte übergeben werden. Die einzige Möglichkeit, vorher Einfluss zu nehmen, ist die Methode `New()` des Formulars bzw. in C# der Konstruktor. Um die Anwendung mit der englischen Oberfläche laufen zu lassen, muss dem aktuellen Thread das entsprechende

CultureInfo-Objekt zugewiesen werden. Das geschieht über die statische (shared) Eigenschaft `CurrentThread` der Klasse `Thread`, die den aktuellen Thread repräsentiert. Die Zeile

```
Thread.CurrentThread.CurrentUICulture = _
    New CultureInfo("en-US")
```

stellt die englische Oberfläche ein. Danach präsentiert sich das Programm wie in Abbildung 8.9 gezeigt.

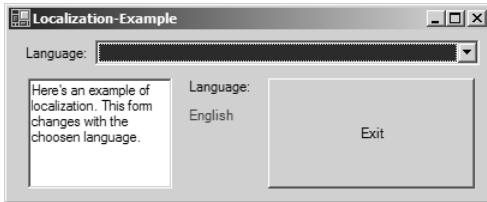


Abbildung 8.9: Das Programm zur Laufzeit in englischer Sprache

Gewünschte Sprache einstellen

Der erforderliche Neustart erzwingt ein Speichern der gewünschten Sprache, die dann beim Start eingestellt wird. Das nun folgende Beispiel – eigentlich immer noch das gleiche – speichert für jedes Land einen Wert, der im Konstruktor des Formulars ausgelesen und der Eigenschaft `CurrentUICulture` des aktuellen Threads zugewiesen wird. Der Ladevorgang erfolgt über einen `StreamReader`, das Speichern über einen `StreamWriter`, wobei eine bestehende Datei überschrieben wird. Beim Start muss weiterhin überprüft werden, ob die Datei existiert, da es ansonsten zu einer Exception kommt. Existiert die Datei nicht, wird Deutsch als Sprache eingestellt. Listing 8.4 zeigt das Listing des Programms.

CurrentUICulture

```
' =====
' Multilanguage.sln
' Copyright 2002 Frank Eller
' Zeigt ein lokalisiertes Formular
' =====

Imports System.IO
Imports System.Globalization
Imports System.Threading

Public Class Form1
    Inherits System.Windows.Forms.Form
```

```

#Region " Windows Form Designer generated code "

Public Sub New()
    MyBase.New()

    ' ---Jetzt Landeseinstellungen laden
    ' --- Deklarationen
    Dim sReader As StreamReader
    Dim countryValue As Integer = 0
    Dim currCulture As CultureInfo

    'Land laden und einstellen

    If File.Exists(fileName) Then
        sReader = New StreamReader(fileName)
        countryValue = _
            Convert.ToInt32(sReader.ReadLine())
        sReader.Close()
    End If

    Select Case countryValue
        Case 0 : currCulture = _
            CultureInfo.CreateSpecificCulture("de-DE")
        Case 1 : currCulture = _
            CultureInfo.CreateSpecificCulture("en-US")
        Case 2 : currCulture = _
            CultureInfo.CreateSpecificCulture("fr-FR")
    End Select

    Thread.CurrentThread.CurrentUICulture = _
        currCulture

    InitializeComponent()

End Sub

' Weitere automatisch generierte Methoden werden aus
' Übersichtsgründen nicht angezeigt.

#End Region

Private fileName As String = "info.txt"

Private Sub btnExit_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnExit.Click

    Me.Close()

```



```

End Sub

Private Sub cbxLanguage_SelectedIndexChanged( _
    ByVal sender As Object, _
    ByVal e As System.EventArgs) _
    Handles cbxLanguage.SelectedIndexChanged

    ' --- Deklarationen
    Dim sWriter As StreamWriter
    sWriter = New StreamWriter(fileName, False)

    sWriter.WriteLine(cbxLanguage.SelectedIndex)
    sWriter.Close()

End Sub

End Class

```

Listing 8.4: Einstellen der gewünschten Sprache [Multilanguage.sln]

In der Ereignisbehandlungsroutine `SelectedIndexChanged` von `cbxLanguage` wird der Sprachenwechsel vollzogen, indem der gewählte Wert in eine Datei geschrieben wird. Deren Name ist global festgelegt. Beim erneuten Start wird die entsprechende Sprache dann für den aktuellen Thread (den Haupt-Thread der Anwendung) eingestellt. Wichtig ist, dass dies vor der Initialisierung der Elemente des Formulars geschieht, da bei dieser Initialisierung die aktuell eingestellte Kultur verwendet wird.

8.4 Zugriff auf Ressourcendateien

Außer der Oberfläche müssen auch noch alle Texte lokalisiert werden, die innerhalb des Programms verwendet werden. Es bringt nichts, lediglich die Oberfläche anzupassen, wenn in diversen Dialogboxen nach wie vor die ursprüngliche Sprache verwendet wird. Die Lösung für dieses Problem sind wie schon in früheren Tagen die so genannten Ressourcendateien. Das Format ist jedoch ein vollkommen anderes, und auch der Zugriff funktioniert auf andere Art und Weise, nämlich über die Klasse `ResourceManager`.

8.4.1 Zusammenhänge

Ressourcendateien sind bereits seit Windows 3.1 bekannt. Pro Programm wurde eine Ressourcendatei eingebunden, in der in der Regel alle Texte ausgelagert wurden, die man in dem Programm

**Ressourcen-
dateien unter
Windows 3.1**

benötigte. Sollte das Programm nun auf ein anderes Land angepasst werden, musste man lediglich die Ressourcendatei übersetzen und das Programm neu kompilieren – diesmal mit der neuen Ressourcendatei – oder man arbeitete mit einem Offset, nummerierte die enthaltenen Zeichenketten und legte innerhalb des Programms je nach eingestelltem Land den Offset neu fest. Damit wurden immer die richtigen Strings geladen.

Richtig zufriedenstellend ist diese Vorgehensweise allerdings nicht. Auch die Größe einer Ressourcendatei ist begrenzt und je mehr Sprachen man darin unterbringen muss, desto eher stößt man an die Grenzen. Unter .NET wird nun anders vorgegangen. Weg von der einzelnen Ressourcendatei hin zu lokalisierten Dateien, die nach einem bestimmten Schema abgelegt werden und auf die das Programm je nach eingestellter Sprache – was wieder über die Kultur festgelegt wird – zugreift.

Namensbelegung von Ressourcen- dateien

Die Namensgebung der Ressourcendateien bestimmt, für welche Sprache die Ressourcen enthalten sind. Verwendet wird hier wieder die RFC1766; eine Ressourcen-Assembly für die Sprache US-Englisch würde also folgenden Namen tragen:

`<Applikationsname>.en.US.resx`

Für die Sprache Deutsch (Deutschland) wäre die Bezeichnung

`<Applikationsname>.de.DE.resx`

Satellite-Assembly

Bei der Kompilierung wird für jede Sprache eine so genannte *Satellite-Assembly* erzeugt, die in einem der Sprache und der Region entsprechenden Unterverzeichnis abgelegt wird.

Eine bestimmte Assembly, deren Name lediglich dem Applikationsnamen entspricht, enthält die Standard-Sprache. Das bedeutet, dass nicht alle Texte des Programms übersetzt werden müssen, sondern nur diejenigen, die auch wirklich anders sind. Ist eine Sprache in einem Programm eingestellt, so sucht das .NET-Framework die richtige Satellite-Assembly heraus und sucht darin nach der passenden Entsprechung. Wird diese nicht gefunden, so wird die Standard-Assembly verwendet, d.h. es tritt keine Exception auf, sondern es wird der Wert in der Ressourcendatei der Standard-Sprache verwendet. In dieser müssen selbstverständlich alle Werte enthalten sein – da sie aber die Basis für lokalisierte Assemblies bildet, versteht sich das von selbst.

8.4.2 Die Klasse ResourceManager

Zum Zugriff auf eine Assembly wird die Klasse `ResourceManager` verwendet, die im Namespace `System.Resources` deklariert ist. Über eine der Methoden `GetObject()` oder `GetString()` können die Ressourcen ausgelesen werden.

Die Verwendung der `ResourceManager`-Klasse gestaltet sich recht einfach. Es wird automatisch die in `CurrentUICulture` eingestellte Sprache verwendet, um auf die Ressourcendateien zuzugreifen. Um den passenden String zu erhalten, wird der Methode `GetString` nur der in der Ressourcendatei festgelegte Name für einen bestimmten String übergeben.

Die Klasse `ResourceManager` dient nicht nur dem Zugriff auf Ressourcen, sie verbraucht auch recht viele Ressourcen. Aus diesem Grund ist es sinnvoll, eine eigene Klasse zu erstellen, die ein `ResourceManager`-Objekt als statisches Element beinhaltet. Damit wird dann nur ein `ResourceManager`-Objekt für die gesamte Applikation benötigt.



8.4.3 Eine Anwendung mit lokalisierten Strings

Die Beispielanwendung besteht nur aus einem kleinen Formular mit zwei Buttons, von denen einer dem Beenden dient und der andere der Anzeige einer MessageBox. Über eine Checkbox wird eingestellt, ob die MessageBox in Englisch oder in Deutsch angezeigt werden soll. Die entsprechenden Ressourcendateien des Projekts enthalten lediglich einen String mit dem Bezeichner `Message`. Listing 8.5 zeigt das Listing des Programms.

**Lokalisierte
Anwendung**

```
'=====
' ResourceMan.sln
' Copyright 2002 Frank Eller
' Anwendung von Ressourcendateien
'=====

Imports System.Globalization
Imports System.Resources
Imports System.Threading

Public Class Form1
    Inherits System.Windows.Forms.Form

    #Region " Windows Form Designer generated code "

    'Der Code, der von VS.NET eingefügt wurde, wird
    'aus Übersichtsgründen nicht dargestellt.
```

```

#End Region

Private Sub btnShowMsg_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnShowMsg.Click

    ' --- Deklarationen
    Dim aManager As ResourceManager
    Dim aCulture As CultureInfo
    Dim msgString As String

    ' --- Checkbox überprüfen
    If cbxEnglish.Checked Then
        aCulture = New CultureInfo("en-US")
    Else
        aCulture = New CultureInfo("de-DE")
    End If

    ' --- Korrekte Kultur zuweisen
    Thread.CurrentThread.CurrentUICulture = aCulture

    ' --- Ressourcenmanager erzeugen
    aManager = New ResourceManager( _
        "ResourceMan.ResourceMan", _
        [Assembly].GetExecutingAssembly())

    ' --- Zugriff auf die Ressourcendatei
    msgString = aManager.GetString("Message")

    ' --- MessageBox anzeigen
    MessageBox.Show(msgString)
End Sub

Private Sub btnExit_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnExit.Click

    Me.Close()
End Sub

```

Listing 8.5: Lokalisierte Ausgabe über eine MessageBox

Einen Screenshot des Programms zeigt Abbildung 8.10.



Abbildung 8.10: Eine MessageBox mit einem lokalisierten String

9 Der Namespace System.Xml

XML spielt als Datenformat eine zentrale Rolle im .NET-Framework. Microsoft hat den Nutzen dieses universellen Datenformats erkannt und es zu einem integralen Bestandteil gemacht. Dementsprechend umfangreich fällt auch die Unterstützung aus.

Unter `System.Xml` gibt es noch weitere Unter-Namespace, die Klassen für die folgenden Funktionalitäten beinhalten:

- ▶ `System.Xml.Schema` für die Unterstützung von XML-Schema nach den Vorgaben des W3Cs
- ▶ `System.Xml.Serialization`, der bereits behandelt wurde
- ▶ `System.Xml.XPath` für die Unterstützung von XPath-Ausdrücken zum Durchsuchen von XML-Dateien
- ▶ `System.Xml.Xsl` für die Unterstützung der Konvertierung von XML-Dateien durch XSL-Stylesheets

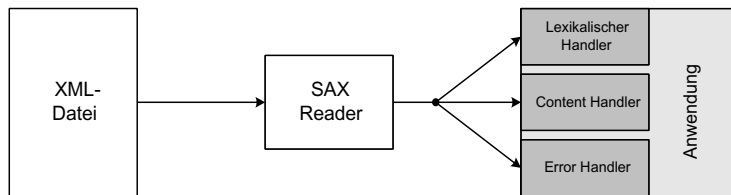
Die unterstützten Standards entsprechen den Vorgaben des W3Cs. Unterstützt werden unter anderem

- ▶ das *Document Object Model* (DOM) über die Klasse `XmlDocument`
- ▶ XPath-Ausdrücke über die Klassen `XmlNavigator` und `XPathIterator`
- ▶ XSL/T – Transformationen über die Klasse `XslTransform`
- ▶ XML-Schema über die Klasse `XmlSchema`

In der Realität arbeiten einige Klassen und Schnittstellen zusammen, um die nötige Funktionalität zu gewährleisten. In diesem Kapitel finden Sie die wichtigsten XML-Klassen und ihre Verwendung.

9.1 XmlReader/XmlWriter

Push-Modell Das Lesen einer XML-Datei wird in anderen Programmiersprachen oftmals mit Hilfe der *Simple API for XML* (SAX) durchgeführt. SAX implementiert ein so genanntes *Push-Modell*, bei dem Daten aus der Datei gelesen und dann innerhalb der Anwendung mit Hilfe von Ereignissen ausgewertet werden. Der SAX-Reader liest jedes Element der XML-Datei und löst dann, abhängig von der Art des Elements, ein Ereignis aus, auf das die Anwendung wiederum reagieren muss. Das ist zwar eine logische Vorgehensweise, bringt aber bei komplexen XML-Dokumenten den Nachteil mit sich, dass die auswertende Anwendung schnell zu einer komplexen Zustandsmaschine ausartet, deren Wartung nicht mehr ganz so einfach ist. Abbildung 9.1 zeigt das Lesemodell eines SAX-Readers.



© 2001 Frank Eller

Abbildung 9.1: Das Lesemodell von SAX

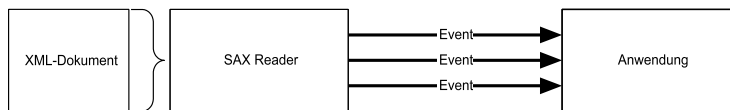
Pull-Modell Microsoft hat den Ansatz des SAX-Modells aufgegriffen, ihn aber in einem entscheidenden Punkt verändert. An die Stelle des Push-Modells von SAX tritt nun das *Pull-Modell* des .NET-Frameworks. Anstatt die vom Reader gelieferten Daten durch Ereignisse auszuwerten, holt sich der .NET-Reader die Daten und erlaubt deren Auswertung innerhalb einer Schleife. Damit sind die Daten wesentlich leichter auszuwerten und der eigentliche Auswertungsalgorithmus verliert seine Komplexität. Abbildung 9.2 zeigt den Vergleich der unterschiedlichen Modelle.

Auswertung im .NET-Framework

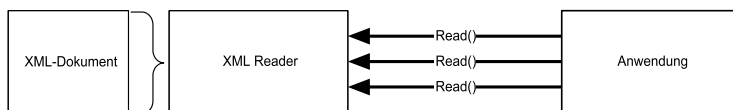
Die Auswertung der Daten erfolgt abhängig vom Typ des gelesenen Elements. Der große Vorteil dabei ist, dass Sie unwichtige Elemente überspringen können – mit SAX ist das nicht möglich, da werden Ihnen grundsätzlich alle Daten geliefert. Der .NET-Reader liefert den Typ des Elements und dessen Inhalt und ermöglicht weiterhin den Zugriff auf weitere Daten, z.B. Attribute eines Knotens. Content Handler und Lexikalischer Handler des SAX-

Modells sind damit innerhalb der Schleife zusammengefasst, die Behandlung der Daten wesentlich vereinfacht. Auftretende Fehler (z.B. bei der Validierung eines Dokuments) werden weiterhin durch Ereignisse abgedeckt, Exceptions durch einen Try...Catch-Block abgefangen.

SAX (Push-Modell)



.NET XML Reader (Pull-Modell)



(C) Holger@Schwichtenberg.de

Abbildung 9.2: Unterschiede zwischen den Modellen

In der Praxis bietet das .NET-Framework sowohl einen `XmlReader`, der auf dem Pull-Modell basiert, als auch einen `XmlWriter`, der es ermöglicht, vorhandene Daten seriell in eine XML-Datei zu schreiben. Beide Klassen, `XmlReader` und `XmlWriter`, sind abstrakte Klassen, d.h. von ihnen muss erst eine neue Klasse abgeleitet werden, in der die vordefinierten Funktionen implementiert sind.

Am häufigsten wird vermutlich der `XmlTextReader` eingesetzt werden, mit dem es möglich ist, Daten aus einer XML-Datei oder einem `Stream`-Objekt zu lesen. Das passende Gegenstück dazu ist der `XmlNodeReader`, der seine Daten aus einem DOM-Dokument bezieht (also aus `XmlDocument`). Der Unterschied besteht eigentlich nur in der Art, wie die Daten geliefert werden, nämlich einmal in Textform über eine Datei und einmal als Objekte vom Typ `XmlNode`.

**XmlTextReader
und XmlNode
Reader**

Eine Sonderstellung nimmt die Klasse `XmlValidatingReader` ein, mit der es möglich ist, ein Dokument auf Wohlgeformtheit (engl. *well-formedness*) zu überprüfen bzw. sicherzustellen, dass das Dokument der verwendeten DTD bzw. dem verwendeten Schema entspricht. Der `XmlValidatingReader` implementiert einen `EventHandler` namens `ValidatingEventHandler`, dem Sie das Ereignis zuweisen können, in dem ein etwaiger auftretender Fehler behandelt werden kann.

**XmlValidating
Reader**

XmlTextWriter Zum Schreiben bestehender Daten dient der `XmlTextWriter`, der wie auch die `XmlReader`-Implementationen einen Nur-vorwärts-Cursor besitzt. Ebenso, wie `XmlReader` nur von vorne nach hinten lesen kann, kann der `XmlTextWriter` auch nur von vorne nach hinten schreiben – die Daten, die geschrieben werden sollen, müssen also bekannt sein. Der `XmlTextWriter` schreibt sequenziell; eine Hierarchie, wie sie in XML-Dokumenten üblich ist, wird durch Verschachtelung der einzelnen Elemente erreicht. Wenn also zweimal direkt nacheinander der Anfang eines Elements geschrieben wird, ist das zweite begonnene Element hierarchisch unterhalb des ersten angeordnet. Im Beispiel wird dies später deutlicher.

Abbildung 9.3 zeigt die Hierarchie der `XmlReader/XmlWriter`-Klassen im .NET-Framework.

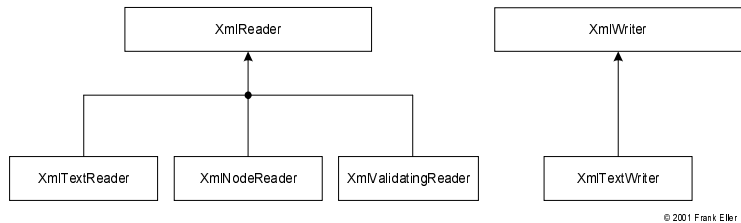


Abbildung 9.3: Hierarchie der Reader-/Writer-Klassen

9.1.1 XML-Daten aus Dateien lesen

Ein einfacher Reader

Serialisierung mit XmlTextReader

Das folgende Beispiel zeigt, wie Daten mit Hilfe eines `XmlTextReader`-Objekts aus einer Datei ausgelesen werden können. Das Reader-Objekt liest alle Knoten und gibt dann die relevanten auf der Konsole aus. Das Beispiel beschränkt sich auf Elemente und enthaltenen Text und berücksichtigt keine Attribute. Das Listing zeigt die komplette Konsolenanwendung mit Visual Basic .NET.

```

' =====
' XmlSimpleReader.sln
' Copyright 2002 Frank Eller
' Auslesen von XML-Dateien
' =====

Imports Microsoft.VisualBasic
Imports System.Xml
Imports System.IO
  
```

Module Module1

```

Private Sub SerializeXml(ByVal fileName As String)
    ' --- Reader deklarieren
    Dim reader As New XmlTextReader(fileName)

    ' --- Lesen
    While reader.Read()
        Select Case reader.NodeType
            Case XmlNodeType.Element
                Console.WriteLine "[" + reader.Name + "]")
            Case XmlNodeType.Text
                Console.WriteLine("  " + reader.Value)
        End Select
    End While

    ' --- Reader schließen
    reader.Close()
End Sub

Sub Main()
    ' --- Dateinamen als Argument einlesen
    Dim parameters As String = Command()
    Dim args() As String = parameters.Split( _
        " ".ToCharArray())
    Dim fn As String = args(0)

    ' --- Existenzprüfung
    If Not File.Exists(fn) Then
        Console.WriteLine("Die Datei existiert nicht")
    Else
        SerializeXml(fn)
    End If
End Sub

```

End Module

Listing 9.1: Serialisieren von XML-Daten [XmlSimpleReader.sln]

Das Lesen der Daten erfolgt über die Methode `Read()`. Diese liest jedes Element der XML-Datei – auch die nicht relevanten Elemente – und liefert den Inhalt im `XmlTextReader`-Objekt zurück. Die Eigenschaft `NodeType` liefert die Art des gelesenen Elements in Form eines `XmlNodeType`-Werts zurück. `XmlNodeType` ist ein Aufzählungstyp, womit sich auch erklärt, warum dieser Typ in einer `Select Case`-Anweisung verwendet werden kann.

Angewandt auf ein Beispieldokument sieht die Ausgabe dann folgendermaßen aus:

```
[BookList]
[Book]
[Title]
    C# lernen
[Autor]
    Frank Eller
[ISBN]
    3-8273-1784-3
[Price]
    24,95 _
[Book]
[Title]
    Windows Scripting
[Autor]
    Holger Schwichtenberg
[ISBN]
    3-8273-1843-2
[Price]
    59,95 _
```

Das Beispieldokument sehen Sie in Abbildung 9.4.

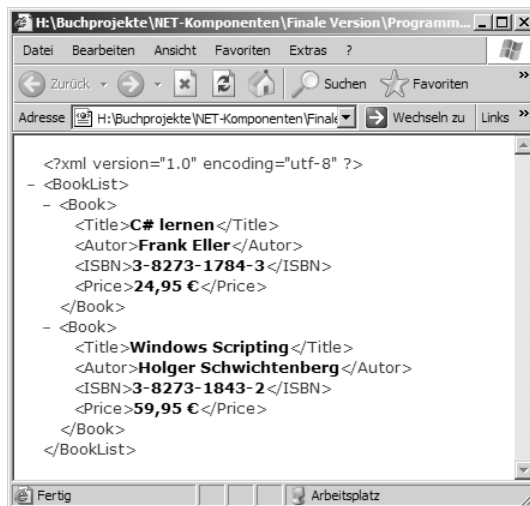


Abbildung 9.4: Das Beispieldokument im Internet Explorer

MoveToNext- Attribute()

`Read()` liest keine Attribute, nur ganze Elemente. Wenn Sie auch die Attribute eines Elements auswerten wollen, müssen Sie die Methoden `MoveToFirstAttribute()` oder `MoveToNextAttribute()` anwenden. Üblicherweise genügt Letztere, um zu den einzelnen

Attributen zu springen. Das Reader-Objekt enthält dann in der Eigenschaft `Name` die Bezeichnung des Attributs und in der Eigenschaft `Value` den dazugehörigen Wert.

Schließlich implementiert `XmlTextReader` noch eine Methode namens `MoveToContent()`, die dazu dient, unwichtige Elemente zu überspringen. Wenn Sie `MoveToContent()` aufrufen, springt der Reader zum nächsten Element, das einen Inhalt bereitstellt, und überspringt alles andere. Eine Ausnahme besteht dann, wenn der Reader bereits auf einem Attribut eines relevanten Knotens steht, in diesem Fall springt er zurück zu dem Element, das das Attribut enthält. Wird kein inhaltlich relevantes Element mehr gefunden, springt `MoveToContent()` zum Ende der Datei und liefert `XmlNodeType.None` zurück.

MoveToContent()

XML in Treeview einlesen

Das folgende Beispiel liest den Inhalt einer XML-Datei in eine Treeview ein und stellt so die Struktur der XML-Datei dar. In diesem Beispiel werden auch die Attribute berücksichtigt und entsprechend dargestellt.

Das Formular besteht aus einer `TreeView`-Komponente mit Namen `xmlTreeView`, einer `RichTextBox` mit Namen `rtfXmlFile`, einem Dialog zum Öffnen der Datei und einem Button, in dessen Ereignisbehandlungsroutine die Funktionalität programmiert ist.

```
'=====
' XmlTree.sln
' Copyright 2002 Frank Eller
' Einlesen von XML in eine Treeview
'=====
```

```
Imports System.Xml
```

```
Public Class Form1
```

```
    Inherits System.Windows.Forms.Form
```

```
#Region " Windows Form Designer generated code "
```

```
    'Der Code, der von VS.NET eingefügt wurde, wird
    'aus Übersichtsgründen nicht dargestellt.
```

```
#End Region
```

```
Private Sub FillTreeView(ByVal fileName As String)
```

```
    ' --- Deklarationen
```

```

Dim newPrevNode As Boolean
Dim actnode As TreeNode = Nothing
Dim xmlReader As New XmlTextReader(fileName)

' --- Treeview leeren
xmlTreeView.Nodes.Clear()
xmlTreeView.Nodes.Add(fileName)

' --- Der aktuelle Knoten, zu dem
' --- Unterknoten hinzugefügt werden
Dim prevnode As TreeNode = xmlTreeView.Nodes(0)

' --- Keine Anzeige während Update
xmlTreeView.BeginUpdate()

' --- Nodes lesen und hinzufügen
While xmlReader.Read()
Select Case xmlReader.NodeType

' --- Element-Node mit Attributen hinzufügen
Case XmlNodeType.Element
actnode = New TreeNode(xmlReader.Name)

' --- Wenn Unterknoten vorhanden, wird
' --- dieser Knoten aktuell
newPrevNode = Not xmlReader.IsEmptyElement

' --- Attribute hinzufügen
While xmlReader.MoveToNextAttribute
Dim newNode As New TreeNode(xmlReader.Name)
newNode.Nodes.Add(New TreeNode(>< + _
xmlReader.Value + ><))
actnode.Nodes.Add(newNode)
End While

' --- Neuen Node hinzufügen
prevnode.Nodes.Add(actnode)

' --- Aktuellen Knoten festlegen
If newPrevNode Then
prevnode = actnode
End If

Case XmlNodeType.EndElement
' --- Aktueller Knoten ist vorheriger Knoten
prevnode = prevnode.Parent

Case XmlNodeType.Text
' --- Knotentext anzeigen
Dim nodetext As String = xmlReader.Value
If nodetext.Length > 20 Then

```

```

nodetext = nodetext.Substring(0, 17) + »...«
End If
prevnode.Nodes.Add(nodetext)
End Select
End While

' --- Treeview-Inhalt anzeigen
xmlTreeView.EndUpdate()
xmlTreeView.ExpandAll()
xmlReader.Close()
End Sub

Private Sub btnLoad_Click( _
ByVal sender As System.Object, _
ByVal e As System.EventArgs) _
Handles btnLoad.Click

If dlgOpen.ShowDialog() = DialogResult.OK Then
rtfXmlFile.LoadFile(dlgOpen.FileName, _
RichTextBoxStreamType.PlainText)
FillTreeView(dlgOpen.FileName)
End If
End Sub

Private Sub btnExit_Click( _
ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles btnExit.Click

Me.Close()
End Sub
End Class

```

Listing 9.2 zeigt die relevanten Teile des Programms.

```

'=====
' XmlTree.sln
' Copyright 2002 Frank Eller
' Einlesen von XML In eine Treeview
'=====

Imports System.Xml

Public Class Form1
    Inherits System.Windows.Forms.Form

    #Region " Windows Form Designer generated code "

        'Der Code, der von VS.NET eingefügt wurde, wird
        'aus Übersichtsgründen nicht dargestellt.

    #End Region

```

```

Private Sub FillTreeView(ByVal fileName As String)
    ' --- Deklarationen
    Dim newPrevNode As Boolean
    Dim actnode As TreeNode = Nothing
    Dim xmlReader As New XmlTextReader(fileName)

    ' --- Treeview leeren
    xmlTreeView.Nodes.Clear()
    xmlTreeView.Nodes.Add(fileName)

    ' --- Der aktuelle Knoten, zu dem
    ' --- Unterknoten hinzugefügt werden
    Dim prevnode As TreeNode = xmlTreeView.Nodes(0)

    ' --- Keine Anzeige während Update
    xmlTreeView.BeginUpdate()

    ' --- Nodes lesen und hinzufügen
    While xmlReader.Read()
        Select Case xmlReader.NodeType

            ' --- Element-Node mit Attributen hinzufügen
            Case XmlNodeType.Element
                actnode = New TreeNode(xmlReader.Name)

                ' --- Wenn Unterknoten vorhanden, wird
                ' --- dieser Knoten aktuell
                newPrevNode = Not xmlReader.IsEmptyElement

                ' --- Attribute hinzufügen
                While xmlReader.MoveToNextAttribute
                    Dim newNode As New TreeNode(xmlReader.Name)
                    newNode.Nodes.Add(New TreeNode("[ " + _
                        xmlReader.Value + "]"))
                    actnode.Nodes.Add(newNode)
                End While

                ' --- Neuen Node hinzufügen
                prevnode.Nodes.Add(actnode)

                ' --- Aktuellen Knoten festlegen
                If newPrevNode Then
                    prevnode = actnode
                End If

            Case XmlNodeType.EndElement
                ' --- Aktueller Knoten ist vorheriger Knoten
                prevnode = prevnode.Parent

            Case XmlNodeType.Text
                ' --- Knotentext anzeigen

```



```

        Dim nodetext As String = xmlReader.Value
        If nodetext.Length > 20 Then
            nodetext = nodetext.Substring(0, 17) + "..."
        End If
        prevnode.Nodes.Add(nodetext)
    End Select
End While

' --- TreeView-Inhalt anzeigen
xmlTreeView.EndUpdate()
xmlTreeView.ExpandAll()
xmlReader.Close()
End Sub

Private Sub btnLoad_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnLoad.Click

    If dlgOpen.ShowDialog() = DialogResult.OK Then
        rtfXmlFile.LoadFile(dlgOpen.FileName, _
            RichTextBoxStreamType.PlainText)
        FillTreeView(dlgOpen.FileName)
    End If
End Sub

Private Sub btnExit_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnExit.Click

    Me.Close()
End Sub
End Class

```

Listing 9.2: XmlTree.exe

Auch wenn es auf den ersten Blick etwas komplex aussieht, so wird bei näherer Betrachtung doch klar, dass die eigentliche Behandlung der Inhalte relativ einfach vonstatten geht. Ein wenig Komplexität kommt natürlich durch die Verwendung einer *TreeView*-Komponente mit hinein, aber grundsätzlich sollte der Quelltext verhältnismäßig leicht verständlich sein.

Der Dateiname dient uns als Basisknoten der *TreeView*-Komponente; an diesen werden alle gefundenen Werte angehängt. Mittels einer *While*-Schleife wird die XML-Datei durchwandert, wobei in diesem Fall auf die Methode *MoveToContent()* verzichtet wurde. Wenn es sich bei dem Knoten um ein Element handelt, benutzen wir eine weitere *While*-Schleife, um die enthaltenen Attribute auszuwerten und ebenfalls in die *TreeView*-Komponente zu schreiben.

Die Variable `newPrevNode` wird für die Festlegung des Knotens, an den aktuell angefügt werden soll, benutzt. Dieser ist in der Variablen `prevNode` gespeichert. Wenn das aktuelle Element nicht leer ist, also entweder Text oder Unterelemente enthält, müssen die nächsten Knoten zu dem Knoten des Elements hinzugefügt werden, um die Hierarchie korrekt darzustellen.

Parent Entsprechend wird beim Erreichen des schließenden Tags eines Elements ein Schritt in der Hierarchie nach oben gegangen. Dazu besitzt jeder Knoten vom Typ `XmlNode` eine Eigenschaft `Parent`, die den übergeordneten Knoten referenziert.

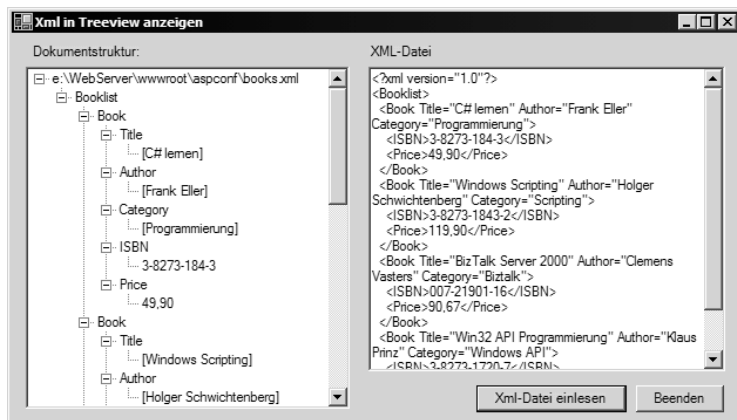


Abbildung 9.5: Die Bildschirmausgabe von `XmlTree` (`XmlTree.exe`)

XmlConvert

Konvertieren von Datentypen

In beiden Beispielen wurden bisher lediglich Daten in Form von Strings ausgelesen. Die Art und Weise des Auslesens ist immer die gleiche, allerdings ist es möglich, die gelesenen Daten einfach in einen anderen Datentyp – nämlich den den Daten entsprechenden Datentyp – zu konvertieren. Statt der Klasse `Convert`, die üblicherweise für solche Konvertierungen benutzt wird, gibt es für den XML-Bereich eine Klasse `XmlConvert`, die das Gleiche mit XML-Daten tut. Der Aufruf dieser Klasse entspricht dem Aufruf von `Convert`, sie beinhaltet also statische Methoden.

XmlConvert

Der Grund für die Verwendung von `XmlConvert` besteht darin, dass Datentypen in XML-Dateien den Datentypen entsprechen, die das W3C für XML-Schema vorgesehen hat. Diese entsprechen nicht zwangsläufig den Datentypen, die im .NET-Framework festgelegt

sind. Um also betriebssystemunabhängig zu bleiben, verwenden Sie zur Konvertierung `XmlConvert`.

Gleiches gilt für die Bezeichner von Attributen und Elementen in XML. Diese folgen der vom W3C festgelegten Spezifikation (XML 1.0) und sind somit im Bereich der Zeichen limitiert. Im .NET-Framework wird immer mit Unicode gearbeitet, daher ist es durchaus möglich, dass Sie Zeichen oder Namen, die Unicode-Zeichen beinhalten, konvertieren müssen. Dazu dienen z.B. die statischen Methoden `EncodeName()` oder `DecodeName()` von `XmlConvert`.

**EncodeName(),
DecodeName()**

Ein Beispiel hierzu wäre, wenn Sie ein XML-Tag einfügen wollen, das ein Leerzeichen beinhaltet. Das ist in XML innerhalb eines Tag-Bezeichners nicht erlaubt. Sie können aber `EncodeName()` verwenden, um aus dem Leerzeichen die numerische Entsprechung zu machen, und das Auslesen dann mit `DecodeName()` erledigen.

Ein Beispiel hierfür:

```
XmlConvert.EncodeName("Erste Adresse")
```

würde liefern

```
Erste_0x0020_Adresse
```

9.1.2 XML validieren

Mit dem `XmlTextReader` können Sie bereits eine einfache Validierung der Datei, die gelesen wird, durchführen. `XmlTextReader` kontrolliert beim Lesen, ob die Datei wohlgeformt ist, d.h. ob alle beginnenden und schließenden Tags an der richtigen Stelle stehen und ob das Ganze korrekt geschachtelt ist. Er kontrolliert auch, ob eine etwaige DTD, die angegeben sein könnte, well-formed ist. Was er nicht kontrolliert ist, ob das Dokument der DTD entspricht.

**Validieren mit
XmlTextReader**

Auch handelt es sich nicht um eine echte Validierung. Vielmehr wird eine `Exception` ausgelöst, wenn der `XmlTextReader` feststellt, dass das Dokument nicht der XML-Spezifikation entspricht. Diese `Exception` namens `XmlException` kann dann abgefangen und eine entsprechende Fehlermeldung ausgegeben werden.

XmlException

Natürlich ist das nicht befriedigend, denn es wird ja immer nur der erste Fehler angezeigt. Wenn dieser korrigiert ist und das Dokument weitere Fehler enthält, tritt die `Exception` wieder auf. Dass die Eigenschaft `LineNumber` die Nummer der Zeile enthält, in der der Fehler aufgetreten ist, ist nur ein schwacher Trost.

LineNumber

XmlValidatingReader

Es gibt aber eine andere Klasse, die ebenfalls von `XmlReader` abgeleitet ist und diese Möglichkeiten der Validierung beinhaltet. Die Klasse `XmlValidatingReader` implementiert einen EventHandler namens `ValidationEventHandler`. Diesem kann dann eine Methode zugeordnet werden, in der z.B. ausgegeben werden kann, welcher Fehler aufgetreten ist und an welcher Stelle. Der Vorteil des `XmlValidatingReader` ist, dass er nicht stehen bleibt, sondern weiterliest. Alle Unterelemente des fehlerhaften Elements werden allerdings nicht weiter beachtet, stattdessen wird beim nächsten Element weitergemacht. Somit wird für jedes Element, das möglicherweise auch mehrere Fehler enthält, immer nur der erste Fehler zurückgeliefert.

Einfache Validierung

Das Beispiel in Listing 9.3 zeigt eine einfache Validierung. Es wird hierbei nur kontrolliert, ob das Dokument wohlgeformt ist, also nicht, wie viele Fehler enthalten sind und wo sie sich verstecken.

```
'=====
' XmlValidation.sln
' Copyright 2002 Frank Eller
' Einfache Validierung von XML-Dateien
'=====

Imports System.Xml
Imports System.Xml.Schema
Imports System.IO

Public Class Form1
    Inherits System.Windows.Forms.Form

    #Region " Windows Form Designer generated code "

        'Der Code, der von VS.NET eingefügt wurde, wird
        'aus Übersichtsgründen nicht dargestellt.

    #End Region

    Dim doc_ok As Boolean = True

    Private Sub ValidatingEvent(ByVal sender As Object, _
                                ByVal e As ValidationEventArgs)
        Me.doc_ok = False
    End Sub

    Private Sub btnLoadFile_Click( _
        ByVal sender As System.Object, _
```

```

ByVal e As System.EventArgs) _
Handles btnLoadFile.Click

If dlgOpenFile.ShowDialog() = DialogResult.OK Then
    tbxXmlFile.Text = dlgOpenFile.FileName
End If
End Sub

Private Sub btnValidate_Click(_
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnValidate.Click

    Dim xmlFile As String = tbxXmlFile.Text
    If Not File.Exists(xmlFile) Then
        Exit Sub
    End If

    Dim xr As New XmlTextReader(xmlFile)
    Dim vr As New XmlValidatingReader(xr)

    ' --- Handler hinzufügen
    AddHandler vr.ValidationEventHandler, _
        AddressOf Me.ValidatingEvent

    While xr.Read
        ' --- Das ist schon alles - keine Aktion
    End While

    ' --- Anzeige des Ergebnisses
    If doc_ok Then
        lblResult.ForeColor = Color.DarkGreen
        lblResult.Text = "Das Dokument ist In Ordnung."
    Else
        lblResult.ForeColor = Color.DarkRed
        lblResult.Text = "Das Dokument ist nicht ok."
    End If
End Sub
End Class

```

Listing 9.3: Einfaches Validieren eines XML-Dokuments [XmlValidation.sln]

Sie sehen an Listing 9.3, dass innerhalb der Leseschleife keinerlei Aktion durchgeführt wird. Zur einfachen Validierung ist das auch nicht nötig. Wenn ein Fehler auftreten sollte, wird ein Ereignis ausgelöst, das in diesem Fall mit der Methode `ValidatingEvent` verbunden ist. Dort wird die für die Klasse global deklarierte Variable `doc_ok` auf `False` gesetzt. Hinterher muss nur noch eine Kontrolle von `doc_ok` durchgeführt werden. Wenn der Wert der Variablen

False ist, ist das Dokument auch nicht gültig. Entsprechend wird die Ausgabe gestaltet. Abbildung 9.6 zeigt einen Screenshot des Beispielsprogramms.

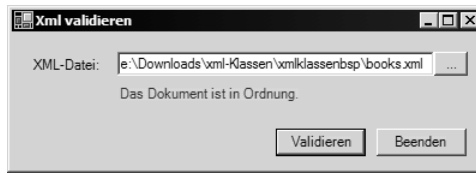


Abbildung 9.6: Validieren eines Dokuments (XmlValidation.exe)

ValidationType Mit der Eigenschaft `ValidationType` können Sie dem `XmlValidatingReader` zusätzlich angeben, auf welche Art er validieren soll. Die Standardeinstellung `ValidationType.Auto` validiert den gefundenen Daten entsprechend, d.h. nach einer DTD, wenn diese angegeben ist, nach einem Schema, wenn ein solches angegeben ist, und nur auf Wohlgeformtheit, wenn nichts angegeben ist.

Weitere Einstellungen für `ValidationType` sind

- ▶ `ValidationType.DTD` für die Validierung nach einer DTD
- ▶ `ValidationType.Schema` für die Validierung nach einem XSL-Schema
- ▶ `ValidationType.XDR` für die Validierung nach einem XDR-Schema (auch inline)
- ▶ `ValidationType.None` für Prüfung auf Wohlgeformtheit, es wird keine Validierung durchgeführt

Damit ist es auf sehr einfache Art möglich, ein Dokument zu überprüfen und auch sicherzustellen, dass das Dokument dem verwendeten Schema oder der verwendeten DTD entspricht, bevor es weiterverwendet wird.

Eingegebenen Text validieren

XML-Strings einlesen

Wenn Sie nicht etwa ein bestehendes Dokument validieren wollen, sondern eines, das in der Erstellung begriffen ist (also wenn der Anwender XML-Code eingibt), so sind Sie nicht gezwungen, den vorhandenen Text erst in eine Datei zu schreiben, bevor Sie ihn kontrollieren können. Stattdessen können Sie einen überladenen

Konstruktor der `XmlTextReader`-Klasse verwenden, dem Sie einfach einen String übergeben. Dieser enthält das XML-Fragment, das validiert werden soll.

Der einzige Unterschied zum Beispiel in Listing 9.3 ist also das Erzeugen des `XmlTextReader`-Objekts. Angenommen, Sie hätten eine Textbox namens `txtXml`, in die der Benutzer das XML-Fragment eingibt. Sie könnten dann mit einer kleinen Änderung auf dieses Fragment zugreifen und im Falle eines Validierungsfehlers diesen ausgeben:

```
Private Sub btnValidate_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnValidate.Click

    Dim xmlFile As String = txtXmlFile.Text
    If Not File.Exists(xmlFile) Then
        Exit Sub
    End If

    Dim xr As New XmlTextReader(txtXml.Text)
    Dim vr As New XmlValidatingReader(xr)

    'Handler hinzufügen
    AddHandler vr.ValidationEventHandler, _
        AddressOf Me.ValidatingEvent

    While xr.Read
        'Das ist schon alles - keine Aktion
    End While

    'Anzeige des Ergebnisses
    If doc_ok Then
        lblResult.ForeColor = Color.DarkGreen
        lblResult.Text = "Das Dokument ist In Ordnung."
    Else
        lblResult.ForeColor = Color.DarkRed
        lblResult.Text = "Das Dokument ist nicht In Ordnung."
    End If
End Sub
```

XML-Daten aus XmlDocument lesen

Wenn die XML-Daten bereits in Form einzelner Nodes vorhanden sind, z. B. wenn Sie sie in ein Objekt des Typs `XmlDocument` geladen haben, können Sie logischerweise nicht die Klasse `XmlTextReader` verwenden, da die Daten ja nicht als Text, Datei oder Stream vorliegen. Dennoch könnte es möglich sein, dass Sie Daten in einem

XmlNodeReader

XmlDocument-Objekt vorliegen haben und diese mittels eines Readers ausgeben wollen (oder auch nur Teile davon). Dann bietet sich der XmlNodeReader an.

Der Konstruktor der XmlNodeReader-Klasse erwartet ein Objekt vom Typ XmlNode. Ein Objekt vom Typ XmlDocument können wir verwenden, da es von XmlNode abgeleitet ist – einer der Vorzüge objekt-orientierter Programmierung. Somit können wir auch mit Hilfe des XmlNodeReader eine TreeView-Komponente recht einfach füllen.

```
'=====
' XmlNodeReaderExample.sln
' Copyright 2002 Frank Eller
' Einlesen von XmlDocument in eine Treeview
'=====

Imports System.Xml
Imports System.IO

Public Class Form1
    Inherits System.Windows.Forms.Form

    #Region " Windows Form Designer generated code "

        'Der Code, der von VS.NET eingefügt wurde, wird
        'aus Übersichtsgründen nicht dargestellt.

    #End Region

    Private Sub FillTreeview(ByVal filename As String)

        ' --- Deklarationen
        Dim actNode As TreeNode = Nothing
        Dim prevNode As TreeNode = Nothing
        Dim doc As New XmlDocument()
        Dim newPrevNode As Boolean

        doc.Load(filename)

        ' --- Einsatz des XmlNodeReader
        Dim nReader As New XmlNodeReader(doc)

        xmlTreeView.Nodes.Clear()
        xmlTreeView.Nodes.Add(filename)

        prevNode = xmlTreeView.Nodes(0)

        xmlTreeView.BeginUpdate()
```



```

While nReader.Read()

    Select Case nReader.NodeType
        Case XmlNodeType.Element
            actNode = New TreeNode(nReader.Name)
            newPrevNode = Not nReader.IsEmptyElement
            While nReader.MoveToNextAttribute()
                Dim newNode As New TreeNode(nReader.Name)
                newNode.Nodes.Add(New TreeNode("[ " + _
                    nReader.Value + "]"))
                actNode.Nodes.Add(newNode)
            End While
            prevNode.Nodes.Add(actNode)
            If newPrevNode Then
                prevNode = actNode
            End If

        Case XmlNodeType.EndElement
            prevNode = prevNode.Parent

        Case XmlNodeType.Text
            Dim nodeText As String = nReader.Value
            If nodeText.Length > 20 Then
                nodeText = nodeText.Substring(0, 17) + "..."
            End If
            prevNode.Nodes.Add(nodeText)
    End Select

End While

xmlTreeView.EndUpdate()
xmlTreeView.ExpandAll()

nReader.Close()
End Sub

Private Sub btnClearTree_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnClearTree.Click

    xmlTreeView.Nodes.Clear()
End Sub

Private Sub btnLoad_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnLoad.Click

    If dlgOpen.ShowDialog() = DialogResult.OK Then
        FillTreeview(dlgOpen.FileName)
    End If

```

```

End Sub

Private Sub btnExit_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnExit.Click

    Me.Close()
End Sub
End Class

```

Listing 9.4: XML-Nodes in eine Treeview einlesen [XmlNodeReaderExample.sln]

Das Ergebnis des Codes aus Listing 9.4 sehen Sie in Abbildung 9.7.

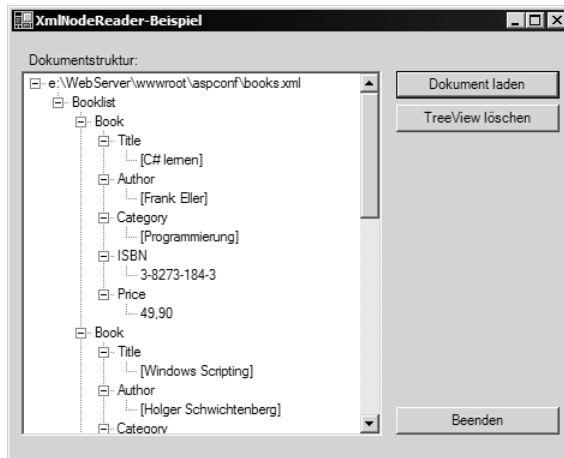


Abbildung 9.7: XML aus XmlDocument lesen (XmlNodeReaderExample.sln)

9.1.3 XML-Daten schreiben

XmlTextWriter

Das Gegenstück zum `XmlReader` ist der `XmlWriter`. Auch hierbei handelt es sich um eine abstrakte Klasse, von der es eine funktionsfähige Implementation im .NET-Framework gibt, nämlich den `XmlTextWriter`. Wie der `XmlTextReader` arbeitet diese Klasse nur vorwärts und schreibt stets alle Daten seriell in eine Datei.

Damit ist klar, dass diese Möglichkeit, XML zu schreiben, nur dann verwendet werden kann, wenn auch wirklich alle Daten vorliegen. Beispielsweise könnte man sich vorstellen, den Inhalt einer komma-basierten Textdatei (CSV-Datei) als XML-Datei abzuspeichern, um die enthaltenen Daten später wieder verwenden zu können.

`XmlTextWriter` bietet mehrere Methoden zum Schreiben von Daten. Die wichtigsten sind:

- ▶ `WriteStartDocument()`, schreibt den Anfang einer XML-Datei (also die XML-Deklaration mit der Version 1.0)
- ▶ `WriteStartElement()`, schreibt das Start-Tag eines Elements
- ▶ `WriteEndElement()`, schreibt das End-Tag eines Elements
- ▶ `WriteElementString()`, schreibt ein Element komplett mit Start- und End-Tag sowie den Wert
- ▶ `WriteStartAttribute()`, schreibt den Beginn eines Attributs. Auch hier muss vorher ein Element begonnen worden sein.
- ▶ `WriteEndAttribute()`, schreibt das Ende eines Attributs
- ▶ `WriteAttributeString()`, schreibt ein Attribut komplett mit Namen und Wert in die Datei. Vorher muss `WriteStartElement()` aufgerufen worden sein.
- ▶ `WriteString()`, schreibt den übergebenen Wert. Wenn vorher `WriteStartElement()` aufgerufen wurde, schreibt `WriteString()` den Wert, den das Element enthalten soll. Wenn vorher `WriteStartAttribute()` aufgerufen wurde, schreibt `WriteString()` den Wert des Attributs.
- ▶ `WriteCData()`, schreibt einen CDATA-Block an die aktuelle Stelle in der Datei

Es gibt natürlich noch wesentlich mehr Methoden zum Schreiben, aber vorerst sollen diese genügen. Die anderen funktionieren entsprechend; so schreibt z.B. die Methode `WriteComment()` einen Kommentar in die Datei.

Das Schreiben selbst funktioniert wieder mehr oder weniger in einer Schleife, indem die Daten nacheinander – z.B. aus einem Array heraus – geliefert und dann entsprechend der Dateistruktur geschrieben werden. Wie letztendlich das fertige Dokument geschrieben werden soll, können Sie über einige Eigenschaften der Klasse `XmlTextWriter` festlegen. So bestimmen Sie über die Eigenschaft `Formatting`, wie das Dokument formatiert werden soll (z.B. mit Einzügen oder ohne), über `Indentation` legen Sie fest, wie groß der Einzug sein soll, falls Sie einen verwenden, und über die Eigenschaft `Namespace`, ob XML-Namespaces unterstützt werden sollen.

Formatieren der Ausgabe

XML-Daten schreiben

Das folgende Beispiel zeigt, wie mit Hilfe eines `XmlTextWriter`-Objekts Daten geschrieben werden können. Das Beispiel ist ein wenig größer als die bisherigen, es schreibt einen kompletten Satz von Links in eine Datei.

LinkItem Basis der Daten ist die Klasse `LinkItem`, die einen kompletten Link beschreibt, mit Namen, URL, Stil und Ziel (Target). Dabei handelt es sich eigentlich um eine simple Klasse mit vier String-Werten, die über Eigenschaften angesprochen werden können. Der überladene Konstruktor der Klasse ermöglicht es, alle Daten direkt zu setzen. Die Klassendeklaration wurde in eine eigene Datei verlagert und sieht wie folgt aus:

```
'=====
' XmlWriterExample.sln
' Copyright 2002 Frank Eller
' Klasse LinkItem
'=====
```

```
Public Class LinkItem
    Private _name As String = String.Empty
    Private _link As String = String.Empty
    Private _target As String = String.Empty
    Private _style As String = String.Empty

    Public Property Name() As String
        Get
            Return _name
        End Get
        Set(ByVal Value As String)
            _name = Value
        End Set
    End Property

    Public Property Link() As String
        Get
            Return _link
        End Get
        Set(ByVal Value As String)
            _link = Value
        End Set
    End Property

    Public Property Target() As String
        Get
            Return _target
        End Get
        Set(ByVal Value As String)
```

```

        _target = Value
    End Set
End Property

Public Property Style() As String
    Get
        Return _style
    End Get
    Set(ByVal Value As String)
        _style = Value
    End Set
End Property

Public Sub New()
    ' --- Standard-Konstruktor
End Sub

Public Sub New(ByVal n As String, ByVal l As String, _
    ByVal t As String, ByVal s As String)

    Me._name = n
    Me._link = l
    Me._target = t
    Me._style = s
End Sub

End Class

```

Listing 9.5: Die Klasse LinkItem [XmlWriterExample.sln]

Dazu gehört eine Klasse `Links`, die eigentlich nur eine `ArrayList` und den Namen der Datei beinhaltet, in die geschrieben werden soll. Die Deklaration ist daher denkbar einfach:

Links

```

Public Class Links

    Public Shared XmlFileName As String = "d:\links.xml"
    Public Shared Items As New ArrayList()

End Class

```

Listing 9.6: Die Klasse Links [XmlWriterExample.sln]

Das statische `ArrayList`-Objekt enthält unsere `Links` in Form von `LinkItem`-Objekten, so dass wir sie später sehr leicht auslesen können. Die eigentliche Funktionalität passiert in der Klasse `XmlWorker`, in der wir die Daten mittels `XmlTextReader` und `XmlTextWriter` lesen und schreiben. Alle Methoden dieser Klasse sind statisch; die Methode `ReadLinks()` liefert ein `ArrayList`-Objekt zurück, die Methode `WriteLinks()` benötigt eines als Parameter. In diesem Fall hätte man das auch anders machen können, da die zu schreibende

XmlWorker

(oder zu lesende) `ArrayList` ja bereits existiert. Die Methoden sind aber auf diese Weise universeller zu benutzen, sollte dies einmal nötig sein. Listing 9.7 zeigt die gesamte Klasse `XmlWorker`.

```
'=====
' XmlWriterExample.sln
' Copyright 2002 Frank Eller
' Klasse XmlWorker
'=====

Public Class XmlWorker

    Public Shared Sub CreateXmlFile()
        Dim writer As New XmlTextWriter(Links.XmlFileName, _
            Encoding.Unicode)
        writer.WriteStartDocument(True)
        writer.WriteEndDocument()
        writer.Flush()
        writer.Close()
    End Sub

    Public Shared Function ReadLinks() As ArrayList
        Dim result As New ArrayList()
        If Not File.Exists(Links.XmlFileName) Then
            XmlWorker.CreateXmlFile()
        End If

        Dim reader As New XmlTextReader(Links.XmlFileName)

        While reader.Read
            If reader.Name = "link" Then
                Dim nName As String = _
                    reader.GetAttribute("name")
                Dim nLink As String = reader.GetAttribute("url")
                Dim nTarget As String = _
                    reader.GetAttribute("target")
                Dim nStyle As String = _
                    reader.GetAttribute("style")
                result.Add(New LinkItem(nName, nLink, nTarget, _
                    nStyle))
            End If
        End While

        Return result
    End Function

    Public Shared Sub WriteLinks(ByVal arl As ArrayList)
        Dim link As LinkItem
        Dim o As Object

        Dim writer As New XmlTextWriter(Links.XmlFileName, _
            Encoding.Unicode)
```

```

writer.Indentation = 4
writer.Formatting = Formatting.Indented
writer.Namespaces = False
writer.IndentChar = CChar(" ")

writer.WriteStartDocument(True)
writer.WriteStartElement("links")

For Each o In ar1
    link = CType(o, LinkItem)
    writer.WriteStartElement("link")
    writer.WriteAttributeString("name", link.Name)
    writer.WriteAttributeString("url", link.Link)
    writer.WriteAttributeString("target", link.Target)
    writer.WriteAttributeString("style", link.Style)
    writer.WriteEndElement()
Next
writer.WriteEndElement()
writer.WriteEndDocument()
writer.Flush()
writer.Close()
End Sub

End Class

```

Listing 9.7: Die Klasse XmlWorker (XmlWriterExample.sln)

Auch diese Klasse steht in einer eigenen Datei. Der Rest des Programms besteht aus dem Einfügen und Löschen der Datensätze und der Anzeige im Formular.

Interessant ist im Zusammenhang mit dem `XmlTextWriter` vor allem die Methode `WriteLinks()`, die allerdings sehr leicht verständlich sein sollte. In diesem Beispiel ist nur eine Verschachtelung enthalten, nämlich die, dass alle Elemente der Datei unter ein Wurzelement geschrieben werden. Im Konstruktor der `XmlTextWriter`-Klasse wird angegeben, dass eine Datei im Unicode-Format erstellt werden soll. Weiterhin soll der Einzugs vier Stellen betragen, als Einzugszeichen sollen Leerzeichen verwendet werden und der Einzugs soll auch wirklich verwendet werden. Namespaces sollen keine benutzt werden.

Zunächst wird der Dokumentstart geschrieben. Der übergebene Parameter bedeutet, dass es sich hierbei um eine Standalone-Datei handelt. Dann schreiben wir das Root-Element, das jede XML-Datei per definitionem beinhalten muss. Da aber die Elemente hierarchisch darunter eingefügt werden sollen, wird zunächst nur der Start des Root-Knotens geschrieben, dann folgen die Elemente

**Einstellungen für
XmlTextWriter**

**Schreiben der
Datei**

und schließlich als Letztes das Ende des Root-Knotens. Der Rest besteht nur noch im physikalischen Schreiben auf die Festplatte und dem Schließen der Datei.

9.2 XmlDocument

Die Klasse `XmlDocument` wurde im Beispiel mit dem `XmlNodeReader` schon einmal kurz angesprochen. `XmlDocument` ist der Mittelpunkt der Implementierung des Document Object Models (DOM) in der Framework Class Library (FCL). Diese Klasse bietet die Möglichkeit, ein komplettes Dokument in den Speicher einzulesen, zu ändern, wieder zu speichern oder Werte daraus zu suchen.

XmlNode Der Aufbau der `XmlDocument`-Klasse ist zwar logisch, aber in der Regel nicht auf den ersten Blick zu durchschauen. Abbildung 9.8 zeigt die Zusammenhänge innerhalb der Klasse.

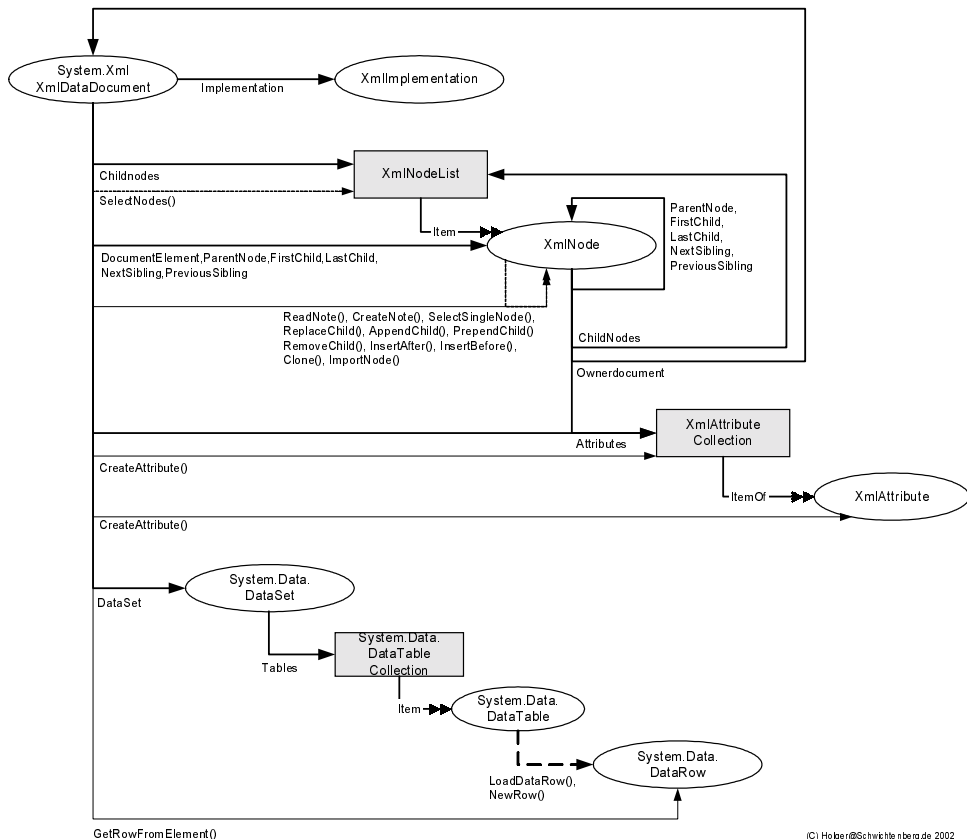


Abbildung 9.8: Die Klasse `XmlDocument` in der Übersicht

Das wichtigste Element ist dabei wohl `XmlNode`, von dem `XmlDocument` selbst abgeleitet ist und von dem auch alle untergeordneten Knoten abgeleitet sind. `XmlNode` enthält alle relevanten Methoden, um weitere Knoten hinzuzufügen, Knoten zu entfernen oder zu ändern.

Abbildung 9.9 zeigt die Hierarchie der relevanten Klassen.

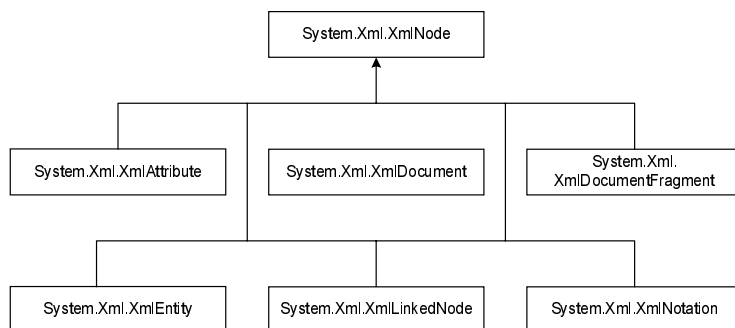


Abbildung 9.9: Die Hierarchie der `XmlNode`-Klassen

9.2.1 Auslesen der Daten

Jedes Objekt vom Typ `XmlNode` kann weitere Unterknoten enthalten, die in einer Collection vom Typ `XmlNodeList` namens `ChildNodes` abgelegt sind. Da auch `XmlDocument` selbst von `XmlNode` abgeleitet ist, enthält auch diese Klasse die untergeordneten Knoten in Form einer `XmlNodeList`. Die enthaltenen Knoten können durch die Eigenschaften `Item` oder `ItemOf` angesprochen werden, wobei es sich bei letzterer um den Indexer der `XmlNodeList` handelt. D.h. wenn Sie einen Node aus dem Dokument herauslesen wollen und wissen, an welcher Stelle er sich innerhalb des Dokuments befindet, führen die folgenden zwei Anweisungen zum gleichen Ergebnis:

NodeList

```

Dim node As XmlNode
node = theDocument.ChildNodes.Item(5)
node = theDocument.ChildNodes(5)

```

Oder, wenn man sich den Code in C# ansieht:

```

XmlNode node;
node = theDocument.ChildNodes.Item[5];
node = theDocument.ChildNodes[5];

```

HasChildNodes Damit haben Sie einen leichten Zugang zu den jeweiligen Unterknoten eines `XmlNode`-Objekts. Möchten Sie wissen, ob der gefundene Knoten weitere Unterknoten hat, können Sie die Eigenschaft `HasChildNodes` benutzen. Ist diese `True`, greifen Sie einfach wieder über den Indexer oder die Eigenschaft `Item` auf die untergeordneten Knoten zu.

Knoten auslesen

Das folgende Beispiel nutzt dies aus, indem es alle in einem `XmlDocument` enthaltenen Knoten entsprechend der Hierarchie in einer `TreeView` ausgibt. Dazu wird zunächst der Dateiname als erster Knoten festgelegt, alle weiteren Knoten des Dokuments werden mittels einer einfachen rekursiven Funktion angehängt.

```
'=====
' XmlDocumentExample.sln
' Copyright 2002 Frank Eller
' Knoten eines Dokuments In Treeview ausgeben
'=====

Imports System.Xml

Public Class Form1
    Inherits System.Windows.Forms.Form

    #Region " Windows Form Designer generated code "

    'Der Code, der von VS.NET eingefügt wurde, wird
    'aus Übersichtsgründen nicht dargestellt.

    #End Region

    Private Function GetNodes(ByVal rootNode As XmlNode, _
        ByVal treeRoot As TreeNode) As TreeNode

        Dim node As XmlNode
        Dim newNode As TreeNode

        If rootNode.HasChildNodes Then
            For Each node In rootNode.ChildNodes
                newNode = treeRoot.Nodes.Add(node.Name)
                GetNodes(node, newNode)
            Next
        End If

        Return treeRoot
    End Function
```

```

Private Sub FillTreeView(ByVal fileName As String)
    Dim doc As New XmlDocument()
    Dim treeRoot As TreeNode

    doc.Load(fileName)
    xmlTreeView.Nodes.Clear()

    'Nicht immer neu zeichnen
    xmlTreeView.BeginUpdate()

    Dim node As New TreeNode(fileName)
    treeRoot = GetNodes(doc.DocumentElement, node)
    xmlTreeView.Nodes.Add(treeRoot)
    xmlTreeView.ExpandAll()

    'Fertig mit Füllen
    xmlTreeView.EndUpdate()
End Sub

Private Sub btnLoad_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnLoad.Click

    If dlgOpenFile.ShowDialog() = DialogResult.OK Then
        FillTreeView(dlgOpenFile.FileName)
    End If
End Sub

Private Sub btnExit_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnExit.Click

    Me.Close()
End Sub
End Class

```

Listing 9.8: XML-Knoten in einer Treeview ausgeben [XmlDocumentExample.sln]

Das Beispielprogramm gibt allerdings nur den Namen des Nodes aus, keinen Text und keine Attribute. Die Ausgabe des Programms sehen Sie in Abbildung 9.10.

Die Eigenschaft `Attributes`, eine Collection vom Typ `XmlAttributeCollection`, enthält alle Attribute in Form von `XmlAttribute`-Objekten. Über die Eigenschaft `Count` der Collection können Sie die Anzahl der enthaltenen Attribute ermitteln, über die Eigenschaft `Item` oder über den Indexer `ItemOf` können Sie auf die Attribute zugreifen. Das wird allerdings erst dann sinnvoll, wenn auch wirklich Attribute enthalten sind – ansonsten liefert eine Abfrage von `Attributes` nämlich den Wert `nothing` (oder `null` in C#).

Attributes

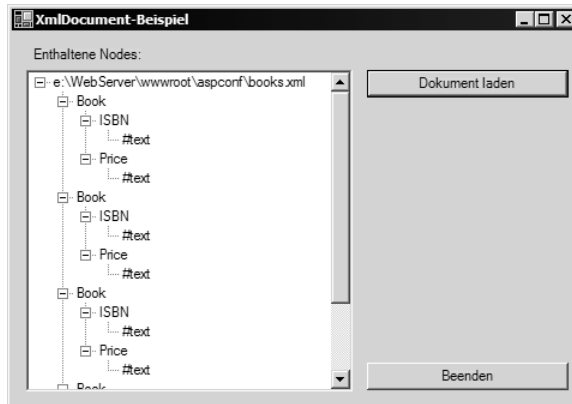


Abbildung 9.10: Die Nodes in einer Treeview (XmlDocumentExample.exe)

Attribute auslesen

Im folgenden Beispiel werden nicht nur die Nodes ausgelesen, sondern auch die enthaltenen Attribute. Wieder werden die Daten in einer TreeView-Komponente dargestellt.

```
'=====
' DocWithAttributes.sln
' Copyright 2002 Frank Eller
' Einlesen von XML mit Attributen
'=====
```

```
Imports System.Xml
```

```
Imports System.Collections
```

```
Public Class Form1
```

```
    Inherits System.Windows.Forms.Form
```

```
#Region " Windows Form Designer generated code "
```

```
'Der Code, der von VS.NET eingefügt wurde, wird
'aus Übersichtsgründen nicht dargestellt.
```

```
#End Region
```

```
Private Function GetNodes(ByVal rootNode As XmlNode, _
    ByVal treeRoot As TreeNode) As TreeNode
```

```
    ' --- Deklarationen
```

```
    Dim node As XmlNode
```

```
    Dim nodeString As String
```

```
    Dim attString As String
```

```
    Dim attr As XmlAttribute
```

```

Dim newNode As TreeNode

' --- Alle Nodes holen
If rootNode.HasChildNodes Then
    For Each node In rootNode.ChildNodes
        nodeString = node.Name
        ' --- Attribute checken
        If Not (node.Attributes Is Nothing) Then
            For Each attr In node.Attributes
                attString = " [" + attr.Name + "]"
                nodeString = nodeString + attString
            Next
        End If
        newNode = treeRoot.Nodes.Add(nodeString)
        GetNodes(node, newNode)
    Next
End If
Return treeRoot
End Function

Private Sub FillTreeView(ByVal filename As String)
    Dim doc As New XmlDocument()
    Dim node As TreeNode
    Dim treeroot As TreeNode

    doc.Load(filename)
    xmlTreeView.Nodes.Clear()

    ' --- Nicht jedes Mal neu zeichnen
    xmlTreeView.BeginUpdate()
    node = New TreeNode(filename)
    treeroot = GetNodes(doc.DocumentElement, node)
    xmlTreeView.Nodes.Add(treeroot)
    xmlTreeView.ExpandAll()

    ' --- Füllen fertig
    xmlTreeView.EndUpdate()
End Sub

Private Sub btnLoad_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnLoad.Click

    If dlgOpenFile.ShowDialog() = DialogResult.OK Then
        FillTreeView(dlgOpenFile.FileName)
    End If
End Sub

Private Sub btnExit_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnExit.Click

```

```

        Me.Close()
    End Sub
End Class

```

Listing 9.9: Dokument mit Attributen lesen [DocWithAttributes.sln]

Es wird kontrolliert, ob der Node, der gerade bearbeitet wird, auch Attribute hat. Wenn dem so ist, werden die Bezeichner dieser Attribute über eine `For Each`-Schleife geholt und in eckigen Klammern hinter den Bezeichner geschrieben.

Nur bestimmte Knoten anzeigen

In den vorangegangenen Beispielen wurden stets alle Knoten des Dokuments ausgelesen. Wie verhält es sich jedoch jetzt, wenn nur bestimmte Knoten ausgelesen werden sollen? Angenommen, Sie wollten erfahren, welche Autoren an all den in der XML-Datei gespeicherten Büchern beteiligt sind. Dann wäre es sinnvoll, nur die Knoten auszulesen, die die Autorennamen beinhalten, und diese dann zurückzuliefern.

GetElements- ByTagName()

Die Klasse `XmlDocument` ermöglicht dies über eine bestimmte Methode namens `GetElementsByTagName()`. Diese Methode liefert eine `XmlNodeList` zurück, die nur die Knoten beinhaltet, deren Name dem der Methode übergebenen entspricht. Auf diese Art und Weise ist es möglich, z. B. alle Autoren der Bücher anzuzeigen.

Die Namen der Autoren sind als Attribute in den `<Books>`-Elementen gespeichert. Es müssen also alle `<Books>`-Elemente kontrolliert und dort jeweils das Attribut `<Author>`, das den Autorennamen enthält, ausgewertet werden. Dabei hilft es, dass auch über den Namen eines Attributs auf dasselbe zugegriffen werden kann.

```

Dim nodes As XmlNodeList
nodes = doc.GetElementsByTagName("Book")
For Each xNode In nodes
    node.Nodes.Add( _
        xNode.Attributes("Author").Value.ToString())
Next

```

Den gesamten Quelltext, der nur die Autoren zurückliefert, sehen Sie in Listing 9.10.

```

'=====
' GetSelectedNodes.sln
' Copyright 2002 Frank Eller
' Ausgewählte Nodes auslesen
'=====

```

```
Imports System.Xml

Public Class Form1
    Inherits System.Windows.Forms.Form

    #Region " Windows Form Designer generated code "

        'Der Code, der von VS.NET eingefügt wurde, wird
        'aus Übersichtsgründen nicht dargestellt.

    #End Region

    Private Sub FillTreeView(ByVal filename As String)
        Dim doc As New XmlDocument()
        Dim nodes As XmlNodeList
        Dim xNode As XmlNode

        doc.Load(filename)
        xmlTreeView.Nodes.Clear()

        ' --- Treeview nicht immer neu zeichnen
        xmlTreeView.BeginUpdate()

        Dim node As New TreeNode(filename)
        nodes = doc.GetElementsByTagName("Book")
        For Each xNode In nodes
            node.Nodes.Add( _
                xNode.Attributes("Author").Value.ToString())
        Next
        xmlTreeView.Nodes.Add(node)
        xmlTreeView.ExpandAll()

        ' --- Füllen fertig
        xmlTreeView.EndUpdate()
    End Sub

    Private Sub btnLoad_Click( _
        ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles btnLoad.Click

        If dlgOpenFile.ShowDialog() = DialogResult.OK Then
            FillTreeView(dlgOpenFile.FileName)
        End If
    End Sub

    Private Sub btnExit_Click( _
        ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles btnExit.Click
```

```

        Me.Close()
    End Sub
End Class

```

Listing 9.10: *GetSelectedNodes.exe*

Zunächst holen wir uns in diesem Beispiel alle Nodes mit dem Namen `<Book>` und liefern sie als `XmlNodeList` zurück. Dann iterieren wir mittels einer `For Each`-Schleife durch diese Nodes und holen uns jeweils den Wert des Attributs mit dem Namen `<Author>`.

`XmlDocument` bietet aber noch weitere Methoden für den Zugriff auf die enthaltenen Daten, die hier nicht im Einzelnen erläutert werden können, da es den Rahmen dieses Kapitels sprengen würde. Einige davon werden wir noch genauer kennen lernen, allerdings nicht anhand eines eigenen Beispiels.

9.2.2 Daten in XmlDocument ändern bzw. hinzufügen

Daten ändern Um Daten in einem `XmlDocument` zu ändern, muss man eigentlich nur herausfinden, welchen Knoten man ändern will, und kann dann dessen Werte über die Methoden der Klasse `XmlNode` ändern. Da die Auswahl eines Knotens der Referenz auf diesen Knoten entspricht, d.h. da nicht etwa die Daten ausgelesen werden, sondern ein Objekt mit Referenz auf den gewünschten Knoten zurückgeliefert wird, ist das relativ einfach, denn man arbeitet ja sozusagen »live« im Dokument.

Node-Typen Üblicherweise wird kein Element vom Typ `XmlNode` benutzt, um Daten hinzuzufügen, sondern gleich ein Element des Typs, der hinzugefügt werden soll. Die Klasse `XmlDocument` stellt hierfür einige Methoden zur Verfügung, mit denen die verschiedenen Knotentypen erzeugt werden können. Einige der wichtigsten sind im Folgenden aufgelistet.

- ▶ `CreateAttribute()` erzeugt ein Objekt vom Typ `XmlAttribute`, das mit Hilfe der Methode `SetAttribute()` von `XmlNode` gesetzt werden kann. `SetAttribute()` allerdings erfordert kein Objekt vom Typ `XmlAttribute`, weil diese Methode auch in der Lage ist, die Werte eines Attributs direkt einzutragen.
- ▶ `CreateCdataSection()` erzeugt einen CDATA-Teil, der in das Dokument eingefügt werden kann. Innerhalb dieses Teils können auch HTML-Tags verwendet werden, die ohne die CDATA-Deklaration als XML-Tags interpretiert würden und so ein Dokument unbrauchbar machen könnten.

- ▶ `CreateElement()` erzeugt ein Objekt vom Typ `XmlElement`, das dem Dokument oder einem Node hinzugefügt werden kann. Da es sich bei `XmlElement` eigentlich auch nur um eine von `XmlNode` abgeleitete Klasse handelt, kann auch hier ein `XmlAttribute`-Objekt mit Hilfe von `SetAttribute()` hinzugefügt werden.
- ▶ `CreateNode()` erzeugt einen `XmlNode` des angegebenen Typs. Diese Methode kann für alle Knotentypen verwendet werden. Üblicherweise plädiere ich auch der Übersichtlichkeit halber für die Verwendung der spezialisierten Methoden.
- ▶ `CreateTextNode()` erzeugt einen Text, der in das Dokument eingefügt werden kann.

Es gibt natürlich noch mehr Methoden, z. B. `CreateComment()`, `CreateWhiteSpace()` oder `CreateDocumentFragment()`, die Funktionsweise ist aber immer die gleiche: Die Methoden liefern ein Objekt des angegebenen Typs zurück.

Ebenfalls interessant und sicher nützlich sind Methoden zum Auffinden bestimmter Knoten innerhalb des Dokuments. `GetElementsByTagName()` haben wir bereits kennen gelernt, es gibt aber noch weitere.

Eine interessante Methode ist beispielsweise `SelectSingleNode()`, die einen bestimmten Knoten anhand eines XPath-Ausdrucks zurückliefert. Die Möglichkeiten sind innerhalb von `XmlDocument` natürlich begrenzt, dennoch ist es möglich, mit einfachen XPath-Ausdrücken bestimmte Knoten auszuwählen. Übergeben wird lediglich der XPath-Ausdruck, der verwendet werden soll. Der erste Knoten, der darauf passt, wird als `XmlNode`-Objekt zurückgeliefert.

SelectSingleNode()

Wollen Sie mehrere Knoten anhand eines XPath-Ausdrucks zurückliefern, verwenden Sie die Methode `SelectNodes()`. Die Knoten werden als `XmlNodeList` zurückgeliefert. Trotz all dieser Möglichkeiten sind diese Funktionen nicht so mächtig wie eine richtige XPath-Implementation. Für viele Fälle werden sie jedoch ausreichen und bieten einen leichten Zugriff auf bestimmte Daten. Im nächsten Beispiel werden wir `SelectSingleNode()` verwenden.

SelectNodes()

Knoten einfügen

Um nun einen neu erstellten Knoten in ein Dokument einzufügen, gibt es mehrere Wege. Der einfachste Weg ist sicherlich, den Kno-

AppendChild()

ten einfach am hinteren Ende anzufügen. Dazu bietet `XmlNode` (und entsprechend auch `XmlDocument`) die Methode `AppendChild()`, die einen Unterknoten am Ende der Liste einfügt.

**InsertAfter(),
InsertBefore()**

Das Einfügen eines Knotens an einer bestimmten Stelle im Dokument ist ebenfalls nicht weiter schwierig. Die Methode `InsertAfter()` fügt einen Knoten nach einem bestimmten bestehenden Knoten ein, die Methode `InsertBefore()` tut das Gleiche vor dem angegebenen Knoten. Sie müssen lediglich darauf achten, diese Methoden für das richtige `XmlNode`-Objekt aufzurufen.

Beispiel: XML-Daten ändern

Anhand eines Beispiels wird vieles klarer. In diesem Fall habe ich mich für ein etwas umfangreicheres Beispiel entschieden, das die verschiedenen Techniken beim Umgang mit `XmlDocument` illustrieren soll.



Das Beispiel ist –wie bereits gesagt– umfangreicher als die vorhergehenden. Unter anderem enthält es auch eine Methode, die mittels eines XPath-Ausdrucks einen Node auswählt. Die entsprechende Methode wird daher verständlicherweise erst später angesprochen.

Benutzt wird wie schon in den vorangegangenen Beispielen eine XML-Datei, die Buchdaten beherbergen soll. Der Aufbau dieser XML-Datei sieht folgendermaßen aus:

```
<?xml version="1.0"?>
<Booklist>
  <Book Title="C# lernen" Author="Frank Eller"
    Category="Programmierung">
    <ISBN>3-8273-184-3</ISBN>
    <Price>49,90</Price>
  </Book>
  <Book Title="Windows Scripting" Author="Holger
    Schwichtenberg" Category="Scripting">
    <ISBN>3-8273-1843-2</ISBN>
    <Price>119,90</Price>
  </Book>
</Booklist>
```

Titel, Autor und Kategorie sind als Attribute eingefügt, die ISBN-Nummer des Buchs und auch der Preis sind eigenständige untergeordnete Knoten. Für die relevanten Daten eines Buchs wurde eine eigene Klasse deklariert:

Public Class BookInfo

```
Private _Isbn As String
Private _Author As String
Private _Category As String
Private _Title As String
Private _Price As Double
```

```
Property Isbn() As String
    Get
        Return _Isbn
    End Get
    Set(ByVal Value As String)
        _Isbn = Value
    End Set
End Property
```

```
Property Author() As String
    Get
        Return _Author
    End Get
    Set(ByVal Value As String)
        _Author = Value
    End Set
End Property
```

```
Property Category() As String
    Get
        Return _Category
    End Get
    Set(ByVal Value As String)
        _Category = Value
    End Set
End Property
```

```
Property Title() As String
    Get
        Return _Title
    End Get
    Set(ByVal Value As String)
        _Title = Value
    End Set
End Property
```

```
Property Price() As Double
    Get
        Return _Price
    End Get
    Set(ByVal Value As Double)
        _Price = Value
    End Set
```

End Property

```
Public Sub New(ByVal t As String, ByVal a As String, _
    ByVal c As String, ByVal i As String, _
    ByVal p As Double)
```

```
    Me._Author = a
    Me._Category = c
    Me._Isbn = i
    Me._Price = p
    Me._Title = t
```

End Sub

```
Public Sub New()
```

End Sub

End Class

Listing 9.11: Die Klasse BookInfo

Alle für die XML-Datei relevanten Methoden wurden in eine Klasse `XmlWorker` gepackt, alle Methoden sind statisch. Da die Daten dargestellt werden sollen, benötigen wir zunächst eine Methode, die uns alle Bücher in Form einer `ArrayList` zurückliefert, die wir dann in einer Listbox darstellen können. Da wir für die Listbox nur die Titel der Bücher benötigen, liefern wir auch nur diese zurück.

Verschiedene Variablen, die öfter benötigt werden, wurden als statische (*shared*) Felder einer im ganzen Projekt verfügbaren Klasse deklariert. Der Name dieser Klasse ist `Global`. `Global.BookFileName` enthält beispielsweise die Bezeichnung der Datei, die die Buchdaten enthält.

```
Public Shared Function GetBookList() As ArrayList
    'Neue leere ArrayList erstellen
    Dim arl As New ArrayList()
    arl.Clear()

    'XML-Daten aus Datei laden
    Dim xmldoc As New XmlDocument()
    xmldoc.Load(Global.BookFileName)

    'Alle Nodes mit Namen 'Book' holen
    Dim xmlNodes As XmlNodeList
    xmlNodes = xmldoc.GetElementsByTagName(Global.DescBook)

    'Alle Nodes der ArrayList hinzufügen
    Dim node As XmlNode
    For Each node In xmlNodes
```

```

ar1.Add(node.Attributes(Global.DescTitle).Value.ToString())
Next

Return ar1
End Function

```

Listing 9.12: GetBookList

Über die Methode `GetElementsByTagName()` werden alle Knoten zurückgeliefert, die den Bezeichner `<Book>` haben. Im Beispiel ist dieser Bezeichner in der statischen Variablen `Global.DescTitle` abgelegt, damit etwaige Rechtschreibfehler während der Programmierung nicht zu einem fehlerhaften Ergebnis führen.

Die erste Methode, die wirklich etwas mit dem Ändern bzw. dem Einfügen von Daten zu tun hat, ist die Methode `CreateBook()` der Klasse `XmlWorker`. In dieser wird ein Objekt vom Typ `XmlElement` erstellt, das den angesprochenen Aufbau aufweist.

```

Public Shared Function CreateBook( _
    ByVal BI As BookInfo, _
    ByVal xmlDoc As XmlDocument) As XmlElement

    'Neuer Node
    Dim newBook As XmlElement
    newBook = xmlDoc.CreateElement(Global.DescBook)

    'Attribute einstellen
    newBook.SetAttribute(Global.DescTitle, BI.Title)
    newBook.SetAttribute(Global.DescAuthor, BI.Author)
    newBook.SetAttribute(Global.DescCategory, BI.Category)

    'ChildNode für ISBN
    Dim newISBN As XmlElement
    newISBN = xmlDoc.CreateElement(Global.DescIsbn)
    newISBN.InnerText = BI.Isbn
    'Hinzufügen
    newBook.AppendChild(newISBN)

    'ChildNode für Preis
    Dim newPrice As XmlElement
    newPrice = xmlDoc.CreateElement(Global.DescPrice)
    newPrice.InnerText = BI.Price.ToString("#,##0.00")
    'Hinzufügen
    newBook.AppendChild(newPrice)

    'Kompletten Node zurückliefern
    Return newBook
End Function

```

Listing 9.13: CreateBook()

Es wird ein neues Element `newBook` vom Typ `XmlElement` erstellt, denn genau so eines soll der Datei hinzugefügt werden. Über die Methode `SetAttribute()` werden die Attribute eingestellt. In diesem Beispiel wird `SetAttribute()` direkt auf das neue Element angewandt, ohne vorher ein Objekt vom Typ `XmlAttribute` erzeugt zu haben.

Die Unterknoten des neuen Buchelements werden ebenfalls über `CreateElement()` erzeugt und dann dem neuen Buchknoten hinzugefügt. Schließlich wird der gesamte Knoten zurückgeliefert, d.h. die Methode kann jetzt von anderer Stelle aus aufgerufen und der zurückgelieferte Knoten dann als neuer Buchknoten verwendet werden.

Die Methode zum Anhängen eines Knotens ist verhältnismäßig simpel. Es wird ein neuer Knoten über die Methode `CreateBook()` erzeugt und hinter dem Wurzelement der Datei eingefügt.

```
Public Shared Sub AppendBook(ByVal BI As BookInfo)
    'XML-Dokument laden
    Dim xmlDoc As New XmlDocument()
    xmlDoc.Load(Global.BookFileName)

    'Neu erzeugtes Element hinzufügen
    Dim newElement As XmlElement
    newElement = XmlWorker.CreateBook(BI, xmlDoc)
    xmlDoc.DocumentElement.AppendChild(newElement)

    'Und speichern
    xmlDoc.Save(Global.BookFileName)
End Sub
```

Listing 9.14: `AppendBook()`

Die nächste Methode, die benötigt wird, ist die zum Einfügen eines Elements in ein XML-Dokument. Da das neue Element jetzt an einer bestimmten Stelle eingefügt werden soll, wird ein Kriterium benötigt, anhand dessen der aktuelle Knoten ausgewählt werden kann. In diesem Beispiel ist das der Titel des Buchs, der eigentlich nahezu eindeutig sein sollte. In einem realen Beispiel würde man vermutlich eine ID hinzufügen (beispielsweise die ISBN-Nummer), die wirklich eindeutig ist und es so ermöglicht, auch gleiche Titel in die Liste einzutragen.

```
Public Shared Sub InsertBook(ByVal BI As BookInfo, _
    ByVal nodeBefore As String)

    'XML-Dokument laden
```

```

Dim xmlDoc As New XmlDocument()
xmlDoc.Load(Global.BookFileName)
Dim xPathString As String = "Booklist/Book[@Title='" + _
    nodeBefore + "'""]"

'Über XPath Node auswählen
Dim actNode As XmlNode
actNode = xmlDoc.SelectSingleNode(xPathString)

'und einfügen
Dim newElement As XmlElement
newElement = XmlWorker.CreateBook(BI, xmlDoc)
xmlDoc.DocumentElement.InsertAfter(newElement, actNode)

'und speichern
xmlDoc.Save(Global.BookFileName)
End Sub

```

Listing 9.15: InsertBook()

Die Methode, mit der der aktuelle Knoten ausgewählt wird, ist `SelectSingleNode()`. Darin übergeben wird der XPath-Ausdruck, der das gewünschte Element zurückliefert, oder `Nothing` (null in C#), wenn das Element nicht existiert. Das Beispiel ist so eingerichtet, dass der gesuchte String aus der Listbox entnommen wird, die vorher mit den Titeln der Bücher in der Datei gefüllt wurde. Damit kann es eigentlich nicht passieren, dass das Element nicht existiert.

SelectSingleNode()

Das neue Element wird dann an der besagten Stelle mit Hilfe von `InsertAfter()` eingefügt. Beachten Sie, dass der Wurzelknoten von `XmlDocument` verwendet wird, auf den man über die Eigenschaft `DocumentElement` zugreifen kann. Unter diesem Wurzelknoten sind alle Buch-Knoten gespeichert.

InsertAfter()

Zwei wichtige Funktionen sind noch übrig, nämlich einmal das Löschen eines Buchs und weiterhin das Ändern der Daten eines Buchs – man könnte sich ja verschrieben haben. Dem Löschen eines Knotens dient die Methode `RemoveChild()` der Klasse `XmlNode`. Der zu löschende Knoten wird über `SelectSingleNode()` gesucht und dann gelöscht.

RemoveChild()

```

Public Shared Sub DeleteSingleBook(ByVal BookTitle As String)
    Dim xPStr As String = "Booklist/Book[@Title='" + BookTitle + _
        "'""]"

    'Laden der Datei
    Dim xmlDoc As New XmlDocument()
    xmlDoc.Load(Global.BookFileName)

    'Heraussuchen des Buch-Nodes
    Dim actNode As XmlNode = xmlDoc.SelectSingleNode(xPStr)

```

```

'Node löschen
xmlDoc.DocumentElement.RemoveChild(actNode)
'Speichern
xmlDoc.Save(Global.BookFileName)
End Sub

```

Listing 9.16: DeleteSingleBook()

Werte ändern Umfangreicher wird es bei der Methode zum Ändern eines Knotens, allerdings nicht schwieriger. Wiederum wird über `SelectSingleNode()` der gewünschte Knoten gesucht und die Werte werden geändert. Da live im Dokument gearbeitet wird, muss dieses dann nur noch gespeichert werden.

```

Public Shared Sub ChangeNode( _
    ByVal BookTitle As String, _
    ByVal BI As BookInfo)

    Dim xPStr As String = "Booklist/Book[@Title='" + BookTitle +
        "']"

    'Datei in ein XmlDocument laden
    Dim xmlDoc As New XmlDocument()
    xmlDoc.Load(Global.BookFileName)

    'Das Buch mit dem angegebenen Titel heraussuchen
    Dim actNode As XmlNode
    actNode = xmlDoc.SelectSingleNode(xPStr)

    'Wenn gefunden, Daten ändern
    If actNode.NodeType = XmlNodeType.Element Then
        Dim actElement As XmlElement
        actElement = CType(actNode, XmlElement)
        actElement.SetAttribute(Global.DescTitle, BI.Title)
        actElement.SetAttribute(Global.DescCategory, BI.Category)
        actElement.SetAttribute(Global.DescAuthor, BI.Author)
    End If

    'Die untergeordneten Knoten holen
    Dim nodeChildren As XmlNodeList = actNode.ChildNodes

    'Alle Knoten kontrollieren und ggf. ändern
    Dim node As XmlNode
    For Each node In nodeChildren
        If node.NodeType = XmlNodeType.Element Then
            If node.Name = Global.DescIsbn Then
                node.InnerText = BI.Isbn
            Else
                node.InnerText = BI.Price.ToString("#,##0.00")
            End If
        End If
    End For
End Sub

```


Next

```
'Dokument speichern  
xmlDoc.Save(Global.BookFileName)  
End Sub
```

Listing 9.17: *ChangeNode()*

Damit sind alle relevanten Methoden programmiert. Beachten Sie, dass diese Methoden alle Bestandteile der Klasse *XmlWorker* sind, d.h. aus dem Hauptformular noch aufgerufen werden müssen. Die Methoden zum Aufruf sind hier unter anderem aus Platzgründen nicht aufgeführt; es sollte aber allein aufgrund der zu übergebenen Parameter klar sein, dass diese nicht allzu schwierig zu programmieren sind. Das gesamte Programm, noch ein wenig umfangreicher als an dieser Stelle präsentiert, finden Sie auf der CD-ROM unter dem Namen *ChangeDocument.exe*.

Ein Bild des Programms zur Laufzeit zeigt Abbildung 9.11.



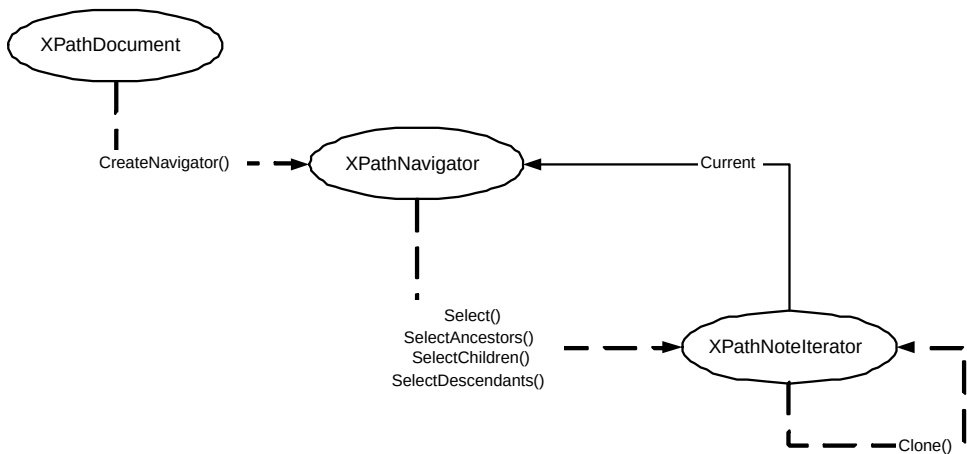
Abbildung 9.11: Ändern eines XML-Dokuments (*ChangeDocument.exe*)

9.3 XPath

Ein sehr mächtiges Instrument zur Auswahl eines oder mehrerer Knoten in einem XML-Dokument ist XPath. Das .NET-Framework bietet eine komplette XPath-Unterstützung im Namespace *System.Xml.XPath*. Die hauptsächlich benötigten Klassen sind:

- ▶ `XPathDocument`, abgeleitet von `XmlDocument` und mit der Unterstützung für XPath (wie der Name schon sagt) und XSL-Transformationen ausgerüstet
- ▶ `XPathExpression` kapselt einen XPath-Ausdruck, der auch als kompilierter Ausdruck verwendet werden kann
- ▶ `XPathNavigator` dient zum Navigieren innerhalb des Dokuments, also zum Auswählen der gewünschten Knoten
- ▶ `XPathNodeIterator` ermöglicht das Iterieren über einen Satz von Knoten, die `XPathNavigator` zurückliefert

Die Zusammenhänge zwischen den Klassen sehen Sie in Abbildung 9.12.



(C) Holger@Schwichtenberg.de

Abbildung 9.12: XPath-Klassen im .NET-Framework

IXPathNavigable Statt der Klasse `XPathDocument` kann auch jede andere Klasse verwendet werden, die die Schnittstelle `IXPathNavigable` implementiert. Unter anderem zählt auch `XmlNode` dazu.

9.3.1 XPathNodeIterator

`XPathNavigator` und `XPathNodeIterator` sind eigentlich abstrakte Klassen. Eine funktionsfähige Navigator-Klasse erhalten Sie über die Methode `CreateNavigator()`. Aus der erhaltenen Instanz können Sie mit den verschiedenen `Select`-Methoden wiederum einen `XPathNodeIterator` erzeugen, der dazu dient, die zurückgelieferten Knoten zu durchlaufen.

**XPathNode
Iterator**

Die Knoten, die durch die `Select`-Methoden zurückgeliefert werden, sind wiederum im `XPathNavigator` gespeichert. Über die Eigenschaft `Current` des `XPathNodeIterator`-Objekts greifen Sie auf den aktuellen Knoten zu, der wiederum als `XPathNavigator`-Objekt zurückgeliefert wird.

Current

Das Durchlaufen der Knoten geschieht über die Methode `MoveNext()`. Dabei ist zu beachten, dass der `XPathNodeIterator` nicht automatisch auf den ersten ausgewählten Knoten zeigt – stattdessen muss zuerst `MoveNext()` aufgerufen werden, bevor die erste Auswertung erfolgen kann. `MoveNext()` springt immer zum nächsten Knoten in der Liste und liefert `False` zurück, wenn das Ende der Liste erreicht ist.

MoveNext()

9.3.2 XPathNavigator

Am Ende einer jeden Auswertung mittels eines XPath-Ausdrucks steht also ein Objekt vom Typ `XPathNavigator`. Diese Klasse ist die Hauptklasse für XPath-Ausdrücke, da wir über sie auf die Daten zugreifen können. `XPathNavigator` besitzt die gleichen Eigenschaften zum Zugriff wie `XmlNode`:

- ▶ `HasAttributes`, kontrolliert, ob der aktuelle Knoten Attribute besitzt
- ▶ `HasChildren`, kontrolliert, ob der aktuelle Node untergeordnete Knoten besitzt
- ▶ `IsEmptyElement`, kontrolliert, ob es sich bei dem aktuellen Node um ein leeres Element handelt
- ▶ `LocalName`, liefert den Namen des Knotens ohne ein eventuelles Namespace-Präfix
- ▶ `Name`, liefert den Namen des aktuellen Elements zurück
- ▶ `NodeType`, liefert die Art des aktuellen Knotens zurück
- ▶ `Value`, liefert den Textwert des aktuellen Knotens zurück

Damit ist es sehr einfach, einen oder mehrere Knoten mittels XPath zu ermitteln.

Beispiel: Knoten mittels XPath auswerten

An dieser Stelle wird kein neues Beispiel begonnen, sondern die `XmlWorker`-Klasse aus dem Beispiel `ChangeDocument` erweitert. Am Screenshot in Abbildung 9.11 sehen Sie schon, dass es möglich sein soll, die Details eines Buchs anzuzeigen. Da in die Listbox lediglich die Titel des Buchs eingefügt wurden, müssen die Details aus dem `XmlDocument` gesucht werden. Hierzu wird XPath verwendet und ein Objekt vom Typ `BookInfo` zurückgeliefert. Die Methode ist wieder eine statische Methode der Klasse `XmlWorker`.

```
Public Shared Function GetSingleBook( _
    ByVal BookTitle As String) As BookInfo

    Dim xpStr As String = "Booklist/Book[@Title='" + _
        BookTitle + "']"

    'Ein neues XPathDocument
    Dim xpDoc As New XPathDocument(Global.BookFileName)
    'Ein neuer XPathNavigator
    Dim xpNav As XPathNavigator = xpDoc.CreateNavigator()
    'Ein neuer XPathIterator
    Dim xpIter As XPathNodeIterator = _
        xpNav.Select(xpathString)
    'Ein erstes MoveNext um zum ersten Knoten zu springen
    xpIter.MoveNext()

    'Neue Bookinfo anlegen und die ersten Werte holen
    Dim bi As New BookInfo()
    bi.Author = xpIter.Current.GetAttribute(Global.DescAuthor, _
        String.Empty)
    bi.Category =
xpIter.Current.GetAttribute(Global.DescCategory, _
        String.Empty)
    bi.Title = xpIter.Current.GetAttribute(Global.DescTitle, _
        String.Empty)

    'Neuer Iterator für untergeordnete Children
    Dim nodeChildren As XPathNodeIterator
    nodeChildren = _
        xpIter.Current.SelectChildren(XPathNodeType.Element)

    'Werte holen
    nodeChildren.MoveNext()
    bi.Isbn = nodeChildren.Current.Value.ToString()
    nodeChildren.MoveNext()
```

```
        bi.Price = _  
            Convert.ToDouble(nodeChildren.Current.Value.ToString())  
  
        Return bi  
    End Function
```

Listing 9.18: GetSingleBook()

Zuerst wird ein neues `XPathDocument` erstellt. Dann wird daraus mittels `CreateNavigator()` ein `XPathNavigator`-Objekt erzeugt und aus diesem wiederum mittels `Select()` ein `XPathNodeIterator`-Objekt. Der Aufruf von `MoveNext()` führt zum ersten Knoten der Auswahlliste, bei dem es sich auch um den gesuchten Knoten handelt.

Die untergeordneten Knoten müssen ebenfalls ausgewählt werden. Dazu wird die Methode `SelectChildren()` verwendet, die auf den aktuellen Knoten angewendet wird, bei dem es sich ja um das gesuchte Buch handelt. `SelectChildren()` liefert die Knoten für ISBN und Preis, die dann durchlaufen werden können.

Aus den Daten wird ein neues Objekt vom Typ `BookInfo` erstellt und zurückgeliefert. Jetzt können im Hauptfenster die enthaltenen Daten angezeigt werden.

9.4 XslTransform

Eine ebenfalls interessante Möglichkeit, die das .NET-Framework bietet, ist die Umwandlung bestehender XML-Daten mittels *XSL* (Extensible Stylesheet Language)-Transformationen. Die entsprechenden Klassen finden sich im Namensraum `System.Xml.Xsl`, wobei eigentlich nur die Klasse `XslTransform` relevant ist. Diese ermöglicht eine einfache Transformation mittels XSL-Stylesheets, z.B. in HTML.

Alles, was für eine solche Transformation benötigt wird, ist eine XML-Datei und ein dazu passendes Stylesheet.

Beispiel: XML transformieren

Für dieses Beispiel wird wieder die XML-Datei mit den Büchern verwendet. Diese soll nun in eine ansprechende Form gebracht werden, die im Internet angezeigt werden kann, d.h. es soll eine Transformation nach HTML durchgeführt werden. Das entsprechende Stylesheet sehen Sie in Listing 9.19.

```

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/
1999/XSL/Transform">

<xsl:template match="/">
<HTML>
  <head>
    <title>Buchliste</title>
  </head>

  <body>
    <table width="100%">
      <tr>
        <td colspan="3" bgcolor="#000080">
          <font face="arial" size="3" color="#FFFFFF">
            Buchliste
          </font>
        </td>
      </tr>

      <xsl:apply-templates/>

    </table>

  </body>
</HTML>

</xsl:template>

<xsl:template match="Booklist">
  <xsl:apply-templates Select="Book"/>
</xsl:template>

<xsl:template match="Book">
  <tr>
    <td width="100" bgcolor="#C0C0C0">
      <font face="arial" size="2" color="#000000">
        <xsl:value-of Select="@Author"/>
      </font>
    </td>
    <td width="200">
      <font face="arial" size="2" color="#000000">
        <xsl:value-of Select="@Title"/>
      </font>
    </td>
    <td width="100" bgcolor="#e0e0e0" align="right">
      <font face="arial" size="2" color="#000000">
        <xsl:value-of Select="Price" /> DM
      </font>
    </td>
  </tr>

```

```
</xsl:template>
```

```
</xsl:stylesheet>
```

Listing 9.19: Das Stylesheet zum Transformieren der Datei

Die eigentliche Routine zum Transformieren ist sehr klein, da im Prinzip nur zwei Sachen getan werden: Das Stylesheet wird in ein XslTransform-Objekt geladen und die Methode Transform() wird aufgerufen. Die Methode sehen Sie in Listing 9.20.

XslTransform

```
Private Sub btnTransform_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnTransform.Click

    Dim transform As New XslTransform()
    Dim outfile As String = Path.GetTempPath() + _
        "test.html"
    Dim p As Process

    ' --- Transformieren
    transform.Load(tbxXsl.Text)
    transform.Transform(tbxXml.Text, outfile)

    ' Im Browser anzeigen
    p = Process.Start(outfile)
End Sub
```

Listing 9.20: Transform.exe

Das Ergebnis der Routine zeigt Ihnen Abbildung 9.13

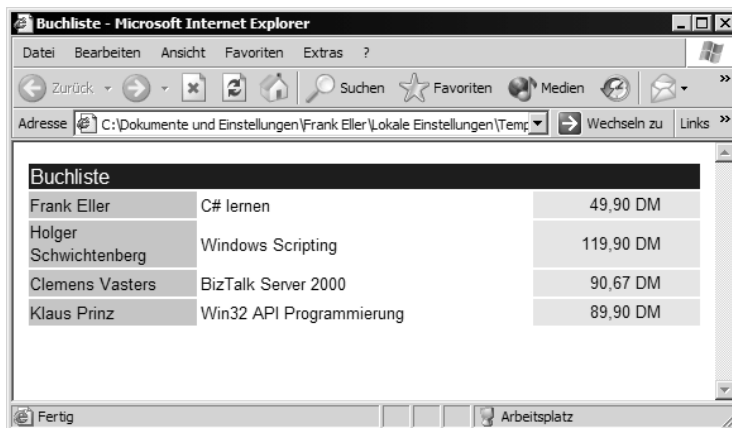


Abbildung 9.13: Das Ergebnis der Transformation (Transform.exe)

10 Namespace System.Data (ADO.NET)

Der Nachfolger der ActiveX Data Objects (ADO) heißt ADO.NET (früher: ADO+) und ist Bestandteil der Framework Class Library (FCL) in Form des Namespaces `System.Data`. Microsoft hat von ADO zu ADO.NET einige Änderungen vorgenommen, die sich sowohl auf die grundsätzliche Architektur und Arbeitsweise als auch auf das Objektmodell beziehen. Die wichtigsten Änderungen gegenüber ADO Version 2.7 sind:

**Aus ADO wird
ADO.NET**

- ▶ Eine Zweiteilung in die beiden Programmierschnittstellen OLEDB und ADO gibt es nicht mehr: ADO.NET ist eine einheitliche Schnittstelle für alle .NET-fähigen Sprachen.
- ▶ Der Zugriff auf Daten erfolgt über so genannte .NET Data Provider (auch Managed Provider).
- ▶ Die Stelle der `RecordSet`-Klasse im Objektmodell übernehmen die Klassen `DataSet`, `DataTable` und `DataReader`.
- ▶ Ein `DataSet` kann mehrere Tabellen aufnehmen und auch hierarchische Beziehungen zwischen den enthaltenen Tabellen herstellen. Ein `DataSet` ist eine Mischung aus dem relationalen und dem hierarchischen Datenmodell.
- ▶ Alle Datenzugriffe sind verbindungslos (engl. disconnected). Mit ADO.NET wird die Verbindung zur Datenquelle nach dem Einlesen der Daten zunächst getrennt. Die Daten werden »offline« d.h. ohne Verbindung zur Datenquelle bearbeitet. Später ist eine Überführung dieser Daten in die Datenquelle möglich (Batch Update). Damit gibt Architektur, Daten unabhängig und »weit« entfernt von der Datenquelle (z.B. auf der anderen Seite einer Firewall) zu bearbeiten.

**Unterschiede zu
ADO**

- XML-Integration** ► Es gibt keine explizite Unterscheidung zwischen den zwei Cursor-Typen (clientseitig oder serverseitig) mehr. Ein `DataSet` arbeitet immer mit einem clientseitigen Cursor, ein `DataReader` arbeitet immer mit einem serverseitigen Cursor.
- ADO.NET ist sehr eng mit dem Standard Extensible Markup Language (XML) verbunden. Relationale Daten können mit den Möglichkeiten von XML bearbeitet werden. Umgekehrt können XML-Daten in ein `DataSet` eingelesen werden und bearbeitet werden – selbst dann, wenn die Daten hierarchisch sind. Eine Beschränkung auf »flache« XML-Daten wie im klassischen ADO gibt es nicht mehr. Durch Verwendung einer Selbstbeschreibung in Form eines XSD-Schemas können Daten verlustfrei zwischen einem `DataSet` und einem XML-Dokument hin- und herkonvertiert werden.

Diese Integration mit XML bietet zwei Vorteile:

1. Eine Integration zwischen dem hierarchischen Datenmodell von XML und dem relationalen Datenmodell wird möglich.
2. Daten können zwischen verschiedenen Computer bzw. zwischen verschiedenen Schichten einer Anwendung per XML transportiert werden, ohne auf eine der beiden Seiten auf die Möglichkeit der Datenbearbeitung mit ADO.NET verzichten zu müssen. Das Passieren von Firewall wird damit im Gegensatz zum klassischen ADO, das COM-Marshalling verwendet, problemlos.

10.1 Architektur

In ADO.NET gibt es eigene Provider für Datenquellen, die .NET Data Provider genannt werden (In der Beta1-Version hießen diese Provider noch Managed Provider).

.NET Data Provider Derzeit gibt es nur drei .NET Data Provider:

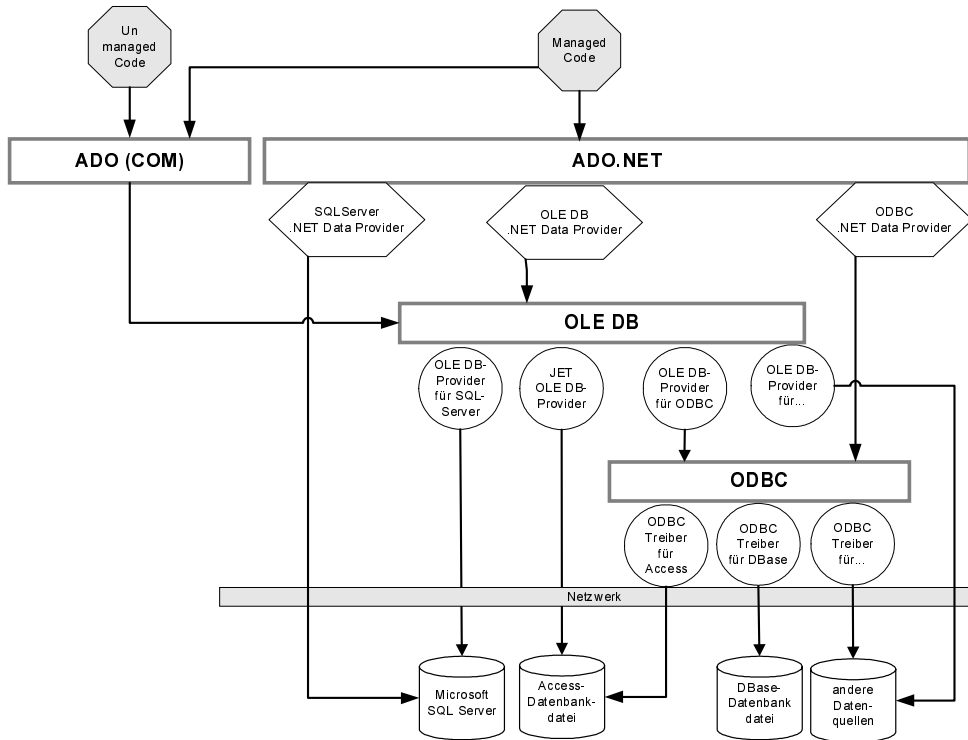
- **SQL Data Provider** (Namespace `System.Data.SqlClient`): Dieser Provider kann nur mit dem Microsoft SQL Server 7.0 und dem Microsoft SQL Server 2000 verwendet werden. Er greift direkt auf die Programmierschnittstellen des SQL-Servers zu und verwendet nicht OLEDB.

- ▶ **OleDb Data Provider** (Namespace `System.Data.OleDb`): Dieser Provider setzt auf OleDb auf und ermöglicht die Verwendung aller Datenquellen, für die es einen OleDb-Provider gibt.
- ▶ **ODBC Data Provider** (Namespace `System.Data.Odbc`): Dieser Provider setzt auf ODBC auf und ermöglicht die Verwendung aller Datenquellen, für die es einen ODBC-Treiber gibt. Dieser Provider ist nicht Teil des Standardinstallationsumfangs des .NET Frameworks. Er muss zusätzlich installiert werden [CD:/Install/DOTNET Add ons/ODBC Data Provider].
- ▶ **Der SQLXML Data Provider** (Namespace `System.Data.SqlXml`) für den Zugriff auf die SQL-Erweiterungen im Microsoft SQL Server 2000 [CD:/Install/DOTNET Add ons/SQLXML].
- ▶ **Den Data Provider for Oracle** [CD:/Install/DOTNET Add ons/Oracle/Data Provider].

In Zukunft wird es weitere .NET Data Provider geben, auch von anderen Anbietern. Die Firma Datadirect Technologies hat unter dem Namen DataDirect Connect .NET verschiedene .NET Data Provider angekündigt [DDT02].

Die folgende Abbildung zeigt die – inzwischen sehr komplex gewordene – Gesamtarchitektur der Universal Data Access (UDA)-Strategie von Microsoft. Mit den ODBC-Treibern, den OleDb-Providern und den .NET Data Providern gibt es inzwischen drei Arten von Datenbanktreibern von Microsoft. Abwärtskompatibilität ist gegeben. Allerdings muss man natürlich mit Performance-Verlusten rechnen, je mehr Schichten man in den Datenzugriff einbezieht. Von Managed Code aus auf eine Access-Datenbank mittels ODBC-Treiber über den Weg *OleDb .NET Data Provider -> OleDb-Provider für ODBC -> ODBC-Treiber* für Access zuzugreifen, ist sicherlich nicht der richtige Weg.

**Universal Data
Access**



(C) Holger@Schwichtenberg.de 2002

Abbildung 10.1: Architektur von ADO.NET

10.2 Objektmodell

Der Namespace `System.Data` enthält verschiedene Unternamespaces:

- OLEDB** ► `System.Data.OleDb` enthält Klassen für den .NET Data Provider für OLEDB. Die darin enthaltenen Klassen beginnen mit dem Namenspräfix »OleDb«.
- SQL Server** ► `System.Data.SqlClient` enthält Klassen für den .NET Data Provider für Microsoft SQL Server. Die darin enthaltenen Klassen beginnen mit dem Namenspräfix »Sql«.
- `System.Data.Common` enthält abstrakte Basisklassen für alle .NET Data Provider.
- `System.Data` enthält allgemeine Klassen für die Arbeit mit Daten.

Änderungen gegenüber ADO

Das Objektmodell in ADO.NET hat sich gegenüber den klassischen ActiveX Data Objects (ADO) komplett geändert. Während in ADO die Klasse `RecordSet` im Mittelpunkt stand, gibt es nun verschiedene Klassen im Zusammenhang mit Tabellen:

- ▶ Ein ADO.NET-**DataReader** ist eine sehr performante Klasse zum Vorwärts-Lesen von Daten (read-only/forward-only). Jeder .NET Data Provider implementiert eine eigene **DataReader**-Klasse. In der FCL werden die Klassen `System.Data.OleDb.OleDb.Data Reader` und `System.Data.SqlClient.SqlDataReader` (optional: `System.Data.SqlClient.OdbcReader`) mitgeliefert. **DataReader**
- ▶ Ein **DataSet** (Klasse `System.Data.DataSet`) ist eine (veränderbare) Menge von Tabellen mit Daten. Die Tabellen können in einem **DataSet** miteinander verknüpft werden. **DataSet**
- ▶ Eine **DataTable** (Klasse `System.Data.DataTable`) ist eine einzelne Tabelle in einem **DataSet**. **DataTable**

Die folgende Tabelle zeigt weitere Unterschiede zwischen ADO und ADO.NET.

Klassen in ADO 2.7 (COM)	Klassen in ADO.NET
RecordSet	OleDbDataReader (SqlDataReader) DataTable DataSet
Connection	OleDbConnection (SqlConnection)
Command	OleDbCommand (SqlCommand)
(keine Entsprechung)	OleDbDataAdapter (SqlDataAdapter)
Field	DataColumn
(keine Entsprechung)	DataRow
Stream	System.IO.Stream

Tabelle 10.1: ADO vs. ADO.NET

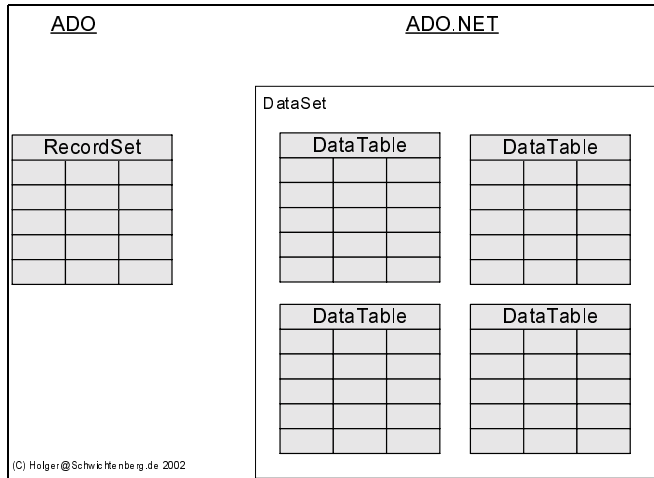


Abbildung 10.2: RecordSet vs. DataSet

Objekthierarchie in System.Data.OleDb und System.Data.SqlClient

System.Data.OleDb

Die folgende Abbildung zeigt das Objektmodell für den Namespace System.Data.OleDb. Der erste Teil stellt den Zusammenhang zwischen OleDbConnection, OleDbCommand, OleDbDataAdapter, OleDbDataReader und DataSet dar. Der Namespace System.Data.SqlClient hat ein analoges Objektmodell; der Unterschied besteht lediglich darin, dass die Namen mit »Sql« statt mit »OleDb« beginnen.

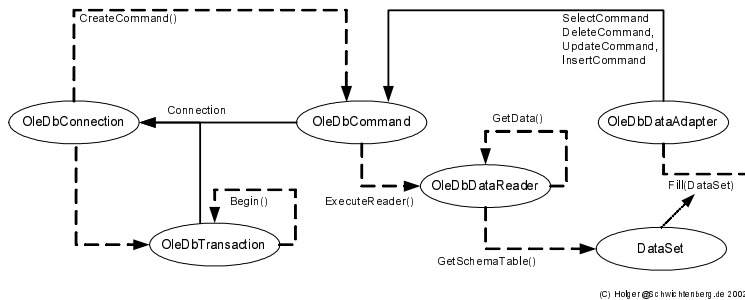


Abbildung 10.3: Objekthierarchie in System.Data.OleDb

Objekthierarchie in System.Data

System.Data

Die folgende Abbildung zeigt die Objekthierarchie aus System.Data. In der Abbildung wurden einige weniger wichtige Klassen und Querverbindungen weggelassen, damit die Abbildung nicht zu unübersichtlich wird.

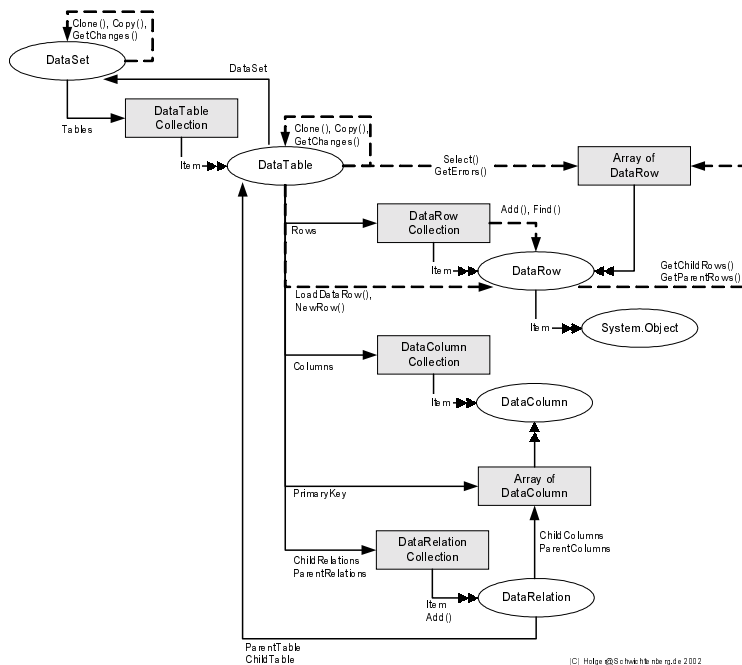


Abbildung 10.4: Objekthierarchie in System.Data

Änderungen seit der Beta 1

In dem Namespace `System.Data` gab es zahlreiche Änderungen zwischen der Beta1 und der Beta2 des .NET Framework. Der Namespace `System.Data.OleDb` hieß in Beta1 noch `System.Data.ADO`. Die folgende Tabelle stellt die alten und neuen Klassennamen gegenüber. Wenn Sie im Internet oder anderen Quellen Beispiele sehen, die diese alten Klassennamen verwenden, müssen Sie die Beispiele zunächst anpassen.

Änderungen seit Beta1

ADO.NET: Heutiger Klassenname (seit Beta2)	ADO.NET: Klassenname in Beta1
OleDbCommand	ADODCommand
OleDbConnection	ADODConnection
OleDbDataReader	ADODDataReader
OleDbDataAdapter	ADODDataSetCommand
OleDbParameterCollection	ADODParameters
OleDbParameter	ADODParameter

Tabelle 10.2: ADO.NET-Klassennamen in Beta1 und späteren Versionen

10.3 Hinweis zu den Beispielen

Access-Datenbank

Alle Beispiele in diesem Kapitel beziehen sich auf die Microsoft Access-Datenbank *EDVShop.mdb*, die auf der CD zu diesem Buch mitgeliefert wird [CD:/Code/FCL-Buch/_Daten/adonet/EDVShop.mdb]. Die Datenbank hat das Format Access 10 (Access XP).

Inhalt der Datenbank

Die folgende Grafik zeigt die Tabellen und deren Beziehung in der Datenbank *EDVShop.mdb*.

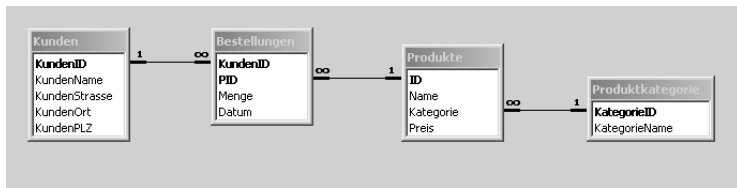


Abbildung 10.5: Datenmodell der Datenbank »EDVShop.mdb«

	ID	Name	Kategorie	Preis
+	1	Commodore C64	23	500,00 €
+	2	Mauspad	10	20,60 €
+	3	Amiga 500	23	600,00 €
+	4	IBM 286er / 2 MB	23	300,00 €
+	5	IBM 286er / 4 MB	23	350,00 €
+	6	Amiga 1000	23	800,00 €
+	7	Drucker, 9 Nadeln NoName	23	50,00 €
+	8	MegaMonitor 14 Zoll	23	25,00 €
+	9	Farbband für 9-Nadel-Drucker	23	5,00 €
+	14	Palm Pilot M505	10	520,00 €
►	(AutoWert)		0	0,00 €

Datensatz: 11 von 11

Abbildung 10.6: Inhalt der Tabelle »Produkte«

Verbindungszeichenfolge

Connection String

Als Data Provider kommt der OLEDB .NET Data Provider zum Einsatz, der über den JET 4.0 OLEDB auf die Access-Datenbankdatei zugreift. Als Verbindungszeichenfolge (Connection String) kommt für alle Beispiele daher folgende Zeichenkette zum Einsatz:


```
Public Const CONNSTRING As String =
    "Provider=Microsoft.Jet.OLEDB.4.0;
    Data Source=..\_DATEN\adonet\EDVShop.mdb;"
```

Diese Konstante ist innerhalb des Projekts »FCL-Buch« in der Datei ADONET.vb definiert. Die Datenbank liegt innerhalb des Projekts im Unterverzeichnis _Daten. Bitte passen Sie die Verbindungszeichenfolge an, wenn Sie die Datenbank verschieben oder eine andere Datenbank nutzen wollen.



Import-Anweisungen

Den folgenden Beispielen sind jeweils diese Imports-Anweisungen voranzustellen:

Namespaces

```
Imports System.Data
Imports System.Data.OleDb
```

Von Access zum SQL Server

Aus Platzgründen kann in diesem Buch nur ein .NET Data Provider beschrieben werden. Die Wahl fiel daher auf den OLEDB Data Provider, weil dieser universell einsetzbar ist. Er kann auch zum Zugriff auf Microsoft SQL Server-Datenbanken verwendet werden. Die Verwendung des SQL Server Data Providers wäre hier nur effizienter. Die Beispiele lassen sich in vielen Fällen fast identisch auf den SQL Server .NET Data Provider übertragen. Es müssen lediglich die Klassennamen mit der Vorsilbe »OleDb« aus dem Namespace `System.Data.OleDb` durch die Klassennamen mit der Vorsilbe »Sql« aus dem Namespace `System.Data.SqlClient` ersetzt werden. Und die Verbindungszeichenfolge muss natürlich geändert werden.

**Migration zu
System.Data.
SqlClient**

Um eine Datenbank über einen ODBC-Treiber zu nutzen, muss der Namespace `System.Data.OdbcClient` verwendet und die Vorsilbe »OleDb« durch »Odbc« ersetzt werden.

System.Data.Odbc

10.4 Daten-Verbindungen

Sehr ähnlich sind sich ADO und ADO.NET in Bezug auf die Kommunikation zwischen der Datenzugriffs-Komponente und der Datenquelle. Sowohl ADO als auch ADO.NET nutzen dazu das Konzept einer Verbindung, die zu einer (passiven) Datenbankdatei (z.B. einer Microsoft Access Datenbank `.mdb`) oder einem (aktiven) Datenbankserverprozess einer Client/Server-Daten-

**Verbindungs-
zeichenfolgen**

bank vor dem ersten Zugriff aufgebaut wird. Im ersten Fall entspricht die Verbindung einem Zugriff auf die Datei, im letzteren einer Netzwerkverbindung. Zur Spezifikation der Datenquelle werden so genannte Verbindungszeichenfolgen (engl. Connection Strings) verwendet.

Verbindungs- aufbau

Durch den Verbindungsaufbau wird eine Sitzung eingerichtet. Innerhalb einer Sitzung können eine oder mehrere Datenbankoperationen ausgeführt werden. Nach dem Ende der Nutzung der Datenquelle sollte die Verbindung unbedingt geschlossen werden, um die von der Sitzung benötigten Ressourcen freizugeben.



Der OLEDB Data Provider und der SQL Server Data Provider unterstützen Connection Pooling, d.h. dass zwei Sitzungen, die mit der exakt gleichen Verbindungszeichenfolge aufgebaut werden, auf ein und dieselbe Sitzung abgebildet werden. Auch werden Sitzungen nicht sofort nach dem Schließen mit Close() abgebaut, sondern noch eine Zeit lang aufrechterhalten für den Fall, dass es neue Verbindungswünsche gibt.

OleDbConnection

Die Änderungen beim Verbindungs-Handling zwischen ADO und ADO.NET sind – im Vergleich zu allen anderen Klassen – gering:

- ▶ Der Klassenname hat sich von `ADODB.Connection` zu `System.Data.OleDb.OleDbConnection` geändert.
- ▶ Die Verbindungszeichenfolge kann nun per Konstruktor übergeben werden.
- ▶ Die Methoden `Open()` und `Close()` existieren weiterhin. `Open()` besitzt in ADO.NET aber im Gegensatz zu ADO keine Parameter mehr.

Beispiel

Datenbankverbin- dung herstellen

Die folgende Routine baut eine Verbindung zu einer Datenbank auf und gibt dann einige Informationen über diese Verbindung aus. Danach wird die Verbindung wieder geschlossen. Die Attribute `Provider`, `DataSource`, `ServerVersion` und `Database` sind dabei Extrakte der übergebenen Verbindungszeichenfolge. Die komplette Verbindungszeichenfolge steht im Attribut `ConnectionString`.

```
' =====
' Beispiel aus FCL-Buch.sln
' Auf- und Abbau einer OLEDB-Verbindung
' (C) Holger@Schwichtenberg.de
' =====
Sub ADONET_Connect()
    Dim Conn As OleDbConnection
```

```

Dim Com As OleDbCommand
Dim DR As OleDbDataReader

outtitle("OLEDB-Verbindung testen")

' --- Verbindung aufbauen
Conn = New OleDbConnection(CONNSTRING)
Conn.Open()

' --- Informationen über die Verbindung ausgeben
out("OLEDB Provider: " & _
    Conn.Provider)
out("Datenquelle: " & Conn.DataSource)
out("Serverversion: " & Conn.ServerVersion)
out("Datenbank: " & Conn.Database)
out("Verbindungszeichenfolge: " & _
    Conn.ConnectionString)
out("Timeout-Zeit: " & Conn.ConnectionTimeout)
out("Verbindungsstatus: " & Conn.State)
out(Conn.ToString)

' --- Verbindung schließen
Conn.Close()

out("Verbindung beendet!")
End Sub

```

Listing 10.1: Auf- und Abbau einer OLEDB-Verbindung [ADONET_DataReader.vb]

Ausgabe

Im Fall einer Access-Datenbank enthält das Attribut `DataSource` den Pfad zur Datenbankdatei, `ServerVersion` die Version des OLEDB-Providers und `Database` ist leer. **Ergebnis**

```

1  OLEDB-Verbindung testen
2  OLEDB Provider: Microsoft.Jet.OLEDB.4.0
3  Datenquelle: D:\data\adonet\EDVShop.mdb
4  Serverversion: 04.00.0000
5  Datenbank:
6  Verbindungszeichenfolge:
Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=D:\data\adonet\EDVShop.mdb;
7  Timeout-Zeit: 15
8  Verbindungsstatus: 1
9  System.Data.OleDb.OleDbConnection
10 Verbindung beendet!

```

10.5 Daten lesen mit dem OleDbDataReader

DataReader Ein `DataReader` ist eine besonders effiziente Klasse zum Lesen von Daten aus (OLEDB-)Datenquellen. Zur Steigerung der Performance unterliegt ein `DataReader` folgenden Beschränkungen:

- ▶ Daten können nur gelesen werden.
- ▶ Daten können nur vorwärts (d.h. nur einmal) gelesen werden.



Damit entspricht ein `DataReader` einem `RecordSet` vom Typ `read-only/forward-only` aus dem klassischen ADO.

Vorgehensweise

Daten lesen mit dem `DataReader`

Um Daten mit einem `DataReader` zu lesen, sind folgende Schritte notwendig:

- ▶ Aufbau einer Verbindung zu der Datenbank mit einem `OleDbConnection`-Objekt. Bei der Instanziierung dieses Objekts kann die Verbindungszeichenfolge übergeben werden.
- ▶ Instanziierung der Klasse `OleDbCommand` und Bindung dieses Objekts an das `OleDbConnection`-Objekt über die Eigenschaft `Connection`
- ▶ Festlegung eines SQL-Befehls im `OleDbCommand`-Objekt in der Eigenschaft `CommandText`

Beispiel

**Produkttable
ausgeben**

Die folgende Routine listet aus der Tabelle »Produkte« alle Produktdatensätze auf, die zur Kategorie 23 gehören.

```
' =====
' Beispiel aus FCL-Buch.sln
' Ausgabe einer Tabelle mit Hilfe eines DataReaders
' (C) Holger@Schwichtenberg.de
' =====
Sub ADONET_DR1()
    Dim Conn As OleDbConnection
    Dim Com As OleDbCommand
    Dim DR As OleDbDataReader

    ' --- Verbindung aufbauen
    Conn = New OleDbConnection(CONNSTRING)
    Conn.Open()

    ' --- Befehl definieren
```

```

Com = New OleDbCommand()
Com.Connection = Conn
Com.CommandText = _
"SELECT * FROM Produkte WHERE Kategorie = 23"

' --- Befehl ausführen
DR = Com.ExecuteReader()

' --- Daten lesen
While DR.Read() = True
    out(DR.Item("Name").ToString & _
        " (Produkt #" & DR.Item("ID").ToString & ") " & _
        " kostet " & _
        DR.Item("Preis").ToString & " _")
End While

' --- Schließen
DR.Close()
Conn.Close()
End Sub

```

Listing 10.2: Ausgabe einer Tabelle mit Hilfe eines DataReaders
[ADONET_DataReader.vb]

Ausgabe

Das obige Beispiel liefert folgende Ausgabe:

```

1 Commodore C64 (Produkt #1) kostet 500 _
2 Amiga 500 (Produkt #3) kostet 800 _
3 IBM 286er / 2 MB (Produkt #4) kostet 1300 _
4 IBM 286er / 4 MB (Produkt #5) kostet 1500 _
5 Amiga 1000 (Produkt #6) kostet 2000 _
6 Drucker, 9 Nadeln NoName (Produkt #7) kostet 4999 _
7 MegaMonitor 14 Zoll (Produkt #8) kostet 2300 _

```

10.6 DataSets

Ein DataSet-Objekt ist eine Menge von Tabellen (Relationen) im Hauptspeicher. Dieses Konzept existierte in ADO noch nicht. Dort gab es nur RecordSet-Objekte, die jeweils genau eine Tabelle (im relationalen Datenbankmodell auch Relation genannt – nicht zu verwechseln mit dem englischen Begriff relation, der Verknüpfung bedeutet) repräsentiert haben. Ein DataSet ist aber nicht die Zusammenfassung mehrerer RecordSet-Objekte. Die einzelnen Tabellen werden nicht mehr durch RecordSet-, sondern durch DataTable-Objekte repräsentiert.

DataSet

Verknüpfungen Zwischen `DataTable`-Objekten in einem `DataSet` können Verknüpfungen definiert werden. Die Verknüpfungen werden dann nicht wie Joins im relationalen Datenbankmodell als Tabellen, sondern durch hierarchische Eltern-Kind-Beziehungen dargestellt.

Batch Update Ein `DataSet` erlaubt alle Zugriffsarten. Es ist disconnected, d.h. die Daten aus der Datenquelle werden komplett in das `DataSet` eingelesen, im Speicher gehalten und erst zu einem definierbaren Zeitpunkt werden die kompletten Änderungen zurück an die Datenquelle gegeben (Batch Update).



Für das Einfügen, Ändern oder Löschen einzelner Datensätze sollte man kein `DataSet` verwenden. Für diese Fälle sollte man mit der `OleDbCommand`-Klasse einen SQL-Befehl an die Datenquelle senden.

10.6.1 Data Tables

DataTable vs. RecordSet Ein `DataTable`-Objekt repräsentiert genau eine Tabelle mit `n` Spalten und `m` Zeilen. Ein `DataTable`-Objekt ist aber nicht der direkte Nachfolger eines `RecordSet`-Objekts. Zwar repräsentiert `DataTable` eine Tabelle, jedoch benötigt ein `DataTable` zwingend die Hilfe anderer Klassen (`DataSet`, `OleDbDataAdapter`), um Daten aus einer Datenquelle einlesen bzw. ändern zu können. Ein `DataTable`-Objekt kann aber auch ohne Datenquelle dynamisch im Hauptspeicher erzeugt werden.

Bewegung durch die Datensätze

DataTable Die Arbeit mit einem `DataTable`-Objekt ist wesentlich objektorientierter als die Arbeit mit einem `RecordSet`-Objekt aus dem klassischen ADO.

Vergleich zu ADO Ein `RecordSet`-Objekt ist einerseits eine ganze Tabelle, aber andererseits zu einem bestimmten Zeitpunkt immer nur genau ein Datensatz – ähnlich wie beim `DataReader`. In einem `RecordSet` gibt es eine Objektmenge zum Durchlaufen der Spalten (`Fields`-Collection); die Iteration über die Zeilen erfolgt aber nicht objektorientiert, sondern mit `MoveNext()` und Prüfung auf das Tabellenende (EOS).

Rows Eine `DataTable` in ADO.NET verweist dagegen im Attribut `Rows` auf eine Collection vom Typ `DataRowCollection`, mit der die Zeilen der Tabelle (mit `For Each`) durchlaufen werden können.

```
' --- Schleife über alle Datensätze
For Each DR In DT.Rows
Next
```

Eine einzelne Zeile besitzt keinen Namen, kann aber über einen numerischen Index direkt angesprochen werden. Der Index beginnt bei 0 (Zeile 1) und läuft dementsprechend bis ZEILEN-ANZAHL-1.

Index

```
' --- Zugriff auf die 3. Zeile
DR = DT.Rows.Item(2)
```

Die Anzahl der Datensätze in einer Tabelle kann über das Attribut Count ermittelt werden.

Count

```
Anzahl = DT.Rows.Count
```

Spaltennamen

Zur Ermittlung der Spaltennamen gibt es im DataTable-Objekt eine Collection Columns vom Typ DataColumnCollection mit einzelnen DataColumn-Objekten.

DataColumn

```
' --- Ermittlung einer CDV-Liste der Spaltennamen
For Each DC In DT.Columns
    zeile += IIf(zeile <> "", ";", "").ToString
    zeile += DC.ColumnName
Next
```

Die Collection DataTable.Columns entspricht der Fields-Collection aus dem klassischen ADO.

Tabelleninhalte

Der Zugriff auf die Inhalte erfolgt über das DataRow-Objekt. Dieses Objekt liefert über das indizierte Attribut Item() den Inhalt einer bestimmten Spalte.

Syntaxformen

```
DataRow.item("SpaltenName")
```

Genau wie im klassischen ADO kann dieser Ausdruck in zwei Formen verkürzt werden:

```
DataRow!SpaltenName
DataRow("SpaltenName")
```

Außerdem ist nicht nur der Zugriff über den Spaltennamen, sondern auch über einen numerischen Index (Spalte 1 hat den Index 0) möglich.

```
DataRow.item(INDEX)
DataRow(INDEX)
```

Datentyp-Herausforderungen

Datentypfehler

Wenn Sie die Option `Strict` in VB.NET verwenden, dann werden Sie aus der klassischen ADO-Programmierung mit VB6 bisher nicht bekannte Datentypprobleme bekommen, weil der Rückgabertyp von `DataRow.Item()` allgemein als `System.Object` definiert ist.

Zum Beispiel führt

```
say(DR("Name") & " kostet " & DR("Preis"))
```

zum Fehler

Option Strict disallows operands of Type Object for operator '&'

und die Befehlsfolge

```
Dim Alterpreis As Double
Alterpreis = DR("Preis")
```

führt zum Fehler

Option Strict disallows implicit conversions from System.Object to System.Double.

Der Grund dafür ist, dass die `Items()`-Collection der `DataRow`-Klasse eine Menge von Objekten des allgemeinen Typs `System.Object` ist, damit diese alle möglichen Werte aufnehmen kann (`Object` ist das, was in früheren Visual Basic-Versionen der Datentyp `Variant` war). Die Option `Strict` verbietet aber, dass ein `Variant` automatisch in einen anderen Typ konvertiert wird.

Lösung

Wenn man Option `Strict` nicht abschalten will, müssen die obigen Beispiele mit expliziten Datentypkonvertierungen programmiert werden:

```
say(DR("Name").ToString & _
    " kostet " & CStr(DR("Preis")))
```

und

```
Dim Alterpreis As Double
Alterpreis = CDb1(DR("Preis")).
```



Für die Ausführung der Konvertierung gibt es unterschiedliche Möglichkeiten:

- ▶ die spracheigenen Typkonvertierungsfunktionen, z.B. `CStr()`, `CInt()`, `CLng()` etc. in VB.NET
- ▶ die Methode `ToString()` in `System.Object`
- ▶ die Methoden der Klasse `System.Convert`

Hilfsroutinen zur Ausgabe

Die folgende Routine packt alle Spalten und Zeilen einer Tabelle `table_to_csv()` zusammen in einen CSV-String.

```
' ### Wandelt eine Tabelle In CSV-Daten um
Function table_To_csv(ByVal DT As DataTable) As String
    Dim zeile As String = ""
    Dim DR As DataRow
    Dim DC As DataColumn

    ' --- Spaltenüberschriften
    For Each DC In DT.Columns
        zeile += IIf(zeile <> "", ";", "").ToString
        zeile += DC.ColumnName
    Next
    table_To_csv += zeile & chr(13)

    ' --- Ausgabe aller Zeilen
    For Each DR In DT.Rows
        Zeile = ""
        ' Schleife über alle Spalten
        For Each DC In DT.Columns
            zeile += IIf(zeile <> "", ";", "").ToString
            zeile += DR.item(DC.ColumnName).ToString
        Next
        table_To_csv += zeile & chr(13)
    Next

End Function
```

Listing 10.3: Umwandlung einer DataTable in CSV-Form [ADONET_Hilfsroutinen.vb]

10.6.2 Datenadapter

Die Vorgehensweise bei der Arbeit mit `DataSet`- bzw. `DataTable`-Objekten unterscheidet sich von der Arbeit mit `RecordSets` im klassischen ADO insbesondere dadurch, dass zum Einlesen der Daten eine weitere Klasse, ein so genannter Datenadapter (z.B. `OleDbDataAdapter` oder `SqlDataAdapter`), nötig ist. Der Datenadapter ist in jedem Data Provider spezifisch für die jeweilige Datenquelle implementiert.

Ein `OleDbDataAdapter`-Objekt ist dafür zuständig, ein SQL-State-ment oder eine Stored Procedure, die Daten zurückgibt, auf einer durch eine Verbindungszeichenkette spezifizierten Datenquelle auszuführen. Nach erfolgreicher Ausführung kann ein `OleDbData`

**OleDbData
Adapter**

Adapter-Objekt mit Hilfe der Methode `Fill()` die Ergebnismenge unter einem definierbaren Tabellennamen in einem zuvor instanziierten `DataSet`-Objekt speichern.

Der Konstruktor der Klasse `OleDbDataAdapter` erwartet neben einem Befehl auch entweder eine Verbindungszeichenfolge oder ein `ADOConnection`-Objekt.

```
OleDbDataAdapter(SQL, connString|objConnection)
```

Fill() Die Syntax von `Fill()` lautet:

```
objDataSetCommand.Fill(objDataSet, Name_fuer_Tabelle)
```

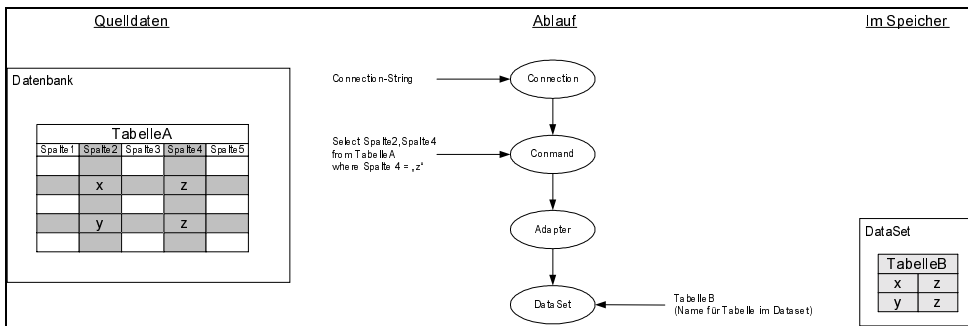


Abbildung 10.7: Ablauf des Einlesens einer Quelltable in ein DataSet

10.6.3 Lesezugriff auf Daten

Schritte zur Arbeit mit einem DataSet

Um auf eine Datenbanktabelle mit einem `DataSet`-Objekt lesend zuzugreifen, sind folgende Schritte notwendig:

- ▶ Aufbau einer Datenbankverbindung mit einem `OleDbConnection`-Objekt
- ▶ Instanziierung eines `DataSet`-Objekts
- ▶ Instanziierung eines `OleDbDataAdapter`-Objekts unter Verwendung des `OleDbConnection`-Objekts und eines SQL-Befehls
- ▶ Ausführen der Abfrage und Füllen des `DataSet`-Objekts mit der Ergebnismenge durch die Methode `Fill()`
- ▶ Auslesen der Daten
- ▶ Schließen der Verbindung (`Close()`)

Alternativ zur expliziten Instanziierung eines `OleDbConnection`-Objekts kann eine Verbindungszeichenfolge auch bei der Instanziierung des `OleDbDataAdapter`-Objekts übergeben werden. Dann besteht die Verbindung nur für einen Befehl. Der Data Adapter baut dann die Verbindung erst mit Ausführung der `Fill()`-Methode auf und direkt danach wieder ab.



Beispiel I

Das folgende einfache Beispiel zeigt das Einlesen einer einzelnen Tabelle in einem `DataSet`. Datenquelle ist die auf der Buch-CD enthaltene Access-Datenbank `EDVShop.mdb`. Aus allen Datensätzen aus der Quelltable »Produkte«, die zur Produktkategorie 23 gehören (`»SELECT * FROM Produkte WHERE kategorie=23«`), wird im `DataSet` eine `DataTable` mit dem Namen »EDV-Antik« erzeugt.

**Produkttable
einlesen**

Diese Tabelle wird im Folgenden in drei Formen ausgegeben: Zunächst durchläuft das Programm alle Datensätze über die `DataRow`-Collection und gibt nur die Spalten »Name« und »Preis« aus. Anschließend wird die komplette Tabelle (einschließlich der Spaltenüberschriften) als CSV-Liste durch die Hilfsroutine `table_to_csv()` ausgegeben. Als Drittes erfolgt die Ausgabe als XML-Dokument durch die Methode `GetXml()` des `DataSet`-Objekts.

**Verschiedene
Ausgaben**

```
' =====
' Beispiel aus FCL-Buch.sln
' Ausgabe einer Tabelle mit Hilfe eines DataSet-Objekts
' (C) Holger@Schwichtenberg.de
' =====
Sub ADONET_DS1()
    Dim DS As System.Data.DataSet
    Dim Com As System.Data.OleDb.OleDbCommand
    Dim SQL As String
    Dim DR As DataRow
    Dim DT As DataTable

    outtitle("Produktliste mit DataSet ausgeben")

    ' --- DataSet erzeugen
    DS = New System.Data.DataSet()

    ' --- Einlesen vorbereiten
    SQL = "SELECT * FROM Produkte WHERE kategorie=23"
    Dim DA As System.Data.OleDb.OleDbDataAdapter
    DA = New System.Data.OleDb.OleDbDataAdapter(SQL, _
        CONNSTRING)
```

```

' --- Tabelle lesen
DA.Fill(DS, "EDV-Antik")

' --- Ausgabe #1
out("### Preisliste")

' Zugriff auf Tabelle
DT = DS.Tables("EDV-Antik")

' Anzahl der Datensätze ausgeben
out("Anzahl Produkte: " & DT.Rows.Count)

' Schleife über alle Datensätze
For Each DR In DT.Rows
    out(DR("name").ToString & " kostet " & _
        DR("Preis").ToString)
    ' Zwei Alternativen:
    'out(DR!name.ToString & " kostet " & _
    'DR!Preis.ToString)
    'out(DR.Item("name").ToString & " kostet " & _
    'DR.Item("Preis").ToString)
Next

' --- Ausgabe #2
out("### Inhalt der Tabelle als CSV:")
out(table_To_csv(DS.Tables("EDV-Antik")))

' --- Ausgabe #3
out("### Inhalt der Tabelle als XML:")
out(DS.GetXml)

End Sub

```

Listing 10.4: Ausgabe einer Tabelle mit Hilfe eines DataSet-Objekts [adonet.vb]

Ausgabe

```

1 Produktliste mit DataSet ausgeben
2 ### Preisliste
3 Anzahl Produkte: 7
4 Commodore C64 kostet 665,5
5 Amiga 500 kostet 1064,8
6 IBM 286er / 2 MB kostet 1730,3
7 IBM 286er / 4 MB kostet 1996,5
8 Amiga 1000 kostet 2662
9 Drucker, 9 Nadeln NoName kostet 6653,67
10 MegaMonitor 14 Zoll kostet 3061,3
11 ### Inhalt der Tabelle als CSV:
12 ID;Name;Kategorie;Preis
1;Commodore C64;23;665,5
3;Amiga 500;23;1064,8

```

4;IBM 286er / 2 MB;23;1730,3
5;IBM 286er / 4 MB;23;1996,5
6;Amiga 1000;23;2662
7;Drucker, 9 Nadeln NoName;23;6653,67
8;MegaMonitor 14 Zoll;23;3061,3

13 ### Inhalt der Tabelle als XML:

```
14 <NewDataSet>
    <EDV-Antik>
        <ID>1</ID>
        <Name>Commodore C64</Name>
        <Kategorie>23</Kategorie>
        <Preis>665.5</Preis>
    </EDV-Antik>
    <EDV-Antik>
        <ID>3</ID>
        <Name>Amiga 500</Name>
        <Kategorie>23</Kategorie>
        <Preis>1064.8</Preis>
    </EDV-Antik>
    <EDV-Antik>
        <ID>4</ID>
        <Name>IBM 286er / 2 MB</Name>
        <Kategorie>23</Kategorie>
        <Preis>1730.3</Preis>
    </EDV-Antik>
    <EDV-Antik>
        <ID>5</ID>
        <Name>IBM 286er / 4 MB</Name>
        <Kategorie>23</Kategorie>
        <Preis>1996.5</Preis>
    </EDV-Antik>
    <EDV-Antik>
        <ID>6</ID>
        <Name>Amiga 1000</Name>
        <Kategorie>23</Kategorie>
        <Preis>2662</Preis>
    </EDV-Antik>
    <EDV-Antik>
        <ID>7</ID>
        <Name>Drucker, 9 Nadeln NoName</Name>
        <Kategorie>23</Kategorie>
        <Preis>6653.67</Preis>
    </EDV-Antik>
    <EDV-Antik>
        <ID>8</ID>
        <Name>MegaMonitor 14 Zoll</Name>
        <Kategorie>23</Kategorie>
        <Preis>3061.3</Preis>
    </EDV-Antik>
</NewDataSet>
```

Hilfsroutinen zum Aufbau eines DataSet-Objekts

FillDataSet() Es bietet sich an, das Einlesen der Daten und die Aufnahme in das DataSet in eine Hilfsroutine zu kapseln. Die Hilfsroutine heißt einfach `AddToDS()`. Um die beiden möglichen Fälle der Übergabe der Datenquellen-Informationen (Verbindungszeichenfolge oder `OleDbConnection`-Objekt) abzubilden, bietet sich eine Überladung der Hilfsroutine an.

```
' ### Hinzufügen einer Tabelle zu einem DataSet
' (unter Angabe eines Connection-Strings)
Overloads Sub AddToDS(ByVal DS As DataSet, _
    ByVal Name As String, ByVal Conn As String, _
    ByVal SQL As String)

    Dim DA As OleDbDataAdapter
    DA = New OleDbDataAdapter(SQL, Conn)
    DA.Fill(DS, Name)
End Sub

' ### Hinzufügen einer Tabelle zu einem DataSet
' (unter Angabe eines Connection-Objekts)
Overloads Sub AddToDS(ByVal DS As DataSet, _
    ByVal Name As String, _
    ByVal Conn As OleDbConnection, _
    ByVal SQL As String)

    Dim DA As OleDbDataAdapter
    DA = New OleDbDataAdapter(SQL, Conn)
    DA.Fill(DS, Name)
End Sub
```

Listing 10.5: Hilfsroutinen zur einfacheren Aufnahme von Tabellen in ein DataSet
[ADONET_Hilfsroutinen.vb]

Beispiel 2

DataSet mit mehreren Tabellen

Das zweite Beispiel verwendet obige Hilfsroutinen, um ein DataSet mit mehr als einer Tabelle zu füllen. Dieses Beispiel geht dabei nicht effizient mit den Systemressourcen um, weil der Hilfsroutine `AddToDS()` jedes Mal die gleiche Verbindungszeichenfolge übergeben wird. Auch wenn OLEDB Connection Pooling verwendet, könnte die Performance gesteigert werden, indem zu Beginn eine Datenbankverbindung erzeugt wird, die bei jedem Vorgang wieder verwendet wird. Dies wird im nächsten Beispiel gezeigt.

```
' =====
' Beispiel aus FCL-Buch.sln
' Ausgabe mehrerer Tabellen mit Hilfe eines
' DataSet-Objekts
```

```

' (C) Holger@Schwichtenberg.de
' =====
Sub ADONET_DS2()
    Dim DS As System.Data.DataSet
    Dim Com As System.Data.OleDb.OleDbCommand
    Dim SQL As String

    ' --- DataSet erzeugen
    DS = New System.Data.DataSet()

    ' --- 1. Tabelle einlesen
    SQL = "SELECT * FROM Produkte where kategorie=23"
    AddToDS(DS, "EDV-Antik", CONNSTRING, SQL)

    ' --- 2. Tabelle einlesen
    SQL = "SELECT * FROM Bestellungen"
    AddToDS(DS, "Bestellungen", CONNSTRING, SQL)

    ' --- 3. Tabelle einlesen
    SQL = "SELECT * FROM Kunden"
    AddToDS(DS, "Kunden", CONNSTRING, SQL)

    ' --- Ausgaben der Tabellen
    out("--- Inhalt des DataSet als CSV:")
    out(table_To_csv(DS.Tables("EDV-Antik")))
    out(table_To_csv(DS.Tables("Bestellungen")))
    out(table_To_csv(DS.Tables("Kunden")))

    out("--- Inhalt des DataSet als XML:")
    out(DS.GetXml)

End Sub

```

Listing 10.6: Ausgabe mehrerer Tabellen mit Hilfe eines DataSet-Objekts[ADONET_DS.vb]

Ausgabe

Die Ausgabe des DataSet-Objekts als XML-Dokument beweist, **Ergebnis** dass alle drei Tabellen in dem einen DataSet-Objekt enthalten sind.

```

<NewDataSet>
  <EDV-Antik>
    <ID>1</ID>
    <Name>Commodore C64</Name>
    <Kategorie>23</Kategorie>
    <Preis>500</Preis>
  </EDV-Antik>
  <EDV-Antik>
    <ID>3</ID>
    <Name>Amiga 500</Name>

```

```

        <Kategorie>23</Kategorie>
        <Preis>800</Preis>
    </EDV-Antik>
    <Bestellungen>
        <KundenID>1</KundenID>
        <PID>1</PID>
        <Menge>7</Menge>
        <Datum>2001-08-01T00:00:00.0000000+02:00</Datum>
    </Bestellungen>
    <Bestellungen>
        <KundenID>1</KundenID>
        <PID>2</PID>
        <Menge>5</Menge>
        <Datum>2001-08-01T00:00:00.0000000+02:00</Datum>
    </Bestellungen>
    <Kunden>
        <KundenID>1</KundenID>
        <KundenName>Holger Schwichtenberg</KundenName>
        <KundenOrt>Essen</KundenOrt>
    </Kunden>
    <Kunden>
        <KundenID>2</KundenID>
        <KundenName>Barbara Hildebrandt</KundenName>
        <KundenOrt>Ratingen</KundenOrt>
    </Kunden>
</NewDataSet>

```

Nutzung von Datenbankverbindungen

OleDbConnection Analog zu ADO kann man auch in ADO.NET durch Verwendung der Klasse `OleDbConnection` wesentlich effizienter längerfristige Verbindungen zu einer Datenquelle unterhalten.

Beispiel 3

Tabellen einlesen mit Verbindung Das folgende Beispiel verbessert das Beispiel 2: Dieses Mal wird die Verbindung nicht für jede Tabelle einzeln auf- und wieder abgebaut, sondern es wird eine (dauerhafte) Verbindung am Anfang erzeugt.

```

' =====
' Beispiel aus FCL-Buch.sln
' Ausgabe mehrerer Tabellen mit Hilfe eines
' DataSet-Objekts und expliziter Connection
' (C) Holger@Schwichtenberg.de
' =====
Sub ADONET_DS3()
    Dim DS As System.Data.DataSet
    Dim Conn As System.Data.OleDb.OleDbConnection
    Dim Com As System.Data.OleDb.OleDbCommand
    Dim SQL As String

```



```

' --- Verbindung öffnen
Conn = New OleDbConnection(CONNSTRING)
Conn.Open()
' --- DataSet erzeugen
DS = New System.Data.DataSet()

' --- 1. Tabelle einlesen
SQL = "SELECT * FROM Produkte where kategorie=23"
AddToDS(DS, "EDV-Antik", Conn, SQL)

' --- 2. Tabelle einlesen
SQL = "SELECT * FROM Bestellungen"
AddToDS(DS, "Bestellungen", Conn, SQL)

' --- 3. Tabelle einlesen
SQL = "SELECT * FROM Kunden"
AddToDS(DS, "Kunden", Conn, SQL)

' --- Ausgaben der Tabellen
out("--- Inhalt des DataSet als CSV:")
out(table_To_csv(DS.Tables("EDV-Antik")))
out(table_To_csv(DS.Tables("Bestellungen")))
out(table_To_csv(DS.Tables("Kunden")))

out("--- Inhalt des DataSet als XML:")
out(DS.GetXml())

' --- Verbindung schließen
Conn.Close()

End Sub

```

Listing 10.7: Ausgabe mehrerer Tabellen mit Hilfe eines DataSet-Objekts und expliziter Connection [ADONET_DS.vb]

10.7 Daten ändern mit DataSets

Datensätze, die über ein DataSet-Objekt im Zugriff sind, können grundsätzlich geändert und gelöscht werden. Außerdem können Datensätze angefügt werden. Allerdings ist das Vorgehen in ADO.NET völlig anders als im klassischen ADO:

Update()

- In ADO musste nach jedem geänderten Datensatz ein `Update()` aufgerufen werden. In ADO.NET reicht ein Aufruf von `Update()` am Ende aller Arbeiten an der Tabelle.
- `Update()` wird auf dem `OleDbDataAdapter`-Objekt ausgeführt, wobei das speichernde `DataSet`-Objekt und – optional – die zu speichernde Tabelle angegeben werden müssen. Ohne Angabe

einer Tabelle werden alle Tabellen gespeichert. Alternativ kann auch ein Array mit `DataRow`-Objekten angegeben werden, um nur einzelne Datensätze zu speichern.

```
obj_DataAdapter.Update(obj_DataSet, "Tabellenname")
```

`Update()` sendet eine Reihe von SQL-DML-Befehlen (INSERT, UPDATE oder DELETE) zur Ausführung der Änderungen an die Datenquelle. Diese SQL-Befehle müssen vor dem Ausführen von `Update()` zunächst vom Entwickler definiert oder aber von einer Instanz der Klasse `OleDbCommandBuilder` zur Laufzeit erzeugt werden. Der Datenadapter selbst erzeugt die Befehle nicht automatisch.

Diese letzte Anforderung wirkt umständlich und aufwendig, bietet aber neue Möglichkeiten zur Optimierung.

Im Folgenden werden zunächst der `OleDbCommandBuilder` und daran anschließend die verschiedenen Änderungs-Operationen besprochen. Wenn Ihnen das alles zu kompliziert erscheint: Schauen Sie doch direkt ins Kapitel zur `OleDbCommand`-Klasse (siehe Kapitel 10.8).

10.7.1 Befehle generieren mit dem CommandBuilder

SQL-Befehle generieren lassen

In vielen Fällen reichen die vom `OleDbCommandBuilder` automatisch generierten SQL-DML-Befehle (UPDATE, INSERT und DELETE). Der `OleDbCommandBuilder` erwartet als Ausgangsbasis für seine Dienste ein `OleDbDataAdapter`-Objekt. Den dort abgelegten SQL-Befehl verwendet er als Grundlage für die Generierung der SQL-DML-Befehle. Die `OleDbCommandBuilder`-Klasse bietet mit `GetInsertCommand()`, `GetUpdateCommand()` und `GetDeleteCommand()` Zugriff auf die generierten SQL-Befehle. Alle drei Methoden liefern ein `OleDbCommand`-Objekt zurück, das in dem Attribut `CommandText` den jeweiligen SQL-Befehl enthält.



Es ist nicht möglich, die generierten SQL-Befehle direkt über den Datenadapter zu ermitteln. Ein `OleDbDataAdapter`-Objekt besitzt zwar Attribute mit Namen `InsertCommand`, `UpdateCommand` und `DeleteCommand`. Diese Attribute sind aber nur für die manuelle Zuweisung von Befehlen zu gebrauchen. Beim Einsatz eines `CommandBuilders` enthalten sie alle `Nothing`.

Beispiel

Das folgende Beispiel gibt die von einem `OleDbCommandBuilder`-Objekt für die Tabelle »Bestellungen« generierten SQL-DML-Befehle aus.

**SQL-Befehle
generieren**

```
' =====
' Beispiel aus FCL-Buch.sln
' Verwendung des CommandBuilders zur Generierung
' von SQL-DML-Befehlen
' (C) Holger@Schwichtenberg.de
' =====

Sub ADONET_CommandBuilder()

    Const FAKTOR As Single = 1.1
    Const SQL As String = _
        "SELECT * FROM Bestellungen"

    ' --- Nicht erlaubter SQL-String
    '     Const SQL As String = _
    '         "SELECT * FROM Produkte,
    '         Bestellungen where Produkte.ID = Bestellungen.PID"

    Dim DA As System.Data.OleDb.OleDbDataAdapter
    Dim CB As System.Data.OleDb.OleDbCommandBuilder

    outtitle("Test des OLEDBCommandBuilder")

    ' --- Tabelle einlesen
    DA = New OleDbDataAdapter(SQL, CONNSTRING)
    CB = New OleDbCommandBuilder(DA)

    out("SELECT-Befehl: " & SQL)
    out("INSERT-Befehl: " & _
        CB.GetInsertCommand.CommandText())
    out("DELETE-Befehl: " & _
        CB.GetDeleteCommand.CommandText())
    out("UPDATE-Befehl: " & _
        CB.GetUpdateCommand.CommandText())

End Sub
```

Listing 10.8: Verwendung des CommandBuilders zur Generierung von SQL-DML-Befehlen [ADONET_DS.vb]

Ausgabe

Der `OleDbCommandBuilder` liefert in obiger Routine die nachfolgend abgedruckten SQL-Befehle. Anstelle der konkreten Werte stehen Fragezeichen als Platzhalter. Diese Platzhalter werden bei Änderungen am `DataSet` von dem `OleDbDataAdapter` gefüllt.

Ergebnis

Test des OleDbCommandBuilder

SELECT-Befehl: SELECT * FROM Bestellungen

INSERT-Befehl: INSERT INTO Bestellungen(KundenID , PID ,
Menge , Datum) VALUES (? , ? , ? , ?)

DELETE-Befehl: DELETE FROM Bestellungen WHERE ((KundenID =
?) AND (PID = ?) AND ((? IS NULL AND Menge IS NULL) OR (Menge
= ?)) AND ((? IS NULL AND Datum IS NULL) OR (Datum = ?)))

UPDATE-Befehl: UPDATE Bestellungen SET KundenID = ? , PID = ?
, Menge = ? , Datum = ? WHERE ((KundenID = ?) AND (PID = ?)
AND ((? IS NULL AND Menge IS NULL) OR (Menge = ?)) AND ((? IS
NULL AND Datum IS NULL) OR (Datum = ?)))

Beschränkungen

Sehr ärgerlich ist, dass der OleDbCommandBuilder zwei Beschränkungen unterliegt:

- Er kann nur SQL-DML-Befehle für einzelne Tabellen erzeugen. Er verweigert die Unterstützung, wenn mehrere Tabellen an dem SQL-Befehl beteiligt sind.
- Der CommandBuilder verweigert auch seine Mithilfe, wenn in den Feldnamen Leerzeichen oder andere nicht-alphanumerische Zeichen vorkommen.

Wenn dem CommandBuilder ein DataAdapter mit einem SQL-Befehl wie »SELECT * FROM Produkte, Bestellungen where Produkte.ID = Bestellungen.PID« überreicht wird, tritt er in den Streik: »Dynamic SQL generation is not supported against multiple base tables.«

10.7.2 Datensätze ändern

Update() Sofern das UPDATE-Kommando über den betreffenden Datenadapter definiert wurde, kann ein DataSet ohne weiteren Befehlsaufruf beschrieben werden, indem man zu den gewünschten Zeilen (DataRow-Objekten) navigiert und den gewünschten Spalten über DataRow.Item("SpaltenName") einen neuen Wert zuweist. Nach Ausführung aller Änderungen müssen die Änderungen mit DA.Update(DS, "TabellenName") zur Datenquelle zurückübermittelt werden. Der Aufruf von Update() löst den Aufruf des im Datenadapter definierten UPDATE-Kommandos aus.

Beispiel

Das folgende Beispiel zeigt das Ändern von Datensätzen. In der Tabelle »Produkte« werden alle Preise zu Produkten der Produktkategorie 23 (»EDV-Antik«) um 10% erhöht. Das UPDATE-Kommando wird über den `OleDbCommandBuilder` erzeugt.

Ändern der Produkttabelle

```
' =====
' Beispiel aus FCL-Buch.sln
' Ändern von Datensätzen In einer Tabelle
' (C) Holger@Schwichtenberg.de
' =====
Sub ADONET_DS_aendern()

    Const FAKTOR As Single = 1.1
    Const SQL As String = _
        "SELECT * FROM Produkte where Kategorie = 23"

    Dim DS As System.Data.DataSet
    Dim DT As System.Data.DataTable
    Dim DR As System.Data.DataRow
    Dim DA As System.Data.OleDb.OleDbDataAdapter
    Dim CB As System.Data.OleDb.OleDbCommandBuilder

    outtitle("Datensatz mit DataSet anfügen")

    ' --- DataSet erzeugen
    DS = New System.Data.DataSet()
    ' --- Adapter definieren
    DA = New OleDbDataAdapter(SQL, CONNSTRING)
    ' --- SQL-DML-Befehle erzeugen
    CB = New OleDbCommandBuilder(DA)
    ' --- Tabelle einlesen
    DA.Fill(DS, "EDV-Antik")

    ' --- Zugriff auf Tabelle
    DT = DS.Tables("EDV-Antik")

    ' --- Ausgabe der Tabelle
    out("--- Bisheriger Zustand der Produkttabelle:")
    out(table_To_csv(DS.Tables("EDV-Antik")))

    ' --- AUSFÜHRUNG EINER PREISERHÖHUNG
    For Each DR In DT.Rows
        Dim AlterPreis As Double
        Dim NeuerPreis As Double

        AlterPreis = CDb1(DR("Preis"))
        NeuerPreis = Math.Round(Alterpreis * FAKTOR, 2)

        DR!Preis = NeuerPreis
```

```

        out(DR("Name").ToString & " wurde von " _
            & AlterPreis & " _ auf " _
            & NeuerPreis & " _ erhöht!")
    Next

    ' --- Änderungen an Datenquelle senden
    DA.Update(DS, "EDV-Antik")

    ' --- Ausgabe der Tabelle
    out("--- Neuer Zustand der Produkttabelle:")
    out(table_To_csv(DS.Tables("EDV-Antik"))))

End Sub

```

Listing 10.9: Ändern von Datensätzen in einer Tabelle [ADONET_DS.vb]

10.7.3 Datensätze anfügen

DataRow() Beim Anfügen eines Datensatzes wird statt der `AddNew()`-Methode (wie beim klassischen ADO) nun eine Instanz der Klasse `System.Data.DataRow` benötigt. Wichtig ist, dass die Klasse `System.Data.DataRow` nicht instanziiert werden kann, sondern neue Instanzen über die Methode `DataRow()` der `DataTable`-Klasse erzeugt werden müssen.

```

DT = DS.Tables("Tabellenname")
DR = DT.NewRow()

```

Update() Das neu angelegte `DataRow`-Objekt kann wie jedes andere `DataRow`-Objekt beim Ändern bestehender Daten beschrieben werden.

```

DR("KundenID") = 2
DR("PID") = 6
DR("Menge") = 1
DR("Datum") = DateTime.Today

```

Am Ende ist ebenfalls der Aufruf von `Update()` notwendig.

```

DA.Update(DS, "Tabellenname")

```

Beispiel

Neue Bestellung In dem folgenden Beispiel wird eine neue Bestellung erzeugt, indem an die Tabelle »Bestellungen« ein Datensatz angehängt wird.

```

' =====
' Beispiel aus FCL-Buch.sln
' Hinzufügen eines neuen Datensatzes
' (C) Holger@Schwichtenberg.de
' =====

```

```
Sub ADONET_DS_neuerDatensatz()

    Const FAKTOR As Single = 1.1

    Const SQL As String = _
        "SELECT * FROM Bestellungen where KundenID = 0"

    Dim DS As System.Data.DataSet
    Dim DT As System.Data.DataTable
    Dim DR As System.Data.DataRow
    Dim DA As System.Data.OleDb.OleDbDataAdapter
    Dim CB As System.Data.OleDb.OleDbCommandBuilder

    outtitle("Datensatz mit DataSet anfügen")

    ' --- DataSet erzeugen
    DS = New System.Data.DataSet()
    ' --- Adapter definieren
    DA = New OleDbDataAdapter(SQL, CONNSTRING)
    ' --- SQL-DML-Befehle erzeugen
    CB = New OleDbCommandBuilder(DA)
    ' --- Tabelle einlesen
    DA.Fill(DS, "DT_Bestellungen")

    ' --- Zugriff auf Tabelle
    DT = DS.Tables("DT_Bestellungen")

    ' --- Ausgabe der Tabelle
    out("--- Bisheriger Inhalt:")
    out(table_To_csv(DS.Tables("DT_Bestellungen")))

    ' --- ANFÜGEN EINES DATENSATZES
    DR = DT.NewRow()
    ' Füllen der Zeile
    DR("KundenID") = 2
    DR("PID") = 6
    DR("Menge") = 1
    DR("Datum") = DateTime.Today
    ' Anfügen der Zeile an Tabelle
    DT.Rows.Add(DR)

    ' --- Speichern der Änderungen
    DA.Update(DS, "DT_Bestellungen")

    ' --- Ausgabe der Tabelle
    out("--- Neuer Inhalt:")
    out(table_To_csv(DS.Tables("DT_Bestellungen")))

End Sub
```

Listing 10.10: Hinzufügen eines Datensatzes [ADONET_DS.vb]

Ausgabe

```

1  Datensatz mit DataSet anfügen
2  --- Bisheriger Inhalt:
3  KundenID;PID;Datum;Menge
4  --- Neuer Inhalt:
5  KundenID;PID;Datum;Menge
6  2;6;21.12.2001 00:00:00;1
7  -----ENDE

```

10.7.4 Datensätze löschen

Delete() Zum Löschen von Datensätzen wird die `Delete()`-Methode auf einem `DataRow`-Objekt benutzt. Die Änderungen müssen ebenfalls mit `Update()` bestätigt werden.

Beispiel

Bestellungen löschen

Die folgende Routine löscht alle Datensätze aus der Tabelle »Bestellungen«, deren Datum dem aktuellen Datum entspricht. Die Herausforderung liegt darin, dass in dem SQL-Befehl das Datum im amerikanischen Datumsformat angegeben werden muss. Eine elegante Möglichkeit dazu ist die Verwendung der FCL-Klasse `System.Globalization.CultureInfo`. Eine Instanz dieser Klasse kann als Parameter bei der Methode `toString()` für ein `DateTime`-Objekt verwendet werden. Es wird also zunächst mit der statischen Methode `DateTime.Today()` das aktuelle Datum ermittelt. Anschließend wird daraus mit `toString()` unter Angabe eines `CultureInfo`-Objekts (»en-us«) eine Zeichenkette erzeugt, die das aktuelle Datum in amerikanischer Schreibweise enthält (siehe auch Kapitel 8).

Nach dem Einlesen der Tabelle werden in einer Schleife alle Datensätze mit `Delete()` gelöscht. Die Änderungen werden mit `Update()` an die Datenquelle zurückgesendet. Die Ausgabe der Tabelle am Ende liefert keine Datensätze mehr.

```

' =====
' Beispiel aus FCL-Buch.sln
' Löschen von Datensätzen In einem DataSet
' (C) Holger@Schwichtenberg.de
' =====
Sub ADONET_DS_loeschen()
    Dim DS As System.Data.DataSet
    Dim Com As System.Data.OleDb.OleDbCommand
    Dim SQL As String
    Dim DT As System.Data.DataTable

```



```

Dim DR As System.Data.DataRow
Dim DA As System.Data.OleDb.OleDbDataAdapter
Dim CB As System.Data.OleDb.OleDbCommandBuilder

outtitle("Datensätze löschen")

' --- Zusammensetzung des SQL-Befehls
Dim heute As DateTime = DateTime.Today() ' aktuelles
Datum
Dim heuteus As String = _
    heute.ToString( _
        New System.Globalization.CultureInfo("en-us")) '
umformatieren
SQL = "SELECT * FROM Bestellungen WHERE Datum = #" &
    heuteus & "#"

' --- DataSet erzeugen
DS = New System.Data.DataSet()
' --- Adapter definieren
DA = New OleDbDataAdapter(SQL, CONNSTRING)
' --- SQL-DML-Befehle erzeugen
CB = New OleDbCommandBuilder(DA)
' --- Tabelle einlesen
DA.Fill(DS, "heutige_Bestellungen")

' --- Zugriff auf Tabelle
DT = DS.Tables("heutige_Bestellungen")

' --- Ausgabe der Tabelle
out("--- Bisheriger Inhalt der Tabelle:")
out(table_To_csv(DS.Tables("heutige_Bestellungen")))

' --- Löschen aller Datensätze
For Each DR In DT.Rows
    out("Bestellung von " & _
        DR!KundenID.ToString() & _
        " von " & DR!PID.ToString() & _
        " wird gelöscht!")
    DR.Delete()
Next

' --- Änderungen zurückliefern
DA.Update(DS, "heutige_Bestellungen")

' --- Ausgabe der Tabelle
out("--- Neuer Inhalt der Tabelle:")
out(table_To_csv(DS.Tables("heutige_Bestellungen")))

End Sub

```

Listing 10.11: Löschen von Datensätzen [ADONET_DS.vb]

Ausgabe

```
1  Datensätze löschen
2  --- Bisheriger Inhalt der Tabelle:
3  KundenID;PID;Datum;Menge
4  2;3;21.12.2001 00:00:00;1
5  2;1;21.12.2001 00:00:00;1
6  2;5;21.12.2001 00:00:00;1
7  2;6;21.12.2001 00:00:00;1
8  Bestellung von 2 von 3 wird gelöscht!
9  Bestellung von 2 von 1 wird gelöscht!
10 Bestellung von 2 von 5 wird gelöscht!
11 Bestellung von 2 von 6 wird gelöscht!
```

10.8 Daten ändern mit OleDbCommand

Kommandos ausführen

Das Ändern von Daten über ein DataSet-Objekt ist dann nicht der geeignete Weg, wenn die Datensätze gar nicht gelesen werden müssen, sondern direkt eine Veränderung auf der Datenquelle ausgeführt werden soll. Sofern die Datenquelle die Ausführung von SQL-DML-Befehlen unterstützt, kann man diese Befehle direkt mit Hilfe der Klasse OleDbCommand (bzw. SqlCommand beim SQL-Server) übermitteln.

Ein OleDbCommand erwartet bei der Instanziierung als Parameter einen Befehlstext und ein geöffnetes OleDbConnection-Objekt:

```
Dim Cmd As New OleDbCommand("Befehl", objConn)
```

Danach können Parameter wie die Timeout-Zeit gesetzt werden. Die Ausführung des Befehls beginnt mit einem der drei in der folgenden Tabelle genannten Methodenaufrufe:

Methode	Erläuterung
ExecuteNonQuery()	Ausführung eines SQL-Befehls oder einer Stored Procedure ohne Rückgabemenge (z.B. INSERT, DELETE, UPDATE)
ExecuteReader()	Ausführung eines SQL-Befehls (SELECT) oder einer Stored Procedure mit Rückgabemenge. Diese Möglichkeit wurde schon in Kapitel 10.5 besprochen.
ExecuteScalar()	Ausführung eines SQL-Befehls oder einer Stored Procedure, die einen einzelnen Wert zurückliefert

Tabelle 10.3: Verschiedene Methoden zum Start des Befehls

Beispiel

Im folgenden Beispiel wird über die bereits verwendete Tabelle »Produkt« aus der Datenbank *EDVShop.mdb* eine SQL-UPDATE-Anweisung ausgeführt, die alle Preise um 3,0 % erhöht.

Preiserhöhung

Für die Erfolgskontrolle (Vorher-Nachher-Vergleich) wird auf die in Kapitel 10.5 vorgestellte Routine `ADONET_DR1()` zurückgegriffen, die den Inhalt der Tabelle »Produkt« ausgibt.

```
' =====
' Beispiel aus FCL-Buch.sln
' Daten ändern mit der Ausführung eines OleDbCommand
' (C) Holger@Schwichtenberg.de
' =====

Sub ADONET_DoCmd()

    ' --- Produktliste vorher ausgeben
    ADONET_DR1()

    Const Befehl = "UPDATE Produkte SET Produkte.Preis =
[Preis]*1.03;"
    ' --- Vorbereiten
    Dim Conn As New OleDbConnection(CONNSTRING)
    Conn.Open()
    Dim Cmd As New OleDbCommand(Befehl, Conn)
    Cmd.CommandTimeout = 2 ' maximal 2 Sekunden warten
    ' --- Ausführen
    Cmd.ExecuteNonQuery()

    ' --- Schließen
    Conn.Close()

    ' --- Produktliste nachher ausgeben
    ADONET_DR1()
End Sub
```

Listing 10.12: Daten ändern mit der Ausführung eines OleDbCommand
[ADONET_Command.vb]

Ausgabe

Die Ausgabe des obigen Programms beweist, dass die Erhöhung des Feldes »Preis« ausgeführt wurde.

```
1 Commodore C64 (Produkt #1) kostet 500 _
2 Amiga 500 (Produkt #3) kostet 600 _
3 IBM 286er / 2 MB (Produkt #4) kostet 300 _
4 IBM 286er / 4 MB (Produkt #5) kostet 350 _
5 Amiga 1000 (Produkt #6) kostet 800 _
```

```

1 Commodore C64 (Produkt #1) kostet 515 _
2 Amiga 500 (Produkt #3) kostet 618 _
3 IBM 286er / 2 MB (Produkt #4) kostet 309 _
4 IBM 286er / 4 MB (Produkt #5) kostet 360,5 _
5 Amiga 1000 (Produkt #6) kostet 824 _

```

10.9 Verknüpfung von Tabellen in einem DataSet

Joins im DataSet

Tabellen, die innerhalb eines `DataSet`-Objekts leben, können verbunden (»gejoint«) werden. ADO.NET erlaubt es, über Schlüssel-felder Eltern-Kind-Beziehungen zu definieren, so dass ein hierarchisches Datenmodell in dem `DataSet` entsteht. Diese Technik ist neu in ADO.NET; im klassischen ADO konnten – wenn überhaupt – Verknüpfungen nur auf Seiten der Datenquelle durchgeführt werden.



Datenbankmanagementsysteme sind dahingehend optimiert, Joins zwischen ihren Tabellen schnell auszuführen. Keineswegs sollte man aufwendige Joins vom Datenbankserver auf den Client verlagern. Wenn es aber darum geht, heterogene Datenquellen zu verknüpfen, kann eine Verknüpfung auf dem Client eine gute Alternative sein.

Klasse `DataRelation`

DataRelation

ADO.NET definiert eine Klasse zum Aufbau von Verknüpfungen innerhalb eines `DataSet`-Objekts: `System.Data.DataRelation`. Der Konstruktor erwartet drei Parameter:

- ▶ Name für die Verknüpfung in Form einer Zeichenkette
- ▶ ein `DataColumn`-Objekt für die Verknüpfungsspalte in der übergeordneten Tabelle (Mastertabelle, Elterntabelle), also in der Regel der Primärschlüssel der Tabelle
- ▶ ein `DataColumn`-Objekt für die Verknüpfungsspalte in der untergeordneten Tabelle (Detailtabelle, Kindtabelle), also ein Fremdschlüssel

Syntax

```
DRel = New System.Data.DataRelation("Name", DC_MASTER,
DC_DETAIL)
```

An die Stelle von `DC_MASTER` und `DC_DETAIL` kann auch jeweils ein `Array of DataColumn` gesetzt werden, wenn die Verknüpfung über mehrere Spalten definiert werden soll.

Ein `DataRelation`-Objekt muss der `Relations`-Collection des `DataSet`-Objekts hinzugefügt werden.

```
DS.Relations.Add(DRel)
```

Relations

Ausgabe einer `DataRelation`

Eine Verknüpfung wird nicht wie ein Join in SQL als eine eigene Tabelle angesehen. Daher wird eine Relation auch nicht in die `Tables`-Collection eines `DataSet`-Objekts aufgenommen. ADO.NET baut aus den Verknüpfungen hierarchische Datensätze. Auf der Ebene jedes einzelnen `DataRow`-Objekts können mit Hilfe der Methoden `GetChildRows()` und `GetParentRows()` die unter- bzw. übergeordneten Datensätze ermittelt werden. Beide Methoden erwarten als Parameter ein `DataRelation`-Objekt und liefern als Rückgabewert ein `Array of DataRow`.

`GetChildRows()` liefert für den aktuellen Datensatz aus der Mastertabelle die zugehörigen Detaildatensätze. `GetParentRows()` liefert für den aktuellen Datensatz aus der Detailtabelle die zugehörigen Masterdatensätze.

GetChildRows()



Beispiel

Die folgende Routine verwendet die Tabellen »Produkte«, »Bestellungen« und »Kunden« aus der Datenbank *EDVShop.mdb*. Es werden zwei Beziehungen definiert:

- ▶ zwischen »Bestellungen« und »Kunden« über die `KundenID`
- ▶ zwischen »Bestellungen« und »Produkte« über die `ProduktID`

Diese Beziehungen werden anschließend dazu genutzt, alle Bestellungen gruppiert nach Kunde auszugeben. Dabei werden statt der `KundenID` und der `ProduktID` der `Kundenname` und der `Produktname` genannt.

Die oberste Schleife läuft über die Tabelle »Kunden«. Für jeden einzelnen Kunden werden mit `GetChildRows()` die zugehörigen Bestelldatensätze aus der Tabelle »Bestellungen« ermittelt. Für jede Bestellung wird über die zweite Verknüpfung zwischen »Bestellungen« und »Produkte« der `Produktname` für die in der Tabelle »Bestellungen« gegebene `Produktnummer` (»PID«) ermittelt. Von dem Bestelldatensatz aus erhält man die zugehörigen Produktdatensätze durch `GetParentRows()`. Da durch das Datenmodell klar ist, dass es zu einer »PID« nur einen Datensatz gibt, ist

Verknüpfung der Tabellen »Produkte«, »Bestellungen« und »Kunden«

hier keine Schleife notwendig. Mit `DR2.GetParentRows(DRe12)(0)` ermittelt man den ersten (und einzigen) übergeordneten Produktdatensatz.

```
' =====
' Beispiel aus FCL-Buch.sln
' Verknüpfung von Tabellen innerhalb eines DataSet-Objekts
' (C) Holger@Schwichtenberg.de
' =====
Sub ADONET_join1()
    Dim DS As System.Data.DataSet
    Dim Conn As System.Data.OleDb.OleDbConnection
    Dim Com As System.Data.OleDb.OleDbCommand
    Dim DT As System.Data.DataTable
    Dim DR1 As System.Data.DataRow
    Dim DR2 As System.Data.DataRow
    Dim DR3 As System.Data.DataRow
    Dim DRe11 As System.Data.DataRelation
    Dim DRe12 As System.Data.DataRelation
    Dim DC1 As System.Data.DataColumn
    Dim DC2 As System.Data.DataColumn
    Dim SQL As String

    ' --- Verbindung öffnen
    Conn = New
System.Data.OleDb.OleDbConnection(CONNSTRING)
    Conn.Open()
    ' --- DataSet erzeugen
    DS = New System.Data.DataSet()

    ' --- 1. Tabelle einlesen
    SQL = "SELECT * FROM Produkte"
    AddToDS(DS, "Produkte", Conn, SQL)

    ' --- 2. Tabelle einlesen
    SQL = "SELECT * FROM Bestellungen"
    AddToDS(DS, "Bestellungen", Conn, SQL)

    ' --- 3. Tabelle einlesen
    SQL = "SELECT * FROM Kunden"
    AddToDS(DS, "Kunden", Conn, SQL)

    ' --- 1. Verknüpfung erstellen
    DC2 = DS.Tables("Bestellungen").Columns("KundenID")
    DC1 = DS.Tables("Kunden").Columns("KundenID")
    DRe11 = New System.Data.DataRelation("Kunden mit
Bestellungen", DC1, DC2)
    DS.Relations.Add(DRe11)

    ' --- 2. Verknüpfung erstellen
    DC2 = DS.Tables("Bestellungen").Columns("PID")
    DC1 = DS.Tables("Produkte").Columns("ID")
```

```

    DRel2 = New System.Data.DataRelation("Produkte mit
Bestellungen", DC1, DC2)
    DS.Relations.Add(DRel2)

    DT = DS.Tables("Kunden")
    ' --- Schleife über alle Kunden
    For Each DR1 In DT.Rows
        out(DR1!KundenName)
        ' --- Schleife über alle zugeordneten Bestellungen
        For Each DR2 In DR1.GetChildRows(DRel1)
            Dim d As DateTime
            d = CType(DR2!Datum, DateTime)

            ' Ermitteln des Produktdatensatzes
            DR3 = DR2.GetParentRows(DRel2)(0)

            ' Ausgabe
            out("Bestellung vom " & _
                d.ToShortDateString() & ":" & _
                DR2!menge.ToString() & "x " & _
                DR3!Name.ToString())
        Next
    Next

    ' --- Verbindung schließen
    Conn.Close()

End Sub

```

Listing 10.13: Verknüpfung von Tabellen innerhalb eines DataSet-Objekts
[ADONET_DS.vb]

Ausgabe

```

1  Holger Schwichtenberg
2  Bestellung vom 01.08.2001:1x Commodore C64
3  Bestellung vom 01.08.2001:3x Mauspad
4  Bestellung vom 01.08.2001:3x Amiga 500
5  Barbara Hildebrandt
6  Bestellung vom 03.01.2002:1x IBM 286er / 2 MB
7  Bestellung vom 03.01.2002:3x IBM 286er / 4 MB
8  Bestellung vom 03.01.2002:1x Amiga 1000

```

10.10 Dynamische Tabellen im Speicher

**Dynamische
Tabellen ohne
persistente
Datenquelle**

Analog zu ADO erlaubt auch ADO.NET dynamische Tabellen im Hauptspeicher, die nicht auf einer persistenten Datenquelle basieren. Sie können eine Instanz von `System.Data.DataTable` erzeugen und manuell Spalten anlegen und mit Daten füllen. Die Vorgehensweise ist intuitiv und der im klassischen ADO sehr ähnlich. Als Datentypen für die Spalten müssen CLR-Typen angegeben werden.



Dynamische Tabellen eröffnen die Möglichkeit, große Datenmengen auf einfache Weise im Hauptspeicher zu halten, zu durchsuchen und miteinander zu verknüpfen. Sie sind eine Alternative zur Verwendung von Collections (siehe Kapitel 5).

Anhängen dynamischer Tabellen an DataSet-Objekte

**Anhängen an
DataSets**

Dynamische Tabellen können auch an ein `DataSet` angehängt werden. Dies zeigt das folgende Beispiel. Der Vorteil dabei ist, dass die Tabelle mit anderen Tabellen verknüpft werden kann und außerdem die Option zur Ausgabe in XML bzw. zur Manipulation durch XML besteht.

Beispiel

**Erzeugen einer
Tabelle ohne
Datenquelle**

In diesem Beispiel wird eine aus drei Feldern bestehende Tabelle im Hauptspeicher erzeugt und probeweise mit einigen Werten gefüllt, bevor die Routine `table_to_csv()` zur Ausgabe aufgerufen wird. Danach wird die Tabelle an ein `DataSet`-Objekt angehängt und das `DataSet` als XML ausgegeben.

```
' =====
' Beispiel aus FCL-Buch.sln
' Aufbau einer dynamischen Tabelle im Speicher
' (C) Holger@Schwichtenberg.de
' =====
Sub ADONET_dynTabelle1()
    Dim DR As DataRow
    Dim DT As DataTable
    Dim DS As DataSet

    outtitle("Erzeugung einer dynamischen Tabelle " _
            "im Speicher")

    ' --- Erzeugen einer dynamischen Tabelle
    DT = New DataTable()
    ' Anfügen der Spalte an Tabelle
    With DT.Columns
        .Add("ComputerID", _
```



```

        System.Type.GetType("System.Int32"))
    .Add("Computername", _
        System.Type.GetType("System.String"))
    .Add("Aktiv", System.Type.GetType("System.Boolean"))
End With

out("Füllen der Tabelle...")
' --- Anfügen von Daten

' Erzeugen der 1. Zeile
DR = DT.NewRow()
' Füllen der Zeile
DR("ComputerID") = "1"
DR("Computername") = "Sonne2000"
DR("Aktiv") = True
' Anfügen der Zeile an Tabelle
DT.Rows.Add(DR)

' Erzeugen der 2. Zeile
DR = DT.NewRow()
' Füllen der Zeile
DR("ComputerID") = "2"
DR("Computername") = "Mars"
DR("Aktiv") = True
' Anfügen der Zeile an Tabelle
DT.Rows.Add(DR)

' Erzeugen der 3. Zeile
DR = DT.NewRow()
' Füllen der Zeile
DR("ComputerID") = "3"
DR("Computername") = "Saturn"
DR("Aktiv") = False
' Anfügen der Zeile an Tabelle
DT.Rows.Add(DR)

' --- Ausgabe
out("Ausgabe der Tabelle als CSV...")
out(table_To_csv(DT))

' -- Anfügen an ein DataSet
out("Erzeugen eines DataSet...")
' Erzeugen eines DataSet-Objekts
DS = New DataSet()
' Anfügen der Spalte an Tabelle
DS.Tables.Add(DT)

' --- Ausgabe als XML
out("Ausgabe des DataSet als XML...")
out(DS.GetXml)
End Sub

```

Listing 10.14: Aufbau einer dynamischen Tabelle im Speicher [ADONET_DS.vb]

Ausgabe

```

1  Erzeugung einer dynamischen Tabelle im Speicher
2  Füllen der Tabelle...
3  Ausgabe der Tabelle als CSV...
4  ComputerID;Computername;Aktiv
1;Sonne2000;True
2;Mars;True
3;Saturn;False
5  Erzeugen eines DataSet...
6  Ausgabe der DataSet als XML...
7  <NewDataSet>
    <Table1>
        <ComputerID>1</ComputerID>
        <Computername>Sonne2000</Computername>
        <Aktiv>true</Aktiv>
    </Table1>
    <Table1>
        <ComputerID>2</ComputerID>
        <Computername>Mars</Computername>
        <Aktiv>true</Aktiv>
    </Table1>
    <Table1>
        <ComputerID>3</ComputerID>
        <Computername>Saturn</Computername>
        <Aktiv>false</Aktiv>
    </Table1>
</NewDataSet>

```

10.11 ADO.NET und XML

Die Unterstützung der Extensible Markup Language (XML) ist ein wichtiges Konzept von ADO.NET. Die `DataSet`-Klasse tauscht Daten in Form von XML-Dokumenten mit ihrer Umwelt aus. Daten in einem `DataSet` können mit den Möglichkeiten von XML bearbeitet werden.



Obwohl oft behauptet ist es nicht richtig, dass ADO.NET die Daten in einem `DataSet` auch intern in XML-Datenstrukturen speichert.

10.11.1 Ausgabe eines DataSets als XML

XML- Unterstützung

Die `DataSet`-Klasse besitzt zwei Attribute und zwei Methoden zum Austausch mit XML:

- `GetXml()` liefert einen String mit dem Inhalt des `DataSet`-Objekts in Form eines XML-Dokuments

- ▶ `GetXmlSchema()` liefert einen String mit der Struktur der Daten im `DataSet` in Form eines XSD-Schemas
- ▶ `WriteXml()` schreibt die XML-Daten und – optional – das zugehörige XSD-Schema in eine Datei (spezifiziert durch einen URL), ein `Stream`-Objekt, ein `TextWriter`-Objekt oder ein `XmlWriter`-Objekt. Mit dem optionalen zweiten Parameter `Xml WriteMode.DiffGram` wird ein XML-Dokument im *DiffGram*-Format erzeugt. Ein *DiffGram* dokumentiert nicht nur den aktuellen Zustand eines `DataSet`, sondern auch alle ausgeführten Änderungen.
- ▶ `ReadXml()` liest XML-Daten in ein `DataSet`-Objekt ein. Mögliche Eingabequellen sind eine Datei (spezifiziert durch einen URL), ein `Stream`-Objekt, ein `TextWriter`-Objekt oder ein `XmlWriter`-Objekt. Die XML-Daten müssen ein XSD-Schema enthalten.

Persistenz von Daten

Classic-ADO unterstützt zwei Formen der Persistierung von Daten: Speicherung in einer Datei oder Speicherung in einem `Stream` (ab ADO 2.5). Bei der Abspeicherung in einer Datei stehen folgende Formate zur Verfügung: Advanced Data Tablegram (ADTG) und Extensible Markup Language (XML).

XML in ADO

ADO.NET unterstützt nur noch XML. Eine `Stream`-Klasse gibt es im Namespace `System.Data` nicht. Die XML-Daten können auf jede beliebige Art weiterverarbeitet werden, einschließlich der Möglichkeit, sie mit einem `FileStream`-Objekt im Dateisystem persistent zu machen.

XML in ADO.NET

Beispiel

Die folgende Routine gibt zunächst das XML-Schema eines `DataSet`-Objekts aus und danach den Inhalt als XML-Dokument.

DataSet als XML ausgeben

```
' =====
' Beispiel aus FCL-Buch.sln
' Ausgabe eines DataSet-Objekts als XML
' (C) Holger@Schwichtenberg.de
' =====
Sub ADONET_DSXML1()
    Dim DS As System.Data.DataSet
    Dim Com As System.Data.OleDb.OleDbCommand
    Dim SQL As String

    outtitle("Ausgabe eines DataSet als XML")

    ' --- DataSet erzeugen
    DS = New System.Data.DataSet()
```

```

' --- Tabelle einlesen
SQL = "SELECT * FROM Produkte where name Like
'Amiga%' "
Dim DA As System.Data.OleDb.OleDbDataAdapter
DA = New System.Data.OleDb.OleDbDataAdapter(SQL,
CONNSTRING)
DA.Fill(DS, "EDV-Antik")

' --- XML ausgeben
out("--- Schema:")
out(DS.GetXmlSchema)
out("--- Daten:")
out(DS.GetXml)

End Sub

```

Listing 10.15: Ausgabe eines DataSet-Objekts als XML [ADONET_XML.vb]

Ausgabe

```

1  --- Schema:
2  <?xml version="1.0" encoding="utf-16"?>
<xs:schema id="NewDataSet" xmlns="" xmlns:xs="http://
www.w3.org/2001/XMLSchema" xmlns:msdata="urn:schemas-
microsoft-com:xml-msdata">
  <xs:element name="NewDataSet" msdata:IsDataSet="true"
msdata:Locale="de-DE">
    <xs:complexType>
      <xs:choice maxOccurs="unbounded">
        <xs:element name="EDV-Antik">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="ID" type="xs:int"
minOccurs="0" />
              <xs:element name="Name" type="xs:string"
minOccurs="0" />
              <xs:element name="Kategorie" type="xs:int"
minOccurs="0" />
              <xs:element name="Preis" type="xs:decimal"
minOccurs="0" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:choice>
    </xs:complexType>
  </xs:element>
</xs:schema>
3  --- Daten:
4  <NewDataSet>
  <EDV-Antik>
    <ID>3</ID>
    <Name>Amiga 500</Name>
  </EDV-Antik>
</NewDataSet>

```

```

    <Kategorie>23</Kategorie>
    <Preis>15972</Preis>
  </EDV-Antik>
  <EDV-Antik>
    <ID>6</ID>
    <Name>Amiga 1000</Name>
    <Kategorie>23</Kategorie>
    <Preis>2662</Preis>
  </EDV-Antik>
</NewDataSet>

```

10.11.2 XmlDocument

Ein `XmlDataDocument`-Objekt ermöglicht die Bearbeitung des Inhalts eines `DataSet`-Objekts über das XML Document Object Model (DOM). Die Klasse `System.Xml.XmlDataDocument` ist abgeleitet von der Klasse `System.Xml.XmlDocument` (vgl. Kapitel 9).

DataSet via XML-DOM bearbeiten

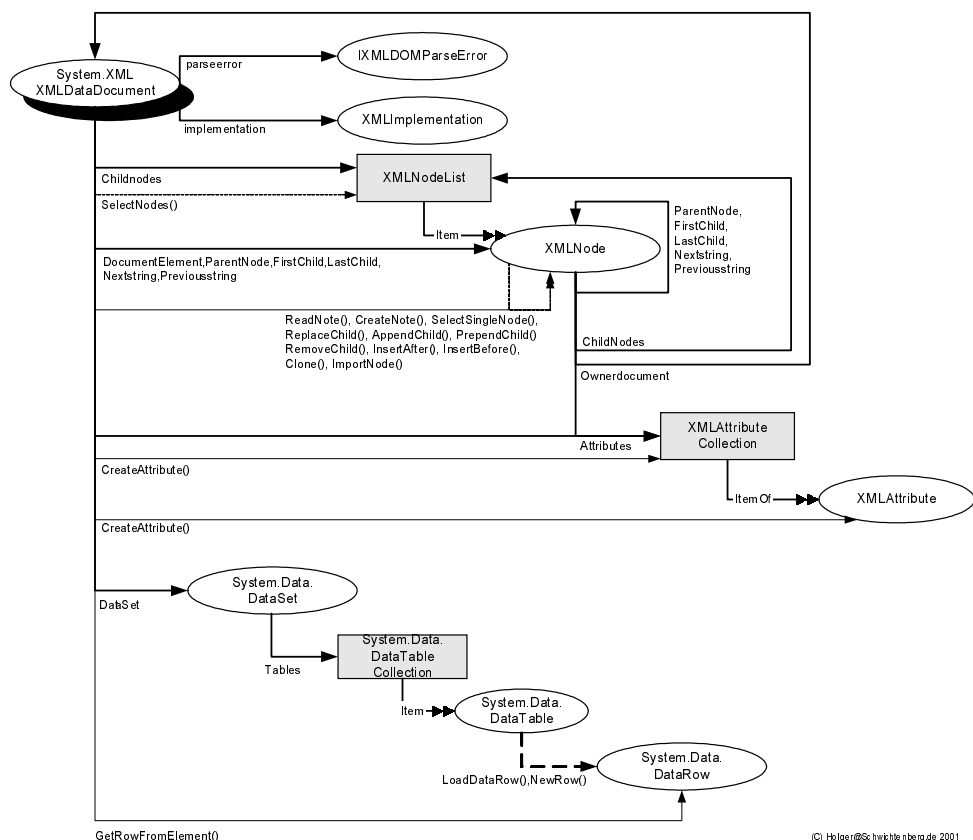


Abbildung 10.8: Objekthierarchie für `System.Xml.XmlDataDocument`

XmlDataDocument vs. DataSet

Ein `XmlDataDocument`-Objekt ist keine Kopie der XML-Daten, sondern eine andere Sicht auf ein `DataSet`. Der Konstruktor der `XmlDataDocument`-Klasse wird mit einem `DataSet`-Objekt aufgerufen.

```
DD = New System.Xml.XmlDataDocument(DS)
```

Danach wirken alle Änderungen in dem `XmlDataDocument`-Objekt sofort auf das `DataSet` und umgekehrt. Um ein `XmlDataDocument`-Objekt beschreiben zu können, ist Voraussetzung, dass in dem zugehörigen `DataSet` die Konsistenzprüfungen ausgeschaltet sind.

```
DS.EnforceConstraints = False
```

Beispiel**DataSet via XML bearbeiten**

In der folgenden Routine wird zunächst ein `DataSet` mit einer Tabelle (»SELECT * FROM Produkte where name like 'Amiga%'«) erzeugt. Daraus wird ein `XmlDataDocument`-Objekt abgeleitet. Dieses Dokument wird mit den Methoden des XML-DOM ausgegeben (`GetOuterXML()`), durchsucht (`XN=SelectSingleNode()`) und geändert (`XN.InnerText = WERT`). Zum Beweis, dass diese Änderungen auf das `DataSet` durchschlagen, wird das `DataSet` in XML-Form ausgegeben (`DS.GetXml()`). Die Änderungen werden zum Schluss mit Hilfe eines `OleDbCommandBuilder`-Objekts an die Datenquelle (hier: die Access-Datenbank *EDVShop.mdb*) zurückgesandt.

```
' =====
' Beispiel aus FCL-Buch.sln
' Veränderung eines DataSet-Objekts via XML
' (C) Holger@Schwichtenberg.de
' =====

Sub ADONET_DSXML2()
    Dim DS As System.Data.DataSet
    Dim Com As System.Data.OleDb.OleDbCommand
    Dim SQL As String

    outtitle("Veränderung eines DataSet-Objekts via XML")

    ' --- DataSet erzeugen
    DS = New System.Data.DataSet()

    ' --- Tabelle einlesen
    SQL = "SELECT * FROM Produkte where name Like
'Amiga%"
    Dim DA As System.Data.OleDb.OleDbDataAdapter
    DA = New System.Data.OleDb.OleDbDataAdapter(SQL,
CONNSTRING)
    DA.Fill(DS, "EDV-Antik")
```

```
' --- DataDocument erzeugen
Dim DD As System.Xml.XmlDataDocument
DD = New System.Xml.XmlDataDocument(DS)

' --- bestimmtes Element finden
Dim XN As System.Xml.XmlNode
XN = DD.SelectSingleNode("*/EDV-Antik[Name='Amiga
500']/Preis")

' --- Änderungen zulassen
DS.EnforceConstraints = False

' --- alten Wert auslesen
Dim alterpreis As Double
alterpreis = CType(XN.InnerText, Double)

' --- Neuen Wert ermitteln
Dim neuerpreis As Double
neuerpreis = alterpreis * 1.5

' --- Neuen Wert setzen
XN.InnerText = neuerpreis.ToString

' --- Ausgabe der Änderung
out("Preis wurde von " & alterpreis & " auf " & neuerpreis & " erhöht!")

' --- Kontrollausgabe des DataSets
out("--- DataSet nach Änderung:")
out(DS.GetXml)

' --- Speicherung vorbereiten
Dim CB As System.Data.OleDb.OleDbCommandBuilder
CB = New OleDbCommandBuilder(DA)

' --- Speicherung des DataSets
DA.Update(DS, "EDV-Antik")
out("Änderungen gespeichert!")

End Sub
```

Listing 10.16: Veränderung eines DataSet-Objekts via XML [ADONET_XML.vb]

Ausgabe

```

1  Veränderung eines DataSet-Objekts via XML
2  <NewDataSet><EDV-Antik><ID>3</ID><Name>Amiga 500</
Name><Kategorie>23</Kategorie><Preis>750</Preis></EDV-
Antik><EDV-Antik><ID>6</ID><Name>Amiga 1000</
Name><Kategorie>23</Kategorie><Preis>2662</Preis></EDV-
Antik></NewDataSet>
3  Preis wurde von 750 _ auf 1125 _ erhöht!
4  --- DataSet nach Änderung:
5  <NewDataSet>
    <EDV-Antik>
      <ID>3</ID>
      <Name>Amiga 500</Name>
      <Kategorie>23</Kategorie>
      <Preis>1125</Preis>
    </EDV-Antik>
    <EDV-Antik>
      <ID>6</ID>
      <Name>Amiga 1000</Name>
      <Kategorie>23</Kategorie>
      <Preis>2662</Preis>
    </EDV-Antik>
  </NewDataSet>
6  Änderungen gespeichert!
```

10.11.3 XML in DataSet einlesen

XML-Dokument in DataSet einlesen

Im Gegensatz zum klassischen ADO kann in ADO.NET jedes beliebige XML-Dokument in ein DataSet eingelesen werden. Eine bestimmte Struktur ist nicht notwendig. Allerdings darf es nicht zwei gleichnamige Elemente an verschiedenen Stellen im XML-Dokument geben; z.B. darf `<Name>` nicht Unterelement von `<Person>` sein und an anderer Stelle im Dokument Unterelement von `<Firma>`. Die Elementnamen müssen also im gesamten Dokument eindeutig sein.

Beispiel

Die in der folgenden Grafik dargestellte XML-Datei *hs.xml* soll in ein DataSet eingelesen werden.

Die XML-Datei wird geöffnet, mit einem Stream gelesen und dann per `ReadXml()` in ein DataSet eingelesen. Zuerst werden alle in dem DataSet enthaltenen Tabellen einschließlich ihrer untergeordneten Tabellen aufgelistet. Die Ausgabe zeigt, dass das DataSet beim Laden der XML-Datei jedes Element als eine Tabelle anlegt und die jeweils untergeordneten Elemente über eine `DataRelation` verknüpft.


```
<?xml version="1.0" encoding="utf-8"?>
<Person>
  <Name>Holger Schwichtenberg</Name>
  <PersDaten>
    <Geb>01.08.1972</Geb>
    <Wohnort>Essen</Wohnort>
  </PersDaten>
  <Publikationen>
    <Buecherliste>
      <Buch bid="1">
        <Titel>Windows Scripting</Titel>
        <Verlag>Addison-Wesley</Verlag>
        <Jahr>2001</Jahr>
      </Buch>
      <Buch bid="2">
        <Titel>COM-Komponenten-Handbuch</Titel>
        <Verlag>Addison-Wesley</Verlag>
        <Jahr>2001</Jahr>
      </Buch>
      <Buch bid="3">
```

Abbildung 10.9: Ausschnitt aus der Datei hs.xml

Danach erfolgt ein gezielter Zugriff auf die Tabelle »Buch«; dies sind alle Buch-Elemente aus dem eingelesenen XML-Dokument. Diese Tabelle wird über die Rows-Collection ausgegeben und anschließend an ein Windows Forms-DataGrid gebunden. Weitere Informationen zu Windows Forms finden Sie in Kapitel 17.

**Bindung an
DataGrid**

```
' =====
' Beispiel aus FCL-Buch.sln
' Einlesen von XML In ein DataSet
' (C) Holger@Schwichtenberg.de
' =====
Sub ADONET_DSXML3()

    Dim DS As New System.Data.DataSet()
    Dim fs As FileStream
    Dim sr As StreamReader
    Dim DR As DataRow
    Dim DT As DataTable

    ' --- Datei öffnen
    fs = New FileStream("../DATEN/xml\hs.xml", _
        FileMode.Open, FileAccess.Read)
    ' --- Datei einlesen
    sr = New StreamReader(fs)
    ' --- Stream In DataSet einlesen
    DS.ReadXml(sr)

    ' --- Schleife über alle Tabellen
    out("Tabellen:")
    For Each DT In DS.Tables
```

```

        out(DT.TableName)
        For Each REL In DT.ChildRelations
            out(" - " & REL.ChildTable.TableName)
        Next
    Next

    ' --- Zugriff auf einzelne Tabelle
    DT = DS.Tables("buch")
    ' --- Anzahl der Datensätze ausgeben
    out("Anzahl: " & DT.Rows.Count)
    ' --- Schleife über alle Datensätze
    For Each DR In DT.Rows
        out(DR("Titel").ToString & " erschienen In " &
DR("Jahr").ToString)
    Next

    ' --- Ausgabe in ein DataGrid
    Dim f As New FORM_Buchliste()
    f.DataGrid1.DataSource = DT
    f.Show()
End Sub

```

Listing 10.17: Einlesen von XML in ein DataSet [ADONET_XML.vb]

Ausgabe

```

1 Tabellen:
2 Person
3   - PersDaten
4   - Publikationen
5   - Vortraege
6 PersDaten
7 Publikationen
8   - Buecherliste
9   - Fachartikelliste
10 Buecherliste
11   - Buch
12 Buch
13 Fachartikelliste
14 Vortraege
15   - Konferenzen
16 Konferenzen
17   - Konferenz
18 Konferenz
19 Anzahl: 3
20 Windows Scripting erschienen in 2001
21 COM-Komponenten-Handbuch erschienen in 2001
22 Programmieren mit der .NET-Klassenbibliothek
   erschienen in 2002

```

	Titel	Verlag	Jahr	bid
▶	Windows Scripting	Addison-Wesley	2001	1
	COM-Komponenten-Handbuch	Addison-Wesley	2001	2
	Programmieren mit der .NET-Klassenbibliothek	Addison-Wesley	2002	3
*				

Abbildung 10.10: Anzeige des DataGrids [FORM_Buchliste.vb]

10.12 Weitere Möglichkeiten von ADO.NET

Einige weitere Möglichkeiten von ADO.NET können hier als Platzgründen nicht besprochen werden:

- ▶ Die `DataTable`-Klasse bietet insgesamt sechs Ereignisse, mit denen sie Veränderungen an ihre Umgebung melden kann: `ColumnChanged()`, `ColumnChanging()`, `RowChanged()`, `RowChanging()`, `RowDeleted()`, `RowDeleting()`. Ein `Changing`-Ereignis tritt während einer Änderung, ein `Changed`-Ereignis nach Abschluss einer Änderung auf.
- ▶ Ein Objekt der Klasse `System.Data.DataView` repräsentiert eine gefilterte Sicht auf ein `DataSet`.
- ▶ Die Data Provider stellen Klassen zur Ausführung von Transaktionen bereit (z.B. `OleDbTransaction`). Eine Instanz dieser Transaktions-Klasse kann an das Attribut `Transaction` der passenden Verbindungs-Klasse (z.B. `OleDbConnection`) zugewiesen werden. Die wichtigsten Methoden der Transaktions-Klasse sind `BeginTransaction()`, `Commit()` und `Rollback()`.
- ▶ Visual Studio .NET bietet an zahlreichen Stellen Unterstützung für den Datenzugriff per ADO.NET: Erstellung von Verbindungszeichenfolgen, graphischer Entwurf von SQL-Befehlen, Generierung von Code für `DataSets` und Datenadapter, Generierung von datengebundenen Windows-Fenster (Windows Forms) und Web-Seiten (Web Forms).

Tabellenereignisse

Sichten

Transaktionen

II Namespaces

System.Runtime.Serialization und System.Xml.Serialization

Die FCL bietet Klassen, um Objekte zu serialisieren und zu deserialisieren. Serialisierung bedeutet die Umwandlung des Zustands eines Objekts in eine Byte-Folge. Deserialisierung ist der umgekehrte Vorgang, bei dem aus einem in eine Byte-Folge serialisierten Objekt wieder ein programmierbares Objekt erzeugt wird. Dabei geht es darum, den ursprünglichen Zustand des Objekts wiederherzustellen.

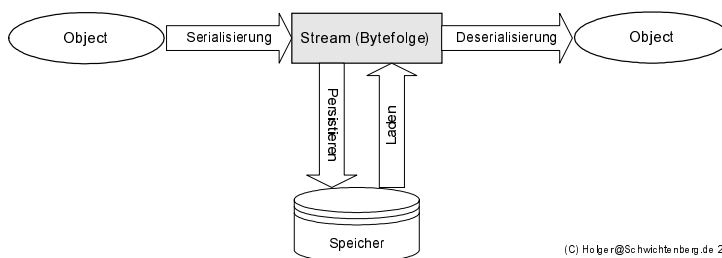
Serialisierung

Serialisierung ist eine Möglichkeit, Objekte auf komfortable Weise persistent zu machen (dauerhaft zu speichern), z.B. in einer Datei oder einer Datenbank.

Persistenz

Andere Einsatzgebiete für Serialisierung sind:

- Übertragung eines Objekts in einen anderen Prozess (Remoting)
- Realisierung von Undo-Funktionen (Wiederherstellung eines vorherigen Zustandes)



(C) Holger@Schwichtenberg.de 2002

Abbildung 11.1: Serialisierung und Persistenz

11.1 Serialisierer in .NET

Formate Die .NET Framework Class Library bietet drei Serialisierungs-Formate:

- Binär
- Extensible Markup Language (XML)
- Simple Object Access Protocol (SOAP)-Nachricht

Es werden nicht nur die öffentlichen Attribute, sondern auch die privaten globalen Variablen eines Objekts gesichert. Die Serialisierung wird dabei auch für Objekthierarchien unterstützt. Mit dem binären Serialisierer können auch Objektmodelle mit zirkulären Referenzen serialisiert werden. Der XML-Serialisierer unterstützt keine zirkulären Referenzen.

Serialisierer	Klasse	Assembly
Binärer Serialisierer	System.Runtime. Serialization. Formatters.Binary	<i>mscorlib.dll</i>
SOAP-Serialisierer	System.Runtime. Serialization. Formatters.Soap. SoapFormatter	<i>System.Runtime. Serialization. Formatters. Soap.dll</i>
XML-Serialisierer	System.Xml. Serialization. XmlSerialization	<i>System. Xml.dll</i>

Tabelle 11.1: Serialisierer in der FCL



Der XML-Serialisierer hebt sich nicht nur vom Namen (`XmlSerializer` statt »XMLFormatter«) her, sondern auch in Funktionsweise und Bedienung von den anderen beiden Serialisierern ab.

Voraussetzung

Meta-Attribut `Serializable()` Ein Objekt kann nur serialisiert werden, wenn die Klasse mit dem Meta-Attribut `<Serializable(>>` versehen wurde. Es gibt – besonders für den XML-Serialisierer – weitere Attribute für die Steuerung der Serialisierung.

Beispiel

Die Anwendung der Serialisierer wird zunächst anhand allgemeiner Unterrouتين und eines einzelnen Objekts gezeigt. Dabei wird folgende Klassendefinition verwendet:

Beispielklasse

```
' =====
' Beispiel aus FCL-Buch.sln
' Klassendefinition für einfache
' Serialisierungs-Beispiele
' (C) Holger@Schwichtenberg.de
' =====
<Serializable> Public Class Person
    Public Name As String
    Public Vorname As String
    Public Geb As Date
    Public Geschlecht As Char
    Public Kinderanzahl As Byte
    Public Taetigkeiten As String()
End Class
```

Listing 11.1: Definition einer einfachen Klasse in Visual Basic .NET zur Verwendung in den folgenden Beispielen

Später in diesem Kapitel folgt ein Beispiel mit einem kleinen Objektmodell (siehe Kapitel 1.5).

11.2 Binärer Serialisierer

Der binäre Serialisierer wandelt ein beliebiges Objekt bzw. eine Objekthierarchie in eine binäre Zeichenfolge um. Zirkuläre Referenzen in Objekthierarchien erkennt der Serialisierer und verarbeitet diese korrekt.

**Binäre
Zeichenfolgen**

Für die folgenden Unterrouتين wird neben dem Namespace `System.Runtime.Serialization.Formatters.Binary` auch der Namespace `System.IO` benötigt, da die serialisierten Objekte in einer Textdatei persistent gemacht und später von dort wieder gelesen werden.

Für die Serialisierung wird nur ein einziger Methodenaufruf benötigt: `Serialize()` erwartet im ersten Parameter ein Stream-Objekt (ein Objekt einer Klasse, die von `System.IO.Stream` abgeleitet ist, z.B. `FileStream`, `MemoryStream`, `NetworkStream`, `CryptoStream`, `BufferedStream`) und im zweiten Parameter das zu serialisierende Objekt. Als Stream wird in der folgenden Unterroutine eine Datei verwendet. `bin_serialize()` kapselt diese Funktionalität.

Serialize()

Deserialize() Auch für die Deserialisierung ist nur ein Methodenaufruf notwendig. `Deserialize()` erzeugt aus einem übergebenen Stream ein Objekt, ggf. mit allen Unterobjekten.



Der parameterlose Konstruktor wird beim Deserialisieren nicht aufgerufen!

11.2.1 Serialisieren in Binärdatei

bin_serialize() In dem folgenden Listing wird mit der `FileStream`-Klasse in eine Datei serialisiert (persistent gemacht).

```
' --- Serialisieren In Binärdatei
Sub bin_serialize(ByVal o, ByVal datei)
    Dim form As New BinaryFormatter()
    Dim stream As FileStream
    stream = New FileStream(datei, _
        FileMode.Create, FileAccess.Write, _
        FileShare.None)
    form.Serialize(stream, o)
    stream.Close()
    out("Objekt wurde gespeichert In " & datei)
End Sub
```

Listing 11.2: Hilfsroutine zur binären Serialisierung und anschließender Speicherung in einer Datei [Serialisierer_Funktionen.vb]

Deserialisieren aus einer Binärdatei

bin_deserialize() `bin_deserialize()` kapselt die Deserialisierung aus einer Datei.

```
' --- Deserialisieren aus einer Binärdatei
Function bin_deserialize(ByVal Datei As String) _
    As Object
    Dim o As Object
    Dim form As New BinaryFormatter()
    Dim stream As FileStream
    stream = New FileStream(Datei, _
        FileMode.Open, FileAccess.Read, _
        FileShare.Read)
    o = form.Deserialize(stream)
    stream.Close()
    out("Objekt wurde geladen aus " & Datei)
    Return o
End Function
```

Listing 11.3: Hilfsroutine zur Deserialisierung eines Objekts aus einer Binärdatei [Serialisierer_Funktionen.vb]

Anwendungsbeispiel

Im folgenden Beispiel wird eine Instanz der Klasse `Person` in einer Datei persistent gemacht und dann von dort wieder geladen.

»Person« serialisieren

```
' =====
' Beispiel aus FCL-Buch.sln
' Beispiel für binäres Serialisieren
' (C) Holger@Schwichtenberg.de
' =====
Sub Bin_Serialisieren_Test()
    ' --- Objekt erzeugen und füllen
    Dim p As New Person()
    p.Vorname = "Holger"
    p.Name = "Schwichtenberg"
    p.Geb = #8/1/1972#
    p.Geschlecht = "m"
    p.Kinderanzahl = 0

    ' --- Serialisierungsziel
    Const DATEI = "h:\data\person.bin"

    ' --- binäres Serialisieren
    bin_serialize(p, DATEI)
    p = bin_deserialize(DATEI)

    ' --- Ausgabe des Objekts zum Test
    out(p.Vorname & " " & p.Name & " wurde am " & _
        p.Geb & " geboren!")
End Sub
```

Listing 11.4: Testroutine für die Funktionen `bin_serialize()` und `bin_deserialize()` anhand der Klasse »Person«

Ausgabe

Die obige Routine liefert die nachfolgend dargestellten Ausgaben.

Ergebnis

```
1 Objekt wurde gespeichert in h:\data\person.bin
2 Objekt wurde geladen aus h:\data\person.bin
3 Holger Schwichtenberg wurde am 01.08.1972 geboren!
```

Listing 11.5: Ausgabe des obigen Beispiels

11.2.2 Serialisieren in Byte-Folge oder Datenbank

Flexiblere Möglichkeiten als die direkte Serialisierung in eine Datei eröffnet die Serialisierung innerhalb des Speichers in eine Byte-Folge, die dann beliebig weiterverwendet (z.B. auch in ein BLOB-Feld einer Datenbank geschrieben) werden kann.

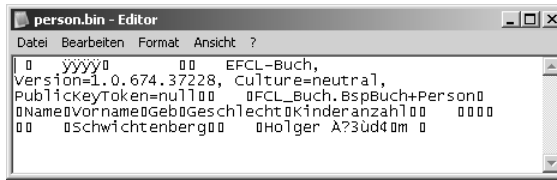


Abbildung 11.2: Ansicht der Binärdatei im Texteditor

Serialisierung

MemoryStream

Die Möglichkeit, eine Byte-Folge zu gewinnen, eröffnet die Klasse `System.IO.MemoryStream`. Das `MemoryStream`-Objekt benötigt keinen Parameter, die eigentliche Serialisierung mit `Serialize()` ist gleich. Zur Gewinnung der Byte-Folge in Form eines `Array of Byte` muss die Methode `ToArray()` auf dem `MemoryStream`-Objekt aufgerufen werden.

Die folgende Hilfsroutine ist eine Variante der bereits vorgestellten Serialisierung in eine Datei. Die Funktionsnamen sind gleich, durch die unterschiedlichen Parameter können die Funktionen jedoch unterschieden werden (Überladung).

```
' ### Serialisieren In Byte-Folge
Function bin_serialize(ByVal o As Byte()) As Byte()
    Dim form As New BinaryFormatter()
    Dim stream As New MemoryStream()
    form.Serialize(stream, o)
    stream.Close()
    out("Objekt serialisiert!")
    Return (stream.ToArray())
End Function
```

Listing 11.6: Serialisieren in eine Byte-Folge [Serialisierer_Funktionen.vb]

Deserialisierung

Beim Deserialisierungsvorgang aus einer Byte-Folge kann ausgenutzt werden, dass ein `MemoryStream`-Objekt direkt aus einer Byte-Folge erzeugt werden kann.

```
' ### Deserialisieren aus einer Byte-Folge
Function bin_deserialize(ByVal ba As Byte()) _
    As Object

    Dim o As Object
    Dim form As New BinaryFormatter()
    Dim stream As New MemoryStream(ba)
    o = form.Deserialize(stream)
```

```

    stream.Close()
    out("Objekt wurde deserialisiert")
    Return o
End Function

```

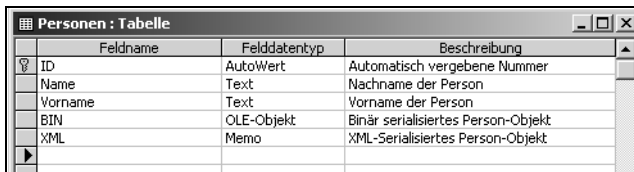
Listing 11.7: Serialisieren aus einer Byte-Folge [Serialisierer_Funktionen.vb]

Beispiel

Das folgende Beispiel speichert ein Person-Objekt in ein BLOB-Feld in einer Datenbank-Tabelle ab. Zuvor wird das Objekt mit der Hilfsroutine `bin_serialize()` in eine Byte-Folge umgewandelt.

Serialisieren in BLOB-Feld

Die folgende Abbildung zeigt die Tabelle »Personen« in der Access-Datenbank *Lieferanten.mdb* [CD:/Code/FCL-Buch/_Daten/adonet/Lieferanten.mdb]. Neben dem BLOB-Feld (in Access Feldtyp »OLE-Objekt«) wurden die Felder für Nachname und Vorname angelegt, um nach einer bestimmten Person suchen zu können. Das Feld »XML« ist für die im nächsten Kapitel besprochene XML-Serialisierung reserviert.



Feldname	Felddatentyp	Beschreibung
ID	AutoWert	Automatisch vergebene Nummer
Name	Text	Nachname der Person
Vorname	Text	Vorname der Person
BIN	OLE-Objekt	Binär serialisiertes Person-Objekt
XML	Memo	XML-Serialisiertes Person-Objekt

Abbildung 11.3: Datenbanktabelle für Objektpersistenz

Als Verbindungszeichenfolge kommt zum Einsatz:

```

Public Const CONNSTRING As String = _
    "Provider=Microsoft.Jet.OLEDB.4.0;
    Data Source=..\_DATEN\adonet\Lieferanten.mdb;"

```

Die zum Einsatz kommenden ADO.NET-Klassen werden in Kapitel 10 besprochen.

```

' =====
' Beispiel aus FCL-Buch.sln
' Objektpersistenz In Datenbank
' (C) Holger@Schwichtenberg.de
' =====
Sub DB_Persistenz1()

    outtitle( _
        "Objekt In Datenbank persistent machen")

```

```

' --- Objektdefinition
Dim p As New Person()
p.Vorname = "Holger"
p.Name = "Schwichtenberg"
p.Geb = #8/1/1972#
p.Geschlecht = "m"
p.Kinderanzahl = 0
p.Taetigkeiten = New String() _
    {"Berater", "Entwickler", "Journalist", _
     "Dozent"}

' --- Serialisieren In binäre Byte-Folge
Dim ba As Byte()
ba = bin_serialize(p)

Dim DS As System.Data.DataSet
Dim DT As System.Data.DataTable
Dim DR As System.Data.DataRow
Dim DA As System.Data.OleDb.OleDbDataAdapter
Dim CB As _
    System.Data.OleDb.OleDbCommandBuilder

' --- DataSet erzeugen
DS = New System.Data.DataSet()
' --- Adapter definieren
Const SQL As String = _
    "SELECT * FROM Personen where ID = 0"
DA = New OleDbDataAdapter(SQL, CONNSTRING)
' --- SQL-DML-Befehle erzeugen
CB = New OleDbCommandBuilder(DA)

out("INSERT-Befehl: " & _
    CB.GetInsertCommand.CommandText())
' --- Tabelle einlesen
DA.Fill(DS, "Personen")

' --- Zugriff auf Tabelle
DT = DS.Tables("Personen")

' --- ANFÜGEN EINES DATENSATZES
DR = DT.NewRow()
' Füllen der Spalten
DR("Name") = p.Name
DR("Vorname") = p.Vorname
DR("BIN") = ba

' Anfügen der Zeile
DT.Rows.Add(DR)
' Speichern der Änderungen
DA.Update(DS, "Personen")

```

```

        DS.Dispose()

        out("Objekt wurde In Datenbank gespeichert!")
    End Sub

```

Listing 11.8: Objektpersistenz in Datenbank [Serialisierung_DB.vb]

11.3 SOAP-Serialisierer

Der SOAP-Serialisierer funktioniert genauso wie der binäre Serialisierer. Sie müssen in den Beispielen aus dem letzten Kapitel lediglich die Klasse `System.Runtime.Serialization.Formatters.Binary` gegen die Klasse `System.Runtime.Serialization.Formatters.Soap` austauschen.

**Soap.Soap-
Formatter**

```

' ### SOAP-Serialisierung In Binärdatei
Sub soap_serialize(ByVal o, ByVal datei)
    Dim form As New SoapFormatter()
    Dim stream As FileStream
    stream = New FileStream(datei, _
        FileMode.Create, FileAccess.Write, _
        FileShare.None)
    form.Serialize(stream, o)
    stream.Close()
    out("Objekt wurde gespeichert In " & datei)
End Sub

```

Listing 11.9: SOAP-Serialisierung in Binärdatei [Serialisierer_Funktionen.vb]

```

' ### SOAP-Deserialisieren aus einer Binärdatei
Function soap_deserialize(ByVal Datei As String) _
    As Object

    Dim o As Object
    Dim form As New SoapFormatter()
    Dim stream As FileStream
    stream = New FileStream(Datei, FileMode.Open)
    o = form.Deserialize(stream)
    stream.Close()
    out("Objekt wurde geladen aus " & Datei)
    Return o
End Function

```

Listing 11.10: SOAP-Deserialisieren aus einer Binärdatei [Serialisierer_Funktionen.vb]

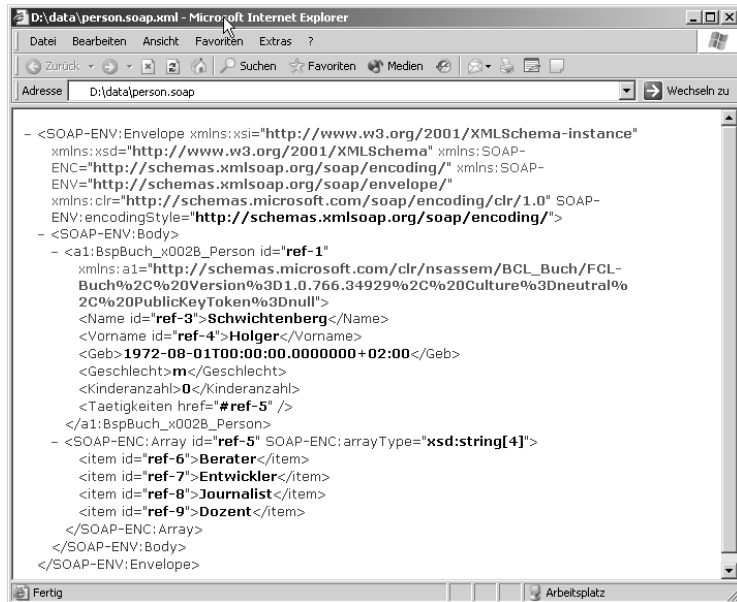


Abbildung 11.4: Ergebnis der Serialisierung in SOAP

11.4 XML-Serialisierer

XML-Dokument Der XML-Serialisierer (Klasse `System.Xml.Serialization.XmlSerializer`) erzeugt aus einem Objekt bzw. einer Objekthierarchie einen ASCII-Byte-Strom im XML-Format, also ein XML-Dokument.

Vergleich zu den anderen Serialisierern

XmlSerializer Der XML-Serialisierer hebt sich nicht nur vom Namen (`XmlSeriali` zer statt »XMLFormatter«) her, sondern auch in einigen weiteren Punkten von den anderen beiden Serialisierern ab:

- ▶ Die Klasse implementiert nicht die Schnittstellen `IRemotingFormatter` und `IFormatter`.
- ▶ Die Klasse ist im Namespace `System.XML` und nicht in `System.Runtime.Serialization.Formatters` realisiert.
- ▶ Beim Instanzieren der Klasse `XmlSerializer` muss ein Objekt vom Typ `System.Type` übergeben werden, das den Typ des zu (de-)serialisierenden Objekts repräsentiert.

- ▶ Im Gegensatz zu den anderen beiden Serialisierern werden keine privaten Mitglieder serialisiert.
- ▶ Im Gegensatz zu den anderen beiden Serialisierern ruft der XML-Serialisierer beim Deserialisieren den parameterlosen Konstruktor der Klasse des zu deserialisierenden Objekts auf.
- ▶ Der XML-Serialisierer serialisiert folgende FCL-Klassen:
 - alle Klassen, die `ICollection` und `IEnumerable` implementieren
 - `System.Xml.XmlElement`
 - `System.Xml.XmlNode`
 - `System.Data.DataSet`

Es ist aber nicht möglich, ein komplettes `XmlDocument`-Objekt (auch nicht als Teil eines anderen Objekts) auf diese Weise zu serialisieren.

- ▶ Der XML-Serialisierer beantwortet leider den Versuch, eine Objekthierarchie mit zirkulären Referenzen zu serialisieren, mit der Fehlermeldung »*There was an error generating the XML document. ---> System.InvalidOperationException: A circular reference was detected while serializing an object.*« Der Grund dafür ist, dass der XML-Serialisierer einen einzelnen Baum von XML-Elementen erzeugt, in dem jedes Objekt an die Stelle einsortiert wird, an der es referenziert wird. Bei mehrfachen Referenzen wird es mehrfach eingeführt (siehe Kapitel 1.5).
- ▶ Die Steuerung der Serialisierung erfolgt über Meta-Attribute, nicht aber über Schnittstellen (siehe Kapitel 1.6).
- ▶ Die Methode `serialize()` kann nicht nur in Streams, sondern auch in von `System.IO.TextWriter` und `System.Xml.XmlWriter` abgeleitete Klassen serialisieren. Umgekehrt arbeitet `deserialize()` auch mit von `System.IO.TextReader` und `System.Xml.XmlReader` abgeleiteten Klassen.

11.4.1 XML-Serialisierung

Die Klasse `XmlSerializer` erwartet bei ihrer Instanziierung ein Type-Objekt mit dem Typ des Objekts, das serialisiert werden soll. Das weitere Vorgehen ist dann analog zum Binär-Serialisierer; der XML-Serialisierer bietet jedoch mehr Steuerungsmöglichkeiten für die Ausgabeform.

Serialize()

Die `Serialize()`-Methode in der Klasse `XmlSerializer` akzeptiert als Ausgabestrom wahlweise ein Objekt des Typs `Stream`, `TextWriter` oder `XMLWriter`.

Serialisierung in XML-Datei

xml_serialize() In der Hilfsroutine `xml_serialize()` ist der Ausgabestrom ein `StreamWriter`, eine Unterklasse von `TextWriter`. Der zweite Parameter der `Serialize()`-Methode ist das zu serialisierende Objekt.

```
' --- Serialisieren In XML-Datei
Sub xml_serialize(ByVal obj As Object, _
    ByVal datei As String)

    Dim serializer As New XmlSerializer(obj.GetType)
    Dim writer As New StreamWriter(datei)
    serializer.Serialize(writer, obj)
    writer.Close()
    out("Objekt wurde gespeichert In " & datei)
End Sub
```

Listing 11.11: Hilfsroutine zur XML-Serialisierung und anschließender Speicherung des XML-Dokuments in einer Datei [Serialisierung_Funktionen.vb]

Serialisierung in XML-String

Stream in String umwandeln

Leider kann die `Serialize()`-Methode nur mit `Stream`- und `Writer`-Objekten, nicht aber nicht mit `String`-Objekten arbeiten. Zur In-Memory-Serialisierung in einen `String` muss die `MemoryStream`-Klasse verwendet werden. Da es hier aber keine direkte Methode gibt, um den `Stream`-Inhalt als `String` zu erhalten, gibt es zwei Wege:

1. Lesen des `MemoryStreams` über ein `StreamReader`-Objekt oder
2. Umwandeln des `MemoryStreams` in eine `Byte`-Folge, zeichenweise Umwandlung in ein `Char`-Objekt und Zusammensetzen zu einem `String`



Ein so gewonnener `XML-String` kann z.B. in eine Datenbank gespeichert werden. Dies ist in der Datei `[Serialisierung_DB.vb]`, die Sie auf der CD-ROM zum Buch finden, gezeigt.

```
' ### XML-Serialisieren In String
Function xml_serialize(ByVal obj As Object) As String
    Dim serializer As New XmlSerializer(obj.GetType)
    Dim s As String

    ' --- Serialisieren In MemoryStream
    Dim ms As New MemoryStream()
```



```

serializer.Serialize(ms, obj)
out("Objekt wurde serialisiert!")

' --- Stream In String umwandeln
Dim r As StreamReader = New StreamReader(ms)
r.BaseStream.Seek(0, SeekOrigin.Begin)
s = r.ReadToEnd

' Alternative:
'Dim b As Byte
'Dim ba As Byte()
'ba = ms.ToArray
'For Each b In ba
'    s = s & Convert.ToChar(b)
'Next

Return s
End Function

```

Listing 11.12: XML-Serialisieren in String [Serialisierung_Funktionen.vb]

11.4.2 XML-Deserialisierung

`Deserialize()` wird auch analog zum Binär-Serialisierer verwendet. Zu beachten ist, dass die `XmlSerializer`-Klasse bei der Deserialisierung den Typ des zu deserialisierenden Objekts erwartet.

`xml_deserialize()`

Deserialisierung aus XML-Datei

Die Hilfsroutine `xml_deserialize()` benötigt für die Instanziierung der `XmlSerializer`-Klasse neben dem Dateinamen einen weiteren Parameter mit einem entsprechenden `Type`-Objekt. Für das Einlesen der XML-Datei kommt hier ein `StreamReader` zum Einsatz. Alternativ könnte ein `Stream`- oder ein `XmlReader`-Objekt verwendet werden.

```

' --- Deserialisieren aus XML-Datei
Function xml_deserialize(ByVal t As Type, _
    ByVal datei As String) As Object
    Dim obj As Object
    Dim reader As New StreamReader(datei)
    Dim serializer As New XmlSerializer(t)
    obj = serializer.Deserialize(reader)
    out("Objekt wurde geladen aus " & datei)
    reader.Close()
    Return (obj)
End Function

```

Listing 11.13: Hilfsroutine zur Deserialisierung eines Objekts aus einer XML-Datei [Serialisierung_Funktionen.vb]

Deserialisierung aus XML-String

String in Stream umwandeln

Bei der Deserialisierung aus einer Zeichenkette, die ein XML-Dokument enthält, ist ein kleiner unschöner Winkelzug notwendig. Grundsätzlich kann die Methode `Deserialize()` genauso wenig wie `Serialize()` direkt einen String verarbeiten. Daher ist ein `MemoryStream` notwendig. Um diesen zu beschreiben, verwendet man ein `StreamWriter`-Objekt. Dieses hat die Eigenschaft, dass es sein Werk nur dann zu Ende bringt, wenn es ordnungsgemäß geschlossen wird. Mit dem Schließen des `StreamWriter`-Objekts mit `Close()` wird aber gleichzeitig auch der Stream geschlossen, weshalb der Deserialisierer nicht mehr darauf zugreifen kann. Der Inhalt des `MemoryStream`-Objekts steht aber noch via `ToArray()` zur Verfügung. Daher muss der Inhalt in ein anderes `MemoryStream`-Objekt umkopiert werden.

```
' ### Deserialisieren aus XML-String
Function xml_deserialize_String(ByVal t As Type, _
    ByVal s As String) As Object

    Dim obj As Object
    ' --- Objekt In Stream kopieren
    Dim stream As New MemoryStream()
    Dim w As New StreamWriter(stream)
    w.BaseStream.Seek(0, SeekOrigin.End)
    w.WriteLine(s)
    w.Close()
    ' Stream umkopieren, weil jetzt geschlossen
    stream = New MemoryStream(stream.ToArray())
    Dim serializer As New XmlSerializer(t)
    obj = serializer.Deserialize(stream)
    out("Objekt wurde deserialisiert!")
    stream.Close()
    Return (obj)
End Function
```

Listing 11.14: Deserialisieren aus XML-String [Serialisierung_Funktionen.vb]

11.4.3 Anwendungsbeispiel

Serialisierung der Klasse »Person«

Im folgenden Beispiel wird eine Instanz der Klasse `Person` in einer Datei persistent gemacht und dann von dort wieder geladen.

```
' =====
' Beispiel aus FCL-Buch.sln
' Beispiel für XML-Serialisieren
' (C) Holger@Schwichtenberg.de
' =====
Sub XML_Serialisieren_Test()
```

```

' --- Objekt erzeugen und füllen
Dim p As New Person()
p.Vorname = "Holger"
p.Name = "Schwichtenberg"
p.Geb = #8/1/1972#
p.Geschlecht = "m"
p.Kinderanzahl = 0
p.Taetigkeiten = New String() {"Berater", _
    "Entwickler", "Journalist", "Dozent"}

' --- Serialisierungsziel
Const DATEI = "h:\Data\person.xml"

' --- binäres Serialisieren
xml_serialize(p, DATEI)
p = xml_deserialize(GetType(Person), DATEI)

' --- Ausgabe des Objekts zum Test
out(p.Vorname & " " & p.Name & " wurde am " & _
    p.Geb & " geboren!")
End Sub

```

Listing 11.15: Testroutine für die Funktionen `xml_serialize()` und `xml_deserialize()` anhand der Klasse »Person«

Ausgabe

Die obige Routine liefert die nachfolgend dargestellten Ausgaben.

- 1 Objekt wurde gespeichert in `h:\data\person.xml`
- 2 Objekt wurde geladen aus `h:\data\person.xml`
- 3 Holger Schwichtenberg wurde am 01.08.1972 geboren!

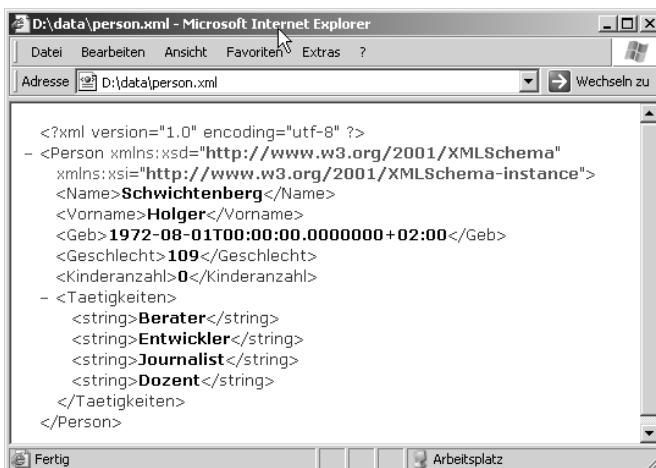


Abbildung 11.5: Die Ausgabedatei `person.xml`

11.5 Serialisierung eines Objektmodells

Beispiel-Klassen- hierarchie

Im folgenden Beispiel wird die Serialisierung/Deserialisierung an einem kleinen Objektmodell demonstriert: Ein `Autor`-Objekt besitzt eine Objektmenge vom Typ `ArrayList`, in der seine Bücher (Typ `Buch`) aufgelistet sind.

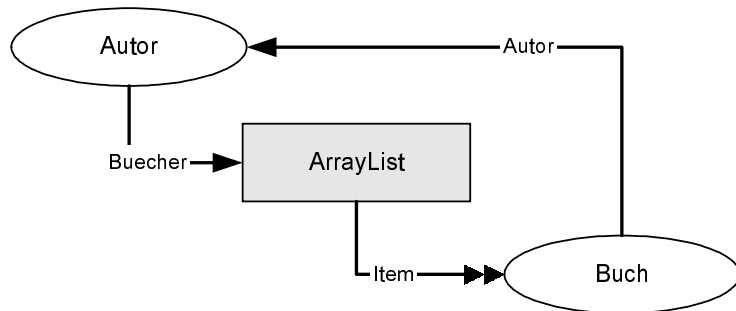


Abbildung 11.6: Objekthierarchie der Klassen »Autor« und »Buch« für die folgenden Beispiele

Klassendefinition

Implementierung der Objekt- hierarchie

Das folgende Listing zeigt die Definition der o.g. Klassen.

```

<Serializable(>> Class Buch
    Public Titel As String
    Public Autor As Autor
    Public Auflage As Integer

    Function Get_auflage() As Integer
        Return (Auflage)
    End Function

    Sub Set_auflage(ByVal a As Integer)
        Auflage = a
    End Sub
End Class

<Serializable(), XmlInclude(GetType(Buch))> _
Class autor
    Public name As String
    Public geb As Date
    Public buecher As New ArrayList()
End Class
  
```

Listing 11.16: Implementierung der Klassen »Autor« und »Buch« in Visual Basic .NET

Für den XML-Serialisierer ist es wichtig, dass er weiß, welcher Objekttyp ihn in der ArrayList erwartet. Daher muss mit dem Meta-Attribut `<XmlInclude(GetType(Buch))>` definiert werden, dass diese Klasse Objekte vom Typ `Buch` zurückliefern kann.



Hauptroutine

In der Hauptroutine werden zunächst ein `Autor`-Objekt und zwei `Buch`-Objekte erzeugt. Das Attribut `Autor` in der Klasse `Buch` wird dabei mit Absicht nicht verwendet, damit es zunächst keine zirkuläre Referenz gibt. Die Unteroutine `autorinfo()` gibt Informationen über einen Autor und seine Bücher aus. Das Objektmodell wird zunächst in eine Binärdatei, danach in eine XML-Datei serialisiert.

Ablauf des Beispiels

Im zweiten Teil wird die zirkuläre Referenz hinzugefügt: Dadurch kommt es im XML-Serialisierer zu einer Fehlermeldung, die abgefangen und ausgegeben wird.

```
' --- Anlegen einer zirkulären Referenz
b1.Autor = a
b2.autor = a
```

Danach funktioniert das binäre Serialisieren weiterhin, der XML-Serialisierer bricht jedoch mit einem Fehler ab.

```
' =====
' Beispiel aus FCL-Buch.sln
' Beispiele für binäres und XML-Serialisieren
' (C) Holger@Schwichtenberg.de
' =====
Sub OModell_Serialisieren()

    ' --- Objektmodell aufbauen
    Dim a As New autor()
    a.name = "Holger Schwichtenberg"
    a.geb = #8/1/1972#

    Dim b1 As New Buch()
    b1.titel = "Windows Scripting"
    b1.Set_auflage(5000)
    a.buecher.Add(b1)

    Dim b2 = New Buch()
    b2.titel = "COM-Komponenten-Handbuch"
    b2.Set_auflage(4000)
    a.buecher.Add(b2)

    ' --- Objektmodell ausgeben
```

```

    autorinfo(a)

    Dim o As Object ' Objekt für Deserialisierung

    ' --- Serialisierungsziele
    Const DATEI = "h:\data\buch.bin"
    Const DATEIXML = "h:\data\buch.xml"

    ' --- binäres Serialisieren
    out("Binäre Serialisierung...")
    bin_serialize(a, DATEI)
    o = bin_deserialize(DATEI)
    autorinfo(o)

    ' --- XML-Serialisieren
    out("XML Serialisierung...")
    xml_serialize(a, DATEIXML)
    o = xml_deserialize(GetType(autor), DATEIXML)
    autorinfo(o)

    ' --- Anlegen einer zirkulären Referenz
    b1.Autor = a
    b2.autor = a

    ' --- binäres Serialisieren
    out("Binäre Serialisierung...")
    bin_serialize(a, DATEI)
    a = bin_deserialize(DATEI)
    autorinfo(a)

    ' --- XML-Serialisieren
    out("XML Serialisierung...")
    Try
        xml_serialize(a, DATEIXML)
        a = xml_deserialize(GetType(autor), DATEIXML)
        autorinfo(a)
    Catch e As Exception
        out("Serialisierung nicht möglich: " & _
            e.ToString)
    End Try
End Sub

' =====
' Beispiel aus FCL-Buch.sln
' Ausgabe des Objektmodells
' (C) Holger@Schwichtenberg.de
' =====
Sub autorinfo(ByVal a As autor)
    Dim b As Buch
    out("Autor: " & a.name & ", " & _
        a.buecher.Count & " Bücher")

```

```

For Each b In a.buecher
    out("Buch: " & b.titel & ": " & _
        b.Get_auflage & " Exemplare")
    If Not b.Autor Is Nothing Then
        out("von " & b.Autor.name)
    End If
Next
End Sub

```

Listing 11.17: Beispiel zur Serialisierung/Deserialisierung einer Objekthierarchie sowohl mit dem binären als auch dem XML-Serialisierer

Ausgabe

Die Screenshots in Abbildung 11.13 und Abbildung 11.16 zeigen **Ergebnis** die erzeugten Dateien.

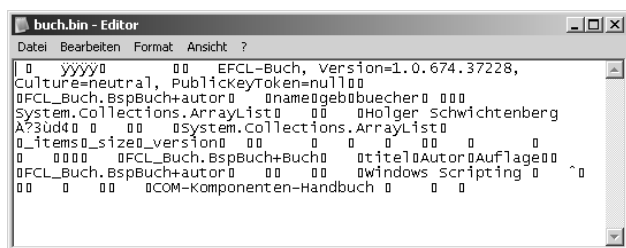


Abbildung 11.7: Ergebnis der binären Serialisierung

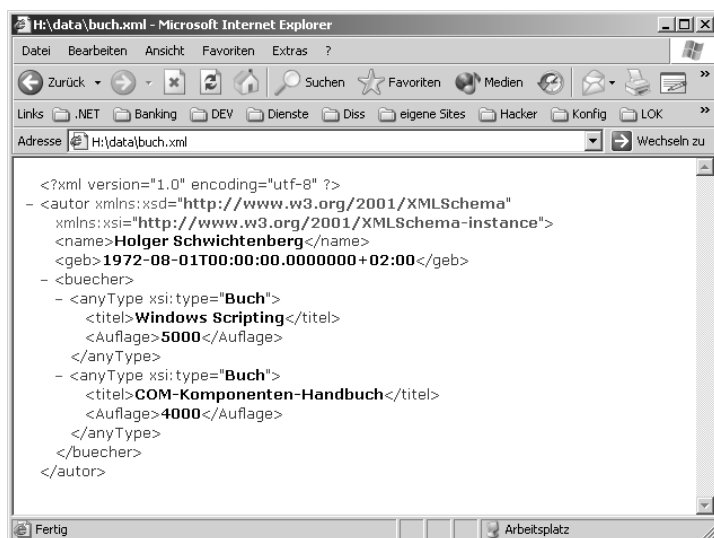


Abbildung 11.8: Ergebnis der XML-Serialisierung

XML- vs. SOAP-Serialisierung

Während der XML-Serialisierer das Objekt in einer Baumstruktur wiedergibt (siehe obige Grafik), arbeitet der SOAP-Serialisierer mit eindeutigen Objektnummern (`id="ref-x"`) und Verweisen anhand dieser Objektnummern (`href="#ref-x"`). Damit erreicht der SOAP-Serialisierer sicher, dass jedes Objekt nur einmal gespeichert werden muss, auch wenn es n-fach referenziert wird. Der XML-Serialisierer dagegen baut das Objekt bei jeder Referenzierung erneut in den XML-Dokumentenbaum ein; eine zirkuläre Referenz würde zu einem Endlos-Dokument führen.

Somit erklärt sich auch, warum der SOAP-Serialisierer zirkuläre Referenzen umwandeln kann, der XML-Serialisierer aber nicht.



Doppel-speicherung

Wenn ein einzelnes Objekt in einem Objektbaum mehrfach vorkommt, wird es durch den XML-Serialisierer mehrfach serialisiert. Bei der Deserialisierung entstehen dann zwei voneinander unabhängige Objekte mit gleichem Inhalt.

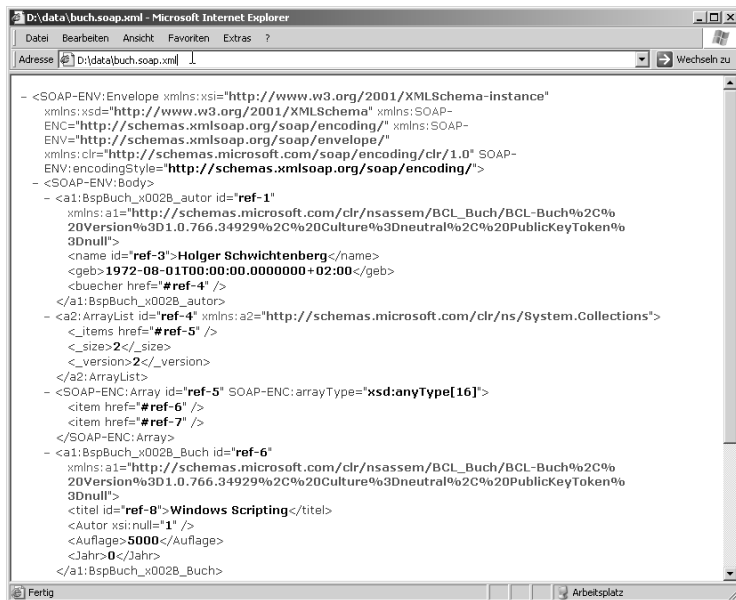


Abbildung 11.9: Ergebnis der SOAP-Serialisierung

SOAP	XML
<pre> <b1 href="#ref-5" /> <b2 href="#ref-5" /> <a1:BspBuch_x002B_Buch id="ref-5" xmlns:a1="..."> <titel id="ref-7">Windows Scripting</titel> <Autor xsi:null="1" /> <Auflage>5000</Auflage> <Jahr>0</Jahr> <a2 xsi:null="1" /> </a1:BspBuch_x002B_Buch> </pre>	<pre> <b1> <titel>Windows Scripting</titel> <Auflage>5000</Auflage> <Jahr>0</Jahr> </b1> <b2> <titel>Windows Scripting</titel> <Auflage>5000</Auflage> <Jahr>0</Jahr> </b2> </pre>

Tabelle 11.2: Doppelspeicherung in XML vs. Referenzen in SOAP

11.6 Änderungen an serialisierten Objekten

Eine interessante Frage ist, was passiert, wenn sich *nach* dem Serialisieren die Klasse, zu der das Objekt gehört, ändert. Grundsätzlich ist der XML-Serialisierer am unempfindlichsten. Die folgende Tabelle gibt Detailinfos.

Änderung	Binär-Serialisierer	SOAP-Serialisierer	XML-Serialisierer
Umbenennen eines Attributs	Fehler	Fehler	Kein Fehler, aber Attribut wird ignoriert
Hinzufügen eines Attributs	Fehler	Fehler	OK
Entfernen eines Attributs	Fehler	Fehler	OK
Ändern des Datentyps Byte → Long	OK	OK	OK
Ändern des Datentyps Byte → String	Fehler	OK	OK
Ändern des Datentyps Long → Byte	Fehler	Nur OK, wenn neuer Typ den serialisierten Wert aufnehmen kann	

Tabelle 11.3: Verhalten der Serialisierer beim Deserialisieren eines Objekts, dessen Klasse sich geändert hat

Eine Lösung dieser Änderungskonflikte liegt in der Verwendung der benutzerdefinierten Deserialisierung über die Schnittstelle `ISerializable`.

11.7 Benutzerdefinierte (De-)Serialisierung

Steuerung der Serialisierung

Der Entwickler hat die Möglichkeit, selbst zu steuern, welche Attribute wie serialisiert werden. Die Steuerung ist unterschiedlich in den Serialisierern in *System.Runtime.Serialization.Formatters* und dem XML-Serialisierer in *System.Xml.Serialization*.

11.7.1 Benutzerdefinierte Serialisierung für BinaryFormatter und SOAPFormatter

Der binäre Serialisierer und der SOAP-Serialisierer bieten drei unterschiedliche Konzepte zur Steuerung der Serialisierung:

- ▶ Mit dem Meta-Attribut `<System.NonSerialized(>` kann ein Attribut von der Serialisierung ausgenommen werden.
- ▶ Über die Schnittstelle *ISerializable* kann die Form der Serialisierung einzelner Attribute gesteuert werden.
- ▶ Über die Schnittstelle *ISerializable* kann die Klasse nach Abschluss der Deserialisierung aufgerufen werden.

ISerializable

ISerializable

Zur Steuerung der Serialisierung beim binären Serialisieren und beim Serialisieren via SOAP muss die zu serialisierende Klasse die Schnittstelle *ISerializable* anbieten. Darin sind zu implementieren:

- ▶ Die Methode `GetObjectData()` mit nachfolgender Signatur:

```
Public Sub GetObjectData( _
    ByVal info As SerializationInfo, _
    ByVal context As StreamingContext) _
    Implements ISerializable.GetObjectData
```

Das *SerializationInfo*-Objekt nimmt dabei die zu serialisierenden Daten als Attribut-Wert-Paare auf.

- ▶ Ein Konstruktor, der als Parameter ein *SerializationInfo*-Objekt und ein *StreamingContext*-Objekt erwartet. Dieser Konstruktor wird bei der Deserialisierung aufgerufen.

```
Sub New(ByVal info As SerializationInfo, _
    ByVal context As StreamingContext)
```

Das *SerializationInfo*-Objekt übergibt dabei die serialisierten Daten als Attribut-Wert-Paare.

- Wenn nicht sowieso schon vorhanden, muss ein parameterloser Konstruktor implementiert werden.

```
Public Sub New()
```

Wenn diese Methoden vorhanden sind, dann ruft der Serialisierer während des Serialisierens automatisch die Methode `GetObjectData()` auf und während des Deserialisierens den spezifischen Konstruktor.



IDeserializationCallback

Eine zu serialisierende Klasse kann auch `IDeserializationCallback` implementieren. Diese Schnittstelle umfasst nur eine Methode, `OnDeserialization(ByVal s As Object)`, die aufgerufen wird, wenn die Deserialisierung beendet ist. Diese Methode zu füllen macht Sinn, um Daten des Objekts zu vervollständigen, die nicht serialisiert wurden bzw. aus serialisierten Daten errechnet werden.

IDeserializationCallback

Beispiel

Das Beispiel zeigt die Klasse `Person`, die so implementiert ist, dass nur die Attribute `Vorname`, `Name` und `Geb` serialisiert werden. Die beiden Namen werden dabei in einem Feld serialisiert und bei der Deserialisierung wieder getrennt (dieses Vorgehen ist natürlich nicht zwingend notwendig – es dient hier nur zu Demonstrationszwecken). In dem Attribut `Deserialisiert` werden Datum und Uhrzeit am Ende der Deserialisierung erfasst.

Serialisierung der Klasse »Person«

```
' =====
' Beispiel aus FCL-Buch.sln
' Klasse für benutzerdefinierte Serialisierung
' (C) Holger@Schwichtenberg.de
' =====

Public Module Serialisieren_Binaer_mit_Kontrolle

    <Serializable(> Public Class Person2
        Implements ISerializable, _
            IDeserializationCallback

        Public Name As String
        Public Vorname As String
        Public Geb As Date
        Public Geschlecht As Char
        Public Kinderanzahl As Byte
        Public Taetigkeiten As String()
        Public Deserialisiert As Date
```

```

Public Sub New()
    out("Konstruktoraufruf!")
End Sub

' --- Serialisierung ausgewählter Eigenschaften
Public Sub GetObjectData( _
    ByVal info As SerializationInfo, _
    ByVal context As StreamingContext) _
    Implements ISerializable.GetObjectData
    out("Serialisierung des Typs: " & _
        info.FullTypeName.ToString()
        info.AddValue("kompletterName", Vorname & _
            " " & Name)
        info.AddValue("Geb", Geb)
End Sub

' Deserialisierung ausgewählter Eigenschaften
Public Sub New( _
    ByVal info As SerializationInfo, _
    ByVal context As StreamingContext)
    Dim namen As String()
    namen = CType(info.GetValue( _
        "kompletterName", GetType(String)), _
        String).Split
    Vorname = namen(0)
    Name = namen(1)
    Geb = CDate(info.GetValue("Geb", _
        GetType(Date)))
End Sub

' wird am Ende der Deserialisierung aufgerufen
Public Sub OnDeserialization( _
    ByVal s As Object) Implements _
    DeserializationCallback.OnDeserialization
    Deserialisiert = Now
    out("Deserialisierung komplett!")
End Sub

End Class

' =====
' Beispiel aus FCL-Buch.sln
' Benutzerdefinierte Serialisierung
' (C) Holger@Schwichtenberg.de
' =====

Sub serial_kontrolle_1()
Const DATEI = "h:\Data\person.bin"

' --- Objekt füllen
Dim p As New Person2()
p.Vorname = "Holger"

```

```

p.Name = "Schwichtenberg"
p.Geb = #8/1/1972#
p.Geschlecht = "m"
p.Kinderanzahl = 0
p.Taetigkeiten = New String() {"Berater", _
    "Entwickler", "Journalist", "Dozent"}

Dim form As New BinaryFormatter()
Dim stream As FileStream

' --- Serialisieren
stream = New FileStream(DATEI, _
    FileMode.Create, FileAccess.Write, _
    FileShare.None)
form.Serialize(stream, p)
stream.Close()
out("Objekt wurde gespeichert!")
p = Nothing

' --- Deserialisieren
stream = New FileStream(DATEI, _
    FileMode.Open, FileAccess.Read, _
    FileShare.Read)
p = form.Deserialize(stream)
stream.Close()
out("Objekt wurde geladen!")
out(p.Vorname & " " & p.Name & " wurde am " _
    & p.Geb & " geboren!")

End Sub
End Module

```

Listing 11.18: Benutzerdefinierte binäre Serialisierung

Ausgabe

Dieses Beispiel gibt folgende Meldungen aus:

- 1 Konstruktoraufruf!
- 2 Serialisierung des Typs:
FCL_Buch.Serialisieren_Binaer_mit_Kontrolle+Person2
- 3 Objekt wurde gespeichert!
- 4 Deserialisierung komplett!
- 5 Objekt wurde geladen!
- 6 Holger Schwichtenberg wurde am 01.08.1972 geboren!

11.7.2 Benutzerdefinierte Serialisierung für den XmlSerializer

Attribute zur Steuerung der Serialisierung

Die Steuerung der Serialisierung beim XML-Serialisierer erfolgt durch Meta-Attribute. Es gibt zahlreiche Meta-Attribute für den Zweck, das folgende Beispiel demonstriert zwei wichtige:

► `<XmlAttributeAttribute()>`

Dieses Meta-Attribut bewirkt, dass das damit versehene Klassenattribut in ein XML-Attribut und nicht – wie es Standard ist – in ein XML-Element serialisiert wird.

► `<XmlIgnore()>`

Dieses Meta-Attribut bewirkt, dass das damit versehene Klassenattribut nicht serialisiert wird.



Das Attribut `<System.NonSerialized()>`, das zur Ausklammerung von Attributen beim Binär- und SOAP-Serialisierer verwendet wird, wirkt nicht auf den XML-Serialisierer.

Beispiel

Das folgende Beispiel zeigt beide o.g. Meta-Attribute in der Anwendung.

```
' =====
' Beispiel aus FCL-Buch.sln
' Klasse für benutzerdefinierte XML-Serialisierung
' (C) Holger@Schwichtenberg.de
' =====
<Serializable()> Public Class Person3
    Public Name As String
    Public Vorname As String
    <XmlAttributeAttribute()> _
    Public Geschlecht As Char
    Public Geb As Date
    <XmlIgnore()> Public Kinderanzahl As Byte
End Class

Sub serial_kontrolle_2()
    Dim p As New Person3()
    p.Vorname = "Holger"
    p.Name = "Schwichtenberg"
    p.Geb = #8/1/1972#
    p.Geschlecht = "m"
    p.Kinderanzahl = 0
```

```
xml_serialize(p, "h:\data\person3.xml")  
End Sub
```

Listing 11.19: Beispiel zur Steuerung der XML-Serialisierung mit Meta-Attributen

Ausgabe

Die obige Routine erzeugt die nachfolgend abgebildete XML-Datei. **Ergebnis**

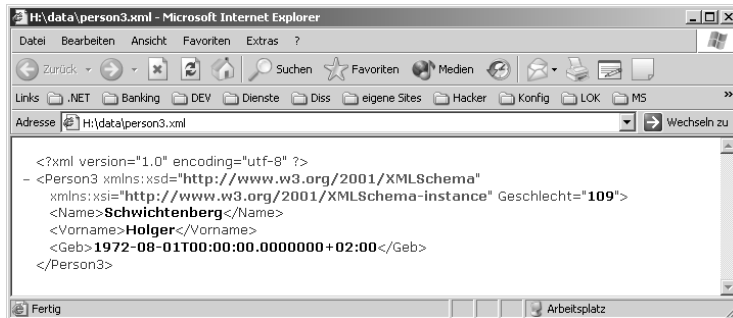


Abbildung 11.10: Erzeugte XML-Datei

12 Namespace System.ServiceProcess

Im Namespace `System.ServiceProcess` finden Sie alle Klassen, die im Zusammenhang mit Windows-Diensten (oder Windows-Services) stehen. Diese Programme, die ihre Arbeit unsichtbar im Hintergrund verrichten, konnten mit bisherigen Mitteln nur mit verhältnismäßig viel Aufwand programmiert werden. Unter Visual Basic 6 war die Programmierung eines Dienstes mit Hausmitteln gar nicht und mit zusätzlichen Tools nur sehr unsauber möglich.

Das .NET-Framework bietet eine umfangreiche Unterstützung für Windows-Dienste. Was bisher für VB-Programmierer nahezu unmöglich war, ist nun ein recht einfacher Vorgang.

Windows-Dienste benötigen einen NT-Kernel. Unter Windows 98 oder ME laufen diese Programme also auch dann nicht, wenn auf dem Zielsystem das .NET-Framework installiert ist. Windows NT4 mit Service Pack 6, Windows 2000 mit Service Pack 2 oder Windows XP werden zur Ausführung benötigt.



12.1 Architektur von Windows-Diensten

Dienste werden verwaltet vom *Service Control Manager* (auf Deutsch »Dienstkontrollmanager«). Im Folgenden wird die gebräuchliche Abkürzung *SCM* verwendet werden. Der SCM kommuniziert mit den im System installierten Diensten und sendet diesen Signale, die der jeweilige Dienst auswertet, woraufhin er eine entsprechende Aktion durchführt. Diese Aktionen – Starten, Stoppen, Anhalten oder Fortsetzen – werden innerhalb des eigentlichen Dienstprogramms über festgelegte Methoden bearbeitet.

**Service Control
Manager**

**Rechte von
Windows-
Diensten**

Dienste laufen auch dann, wenn kein Benutzer im System eingeloggt ist. Sogar das Login selbst ist bereits ein Windows-Dienst, der in diesem Fall Eingaben vom Benutzer entgegennimmt, kontrolliert und dann die entsprechende Umgebung und die Einstellungen lädt. Dieses Verhalten erzwingt, dass ein Dienst nicht zwangsläufig einem Benutzer zugewiesen sein muss, aber es ist möglich. Es geht dabei um die Rechteverwaltung, d.h. ein solcher Dienst kann unterschiedlich hohe Rechte besitzen, mit denen er auf das System zugreifen kann. Sie können einen Dienst sowohl mit Rechten versehen, die denen eines im System registrierten Benutzers entsprechen, als auch mit Rechten, die sogar weitreichender sind als die des System-Administrators. Ein Dienst kann so konfiguriert werden, dass er auf das gesamte System ohne irgendwelche Einschränkungen zugreifen kann. Der dafür verwendete Benutzer-Account ist der so genannte *System*-Benutzer.

**Bestandteile von
Diensten**

Dienste müssen tatsächlich installiert und in der Registry eingetragen werden. Obwohl es in der Zukunft mit Hilfe von .NET möglich sein wird, Programme durch simples Kopieren weiterzugeben, gilt dies nicht für Dienste. Der SCM muss ja wissen, wo er sie findet und – was noch viel wichtiger ist – mit welcher Methode der Dienst gestartet werden kann. Diese Informationen bekommt der SCM von der Registry bzw. vom Dienst selbst. Ein Dienstprogramm besteht damit aus den folgenden Bestandteilen:

- ▶ Einem Hauptprogramm zur Registrierung des Dienstes beim SCM
- ▶ Einem Dienst-Hauptprogramm zum Starten eines Dienstes
- ▶ Ereignisbehandlungsroutinen zum Stoppen, Anhalten oder Fortsetzen eines Dienstes
- ▶ Einem Installer zum Eintragen des Dienstes in die Registry

Die Aufteilung in ein Hauptprogramm zum Registrieren des Dienstes und eines zum Starten eines Dienstes hat durchaus Sinn. Sie haben so die Möglichkeit, in einer ausführbaren Datei mehrere Dienste unterzubringen, die über die Hauptmethode registriert und dann einzeln gestartet werden können. Dabei ist es nicht notwendig, alle Dienste zu starten – es besteht auch die Möglichkeit, einige Dienste automatisch zu starten, andere wiederum manuell.

Schematisch dargestellt (allerdings hier stark vereinfacht) sehen Sie die Vorgehensweise beim Starten/Stoppen eines Dienstes in Abbildung 12.1. Wie gesagt, stark vereinfacht. Der Grund hierfür ist, dass uns an dieser Stelle das Warum und Wie, das hinter den Kulissen stattfindet, nicht so sehr interessieren soll.

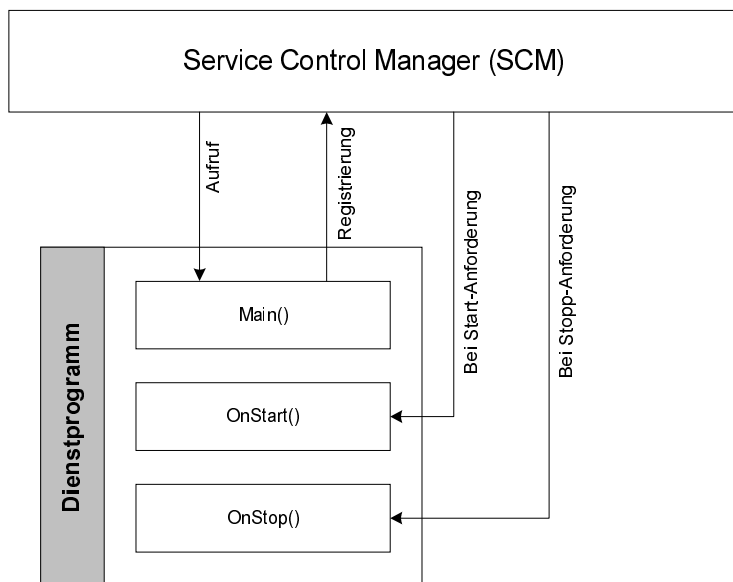


Abbildung 12.1: Kommunikation des SCMs mit dem Dienst (vereinfacht)

12.2 Die Klasse ServiceBase

Die Basisklasse für den eigentlichen Dienst ist die Klasse `ServiceBase` aus `System.ServiceProcess`. Von dieser Klasse müssen wir eine eigene Klasse für unseren Dienst ableiten. Damit das Ganze ein wenig einfacher wird, stellt uns Visual Studio den Projekttyp `WINDOWS SERVICE` zur Verfügung, der automatisch eine eigene Klasse erzeugt und diese von `ServiceBase` ableitet.

ServiceBase

12.2.1 Eigenschaften und Methoden

Die neue Klasse besitzt einige Eigenschaften, die recht leicht verständlich sind. Sie finden sie in Tabelle 12.1.

Eigenschaft	Bedeutung
AutoLog	Gibt an, ob die Ereignisse diesen Dienst betreffend automatisch im Ereignisprotokoll eingetragen werden. Standardmäßig ist diese Eigenschaft <code>True</code> .
CanHandlePowerEvent	Gibt an, ob der Dienst auf Änderungen des Systemstatus reagieren kann, z.B. wenn der Computer in den Suspend-Modus geschaltet wird oder (bei Notebooks) die Batterie leer wird
CanPauseAndContinue	Gibt an, ob der Dienst angehalten und danach wieder fortgesetzt werden kann. Die entsprechenden Ereignisse werden dann aktiviert, der SCM ermöglicht im Folgenden das Anhalten und Fortsetzen des Dienstes.
CanShutdown	Gibt an, ob der Dienst eine Nachricht erhält, wenn das System heruntergefahren wird
CanStop	Gibt an, ob der Dienst gestoppt (also beendet) werden kann. Standardmäßig ist diese Eigenschaft auf <code>True</code> gesetzt.
EventLog	Ruft ein Ereignisprotokoll ab. Wenn <code>AutoLog</code> <code>True</code> ist, werden Ereignisse wie das Starten oder Stoppen des Dienstes direkt in das Ereignisprotokoll geschrieben. Über die Eigenschaft <code>EventLog</code> können Sie auf das Ereignisprotokoll zugreifen und weitere Ereignisse eintragen.
ServiceName	Der Name, mit dem der Dienst beim System angemeldet wird

Tabelle 12.1: Die Eigenschaften von `ServiceBase`

Abbildung 12.2 zeigt das Objektmodell der Klasse `ServiceBase`.

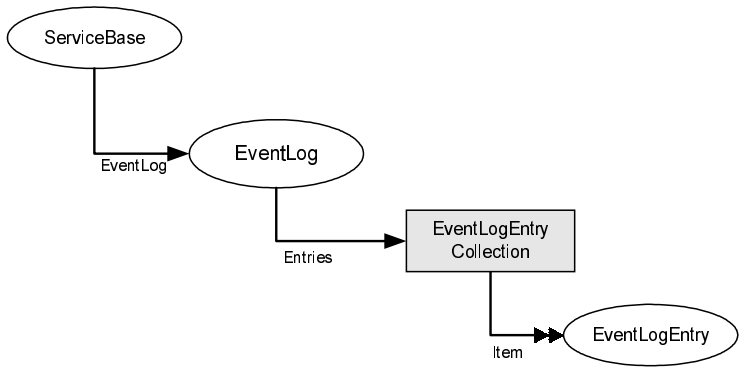


Abbildung 12.2: Das Objektmodell von `ServiceBase`

Die entsprechenden korrespondierenden Methoden, die der SCM im Falle diverser Aktionen aufruft, finden Sie in Tabelle 12.2.

Methode	Bedeutung
OnContinue()	Wird vom SCM aufgerufen, wenn ein Dienst, der angehalten wurde, fortgesetzt wird. <code>CanPauseAndContinue</code> muss <code>True</code> sein.
OnCustomCommand()	Ermöglicht es dem Dienst, auf ein benutzerdefiniertes Kommando zu reagieren. Dieses Kommando kommt ebenfalls über den SCM zum Dienst, es wird jedoch eine von <code>ServiceController</code> abgeleitete Komponente benötigt, deren <code>ExecuteCommand()</code> -Methode dem Aufruf von <code>OnCustomCommand</code> dient.
OnPause()	Wird vom SCM aufgerufen, wenn der Dienst angehalten (nicht gestoppt) wird. <code>CanPauseAndContinue</code> muss <code>True</code> sein.
OnPowerEvent()	Wird vom SCM aufgerufen, wenn eine Änderung des Powerstatus auftritt. <code>CanHandlePowerEvent</code> muss <code>True</code> sein.
OnShutdown()	Wird vom SCM aufgerufen, wenn das System heruntergefahren wird. <code>CanShutdown</code> muss <code>True</code> sein.
OnStart()	Bezeichnet die Startmethode des Dienstes. Wird aufgerufen, wenn der Dienst gestartet wird. Diese Methode ist die Dienst-Hauptmethode.
OnStop()	Wird vom SCM zum Stoppen des Dienstes (also zum Beenden) aufgerufen

Tabelle 12.2: Ereignisbehandlungsroutinen für den Service

12.2.2 Die Dienst-Hauptmethode

Eine wichtige Methode ist hier nicht aufgeführt, einfach aus dem Grund, weil es sich bei allen diesen Methoden um Instanzmethoden der Klasse handelt. Es geht um die Hauptmethode, die in unserer Datei enthaltenen Dienste beim SCM registriert. Diese Methode ist als `Shared` deklariert (`static` in C#) und heißt `Run()`. Übergabeparameter ist entweder ein einzelner Dienst oder ein Array aus Diensten, die beim SCM registriert werden. Da es sich bei einem Dienst auch nur um ein Programm handelt und dessen Einsprungspunkt wieder eine Methode `Main()` ist, wird darin auch die Methode `Run()` aufgerufen. Eine typische `Main()`-Methode mit einem Dienst, der registriert werden soll, sieht damit folgendermaßen aus:

Run()

```
<MTAThread(>> _
Shared Sub Main()
    ServiceBase.Run( Service1 )
End Sub
```

Das Attribut `MTAThread` bestimmt das Threading-Modell für diese Art von Anwendung. `MTAThread` legt *MultiThreadedApartment* als Standard fest.

Der Methode `Run()` muss lediglich der Dienst übergeben werden (in diesem Fall `Service1`), da es sich ja um einen einzelnen Dienst handelt. Bei mehreren Diensten müssen Sie zunächst ein Array aus diesen Diensten erzeugen und dieses dann der Methode `Run` übergeben.

```
<MTAThread(>> _
Shared Sub Main()
    Dim ServicesToRun() As ServiceBase
    ServicesToRun = New ServiceBase() { New Service1(), _
        New Service2(), New Service3() }
    ServiceBase.Run(ServicesToRun)
End Sub
```



Die Entwicklungsumgebung fügt vor den Klassennamen natürlich auch noch den entsprechenden Namespace ein. Die Originalzeile sieht daher so aus:

```
Dim ServicesToRun() As _
    System.ServiceProcess.ServiceBase
```

Wenn dieser Namespace über `Imports` eingebunden wird, wird seine Angabe natürlich nicht mehr benötigt. Daher kann in diesem Beispielcode darauf verzichtet werden, was aus Gründen der Zeilenlänge und der Übersichtlichkeit geschah. Daraus ergibt sich dann die Zeile

```
Dim ServicesToRun() As ServiceBase
```

12.3 Die Klassen `ServiceProcessInstaller` und `ServiceInstaller`

Dienst installieren

Bevor Sie mit der Programmierung des Dienstes anfangen können, müssen Sie noch eine Möglichkeit finden, den Dienst zu installieren, d.h. die ihn betreffenden Daten in die Registry zu schreiben. Dazu benötigen Sie die Daten der Dienst-Applikation selbst und dann einen Installationsvorgang für jeden Dienst, der in der Applikation festgelegt ist.

Auch hier nimmt Ihnen Visual Studio .NET die Hauptarbeit ab. Wenn Sie mit der rechten Maustaste auf die Entwurfsansicht des Dienstprogramms klicken, finden Sie einen Menüpunkt **INSTALLER HINZUFÜGEN**, der die meiste Arbeit – nämlich das Erstellen des Codes – übernimmt. Für die Applikation wird eine neue Klasse, abgeleitet von `ServiceProcessInstaller`, generiert, für jeden enthaltenen Dienst eine Klasse `ServiceInstaller`. Alles, was für uns zu tun bleibt, ist, die Eigenschaften anzupassen. Sie finden die Eigenschaften von `ServiceProcessInstaller` und `ServiceInstaller` in Tabelle 12.3 und Tabelle 12.4.

ServiceProcess Installer und ServiceInstaller

Eigenschaft	Bedeutung
Account	Gibt an, unter welchem Account diese Dienst-Applikation laufen soll. Die möglichen Einstellungen sind <code>User</code> , <code>LocalSystem</code> , <code>LocalService</code> oder <code>NetworkService</code> .
Password	Das Passwort für den Account, der für die Dienst-Applikation verwendet wird. Muss ausgefüllt werden, wenn ein <code>User-Account</code> benutzt werden soll.
Username	Der Name des Benutzers, mit dessen Rechten der Dienst laufen soll. Wird benötigt, wenn ein <code>User-Account</code> benutzt werden soll.

Tabelle 12.3: Die Eigenschaften von `ServiceProcessInstaller`

Eigenschaft	Bedeutung
DisplayName	Gibt an, unter welchem Namen der Dienst später im SCM erscheint
ServiceName	Gibt den Namen des Dienstes an, unter dem er beim System registriert ist. Der Wert dieser Eigenschaft muss mit dem Wert <code>ServiceName</code> des Dienstes identisch sein.
ServicesDependOn	Gibt an, von welchen Diensten der aktuelle Dienst abhängig ist. Diese Dienste werden dann mitgestartet, wenn der aktuelle Dienst gestartet wird.
StartType	Gibt an, auf welche Art dieser Dienst gestartet werden soll (manuell, automatisch oder überhaupt nicht)

Tabelle 12.4: Die Eigenschaften von `ServiceInstaller`

Normalerweise gehen Sie so vor, dass Sie lediglich die Eigenschaften anpassen und mit den Methoden, die diese beiden Klassen ebenfalls bereitstellen, nicht arbeiten. Diese werden automatisch vom Installer aufgerufen, wenn die eigentliche Installation des

InstallUtil.exe

Dienstes durchgeführt wird. Das erledigt ein Hilfsprogramm namens *InstallUtil.exe*. Da wir mit den Methoden also nicht arbeiten werden, sind sie hier auch nicht aufgeführt.

12.3.1 Beispiel: Ein einfacher Dienst

Als Beispiel soll ein recht einfacher Dienst programmiert werden, der lediglich irgendwelche Daten in eine Datei schreiben soll. Da es sich um einen einfachen Dienst handeln soll, werden schlicht Zeiten protokolliert, d.h. jede Minute wird die Zeit in eine Datei geschrieben.



Bei diesem Dienst handelt es sich um eine recht simple Funktionalität. Dennoch wurde der Code vor dem Erstellen des Dienstes in einem Windows-Forms-Projekt getestet, damit sichergestellt war, dass die Funktionalität in Ordnung ist. Vor allem bei etwas umfangreicheren Dienstprogrammen sollten Sie sich diese Vorgehensweise zu Eigen machen, da ein Dienst nach wie vor installiert werden muss. Das bedeutet dann aber auch bei jedem entdeckten Fehler eine Deinstallation, Fehlerbeseitigung und Reinstallation.

Ein Dienst-Projekt erzeugen

Zunächst wird ein neues Projekt vom Typ WINDOWS-SERVICE erstellt. Der Service erhält den Namen *SimpleService*. Den generierten Code sehen Sie in Listing 12.1, wobei einige Kommentare entfernt wurden.

```
Imports System.ServiceProcess
Imports System.Threading
Imports System.IO

Public Class SimpleService
    Inherits System.ServiceProcess.ServiceBase

    #Region " Component Designer generated code "

    Public Sub New()
        MyBase.New()

        ' Dieser Aufruf wird vom Component-Designer
        ' benötigt.
        InitializeComponent()

        ' Weitere Initialisierungsvorgänge

    End Sub
```



```

Protected Overrides Sub Dispose( _
    ByVal disposing As Boolean)

    If disposing Then
        If Not (components Is Nothing) Then
            components.Dispose()
        End If
    End If
    MyBase.Dispose(disposing)

End Sub

' Der Haupteintrittspunkt für den Dienst
<MTAThread(> _
Shared Sub Main()

    Dim ServicesToRun() As ServiceBase
    ServicesToRun = New ServiceBase() _
        {New Service1()}
    System.ServiceProcess.ServiceBase.Run(ServicesToRun)

End Sub

' Benötigt vom Component-Designer
Private components As System.ComponentModel.IContainer

<System.Diagnostics.DebuggerStepThrough(> _
Private Sub InitializeComponent()
    '
    'SimpleService
    '
    Me.AutoLog = False
    Me.ServiceName = "SimpleService"

End Sub

#End Region

Protected Overrides Sub OnStart(ByVal args() As String)

    ' Add code here To start your service.

End Sub

Protected Overrides Sub OnStop()

    ' Add code here To Stop your service.

End Sub

```

Listing 12.1: Der automatisch generierte Code (leicht angepasst)

**Dienstnamen
ändern**

Am Code fällt bei genauerer Betrachtung auf, dass trotz der Änderung des Service-Namens immer noch der Name `Service1` an die `Start`-Methode übergeben wird. Hier müssen wir manuell eingreifen und den Code abändern, statt `Service1` verwenden wir natürlich hier den Namen `SimpleService`, da unsere Dienst-Klasse ja genauso heißt. In Visual Studio müssen Sie dazu den normalerweise geschlossenen `Region`-Teil öffnen und den Namen des Dienstes in der Methode `Main()` ändern. Ansonsten fällt der Fehler spätestens beim Kompilieren auf.

Jetzt geht es zunächst darum, die Eigenschaften korrekt abzuändern. Sie brauchen keine Einträge in der Log-Datei (wie gesagt, es ist ein simpler Dienst, den wir hier programmieren wollen). Sie setzen also die Eigenschaft `AutoLog` auf `False`. Ansonsten muss nichts geändert werden.

Hinzufügen der Funktionalität

Obwohl der Dienst sehr einfach gestrickt ist, sollen in diesem Fall Threads verwendet werden. Der Grund, warum Sie das immer tun sollten, ergibt sich aus dem Verhalten des SCMs, der das Starten eines Dienstes nach einer gewissen Zeit mit einer Fehlermeldung abbricht. Sollten Sie also einen komplexeren Dienst programmiert haben, so ist es möglich, dass der Start zu lange dauert und der SCM davon ausgeht, ein Fehler sei aufgetreten.

Um dies zu vermeiden, verwenden Sie einen eigenen Thread für den Dienst. In der Start-Routine des Dienstes wird dann nur noch der Thread gestartet, der die eigentliche Arbeit danach übernimmt. Dadurch kommen Sie beim Starten des Dienstes nie in Zeitbedrängnis.

Funktionalität

Bevor wir jedoch unseren Thread programmieren, wenden Sie sich zunächst der eigentlichen Funktionalität des Dienstes zu. Zum Schreiben in die Datei verwenden Sie eine eigene Unterroutine und da diese die Arbeit erledigt, habe ich sie im Beispiel `WorkerSub` genannt. Sie sehen den Code, der eigentlich nur jede Minute die Zeit in eine Datei schreibt, in Listing 12.2.

```
Private Sub WorkerSub()  
  
    'Schreibt jede Minute in eine Datei  
    Dim actMinute As Integer  
    actMinute = DateTime.Now.Minute  
    Dim sr As StreamWriter
```

```

While True

    If actMinute <> DateTime.Now.Minute Then
        sr = New StreamWriter( _
            "C:\simpleservice.txt", True)
        sr.WriteLine("Zeit: " + _
            DateTime.Now.ToLongTimeString())
        sr.Flush()
        sr.Close()
        actMinute = DateTime.Now.Minute
    End If

End While

End Sub

```

Listing 12.2: Die Arbeitermethode WorkerSub

Die Methode schreibt minütlich die aktuelle Zeit in die Datei *simpleservice.txt* auf Laufwerk C.

Jetzt soll der angesprochene Thread programmiert werden. Zunächst wird ein neues `private`s Feld für den Thread deklariert, da ja aus mehreren Methoden darauf zugegriffen werden muss – in diesem Fall aus den Methoden zum Starten und Stoppen heraus. Die Deklaration ist eigentlich recht einfach:

```
Private simpleThread As Thread
```

Der Thread wird in der Methode, in der der Dienst gestartet wird, erzeugt und abgebrochen, wenn die Anforderung zum Stoppen des Dienstes kommt. Die beiden bereits vordefinierten Methoden `OnStart()` und `OnStop()` sehen Sie mit der Funktionalität in Listing 12.3.

```

Protected Overrides Sub OnStart( _
    ByVal args() As String)

    simpleThread = New Thread( _
        New ThreadStart(AddressOf WorkerSub))
    simpleThread.Start()

End Sub

Protected Overrides Sub OnStop()

    simpleThread.Abort()

End Sub

```

Listing 12.3: Die Methoden OnStart und OnStop

Das war's eigentlich schon – der Dienst ist fertig. Jetzt muss er allerdings noch installiert werden.

Installation

Der Installer für den Dienst

Visual Studio .NET erledigt auch hier die Hauptarbeit. Sie fügen einen Installer hinzu, indem Sie mit der rechten Maustaste auf die Entwurfsansicht des Dienstes klicken und **INSTALLER HINZUFÜGEN** anwählen. Sie können auch einfach zur Entwurfsansicht wechseln und sehen dann schon im Eigenschaftsfenster die Möglichkeit, den Installer hinzuzufügen (ganz unten).

Visual Studio .NET erstellt alles, was benötigt wird, es müssen lediglich noch einige Eigenschaften angepasst werden. Der Dienst soll als **LocalSystem-Dienst** ausgeführt werden, die Eigenschaft **StartType** bleibt zum Testen der Funktionalität auf **Manual** stehen. Nach dem Einstellen der Eigenschaften enthält die Installer-Routine den Code aus Listing 12.4.

```
Imports System.ComponentModel
Imports System.Configuration.Install

<RunInstaller(True)> Public Class ProjectInstaller
    Inherits System.Configuration.Install.Installer

    #Region " Component Designer generated code "

    Public Sub New()
        MyBase.New()

        InitializeComponent()

    End Sub

    Protected Overloads Overrides Sub Dispose( _
        ByVal disposing As Boolean)

        If disposing Then
            If Not (components Is Nothing) Then
                components.Dispose()
            End If
        End If
        MyBase.Dispose(disposing)

    End Sub

    ' Benötigt vom Component-Designer
    Private components As _
```

```

System.ComponentModel.IContainer

Friend WithEvents ServiceProcessInstaller1 As _
    System.ServiceProcess.ServiceProcessInstaller
Friend WithEvents ServiceInstaller1 As _
    System.ServiceProcess.ServiceInstaller

<System.Diagnostics.DebuggerStepThrough(> _
    Private Sub InitializeComponent()

        Me.ServiceProcessInstaller1 = _
            New System.ServiceProcess.ServiceProcessInstaller()
        Me.ServiceInstaller1 = _
            New System.ServiceProcess.ServiceInstaller()
        'ServiceProcessInstaller1
        '
        Me.ServiceProcessInstaller1.Account = _
            System.ServiceProcess.ServiceAccount.LocalSystem
        Me.ServiceProcessInstaller1.Password = Nothing
        Me.ServiceProcessInstaller1.Username = Nothing
        'ServiceInstaller1
        '
        Me.ServiceInstaller1.DisplayName = "SimpleService"
        Me.ServiceInstaller1.ServiceName = "SimpleService"
        '
        'ProjectInstaller
        '
        Me.Installers.AddRange( _
            New System.Configuration.Install.Installer() _
            {Me.ServiceProcessInstaller1, _
            Me.ServiceInstaller1})

    End Sub

#End Region

End Class

```

Listing 12.4: Die Installationsroutine des Dienstes (angepasst)

Der Code wurde um einige Kommentare bereinigt und etwas an die Buchbreite angepasst.

Der gerade erzeugte Installer erledigt die gesamte Installationsarbeit. Die Registry-Einträge werden erzeugt und der Dienst damit beim System angemeldet. Allerdings wird noch eine Installationsroutine benötigt, denn mit reinem Kopieren ist es nicht getan und die Funktionalität des Installers befindet sich ja innerhalb des Dienstes. Zum Installieren wird also noch ein Setup-Projekt benötigt.

12.3.2 Das Setup-Projekt für den Dienst

Alle Installer-Klassen sind abgeleitet von der Basisklasse `Installer` aus dem Namespace `System.Configuration.Install`. Diese liefert einige Methoden, die für die Installation benötigt werden, nämlich

- ▶ `Install()` zum Installieren,
- ▶ `Uninstall()` zum Deinstallieren,
- ▶ `Commit()`, um die Änderungen an der Konfiguration speichern zu können,
- ▶ `Rollback()`, um die Änderungen an der Konfiguration zurücknehmen zu können, falls etwas schief läuft.

Die verwendeten Klassen `ServiceInstaller` und `ServiceProcessInstaller` stammen auch von `Installer` ab, dazwischen liegt allerdings noch die Klasse `ComponentInstaller`. Das Setup-Projekt muss aber in jedem Fall in der Lage sein, diese Methoden aufrufen zu können. Sie werden gleich sehen, wie das erreicht werden kann.

Ein neues Setup-Projekt hinzufügen

Zunächst wird der aktuellen Projektmappe ein neues Setup-Projekt hinzugefügt. Im Falle dieses Beispiels heißt es *MySetup*. Sie können wahlweise den Assistenten verwenden oder das Projekt selbst erstellen und die benötigten Dateien hinzufügen. Wo Sie die Setup-Projekte finden, sehen Sie in Abbildung 12.3.

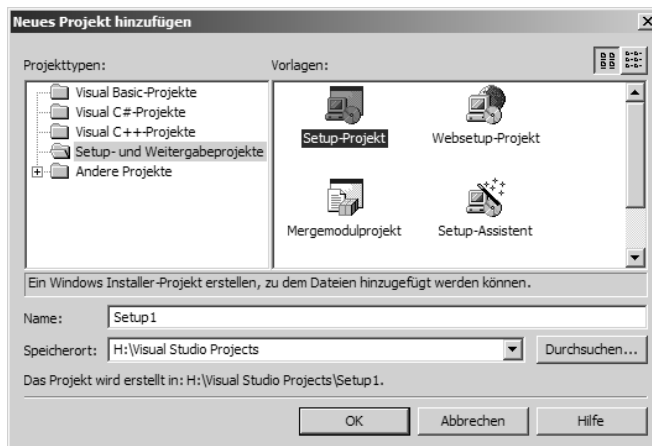


Abbildung 12.3: Ein neues Setup-Projekt wird hinzugefügt.

Wenn Sie das Projekt von Hand erstellt haben, müssen Sie dem Ordner `ANWENDUNGSORDNER` des Projekts die Dateien hinzufügen, die Sie installieren wollen. Der Wizard nimmt Ihnen die Hauptarbeit ab, indem er Ihnen auflistet, welche Dateien verfügbar sind. Sie müssen dann nur noch auswählen, was hinzugefügt werden soll und was nicht.

Hinzufügen der Installationsfunktionen

Das Setup-Programm soll nun die im Projekt vorhandenen Funktionen `Install`, `Uninstall`, `Commit` und `Rollback` bei der Installation ausführen. Diese sind wie gesagt Bestandteile der Basisklasse der hinzugefügten Installer, d.h. Sie sehen sie zwar nicht, aber das Setup-Projekt findet sie schon. In jedem Fall muss eine so genannte *Benutzerdefinierte Aktion* hinzugefügt werden. Wir wechseln über das Kontextmenü des Setup-Projekts in die entsprechende Ansicht. Wählen Sie dort zunächst den Menüpunkt `ANSICHT | EDITOR | BENUTZERDEFINIERTER AKTIONEN`. Wählen Sie nun aus dem Kontextmenü des Haupteintrags den einzigen verfügbaren Menüpunkt `BENUTZERDEFINIERTER AKTION HINZUFÜGEN`. Der nun erscheinende Dialog zeigt das Dienstprogramm. Wählen Sie es aus und klicken Sie auf `OK`, es werden dann alle benötigten Aktionen hinzugefügt.

Damit wäre das Installationsprogramm auch schon fertig gestellt. Sie können das Setup-Projekt nun erzeugen und ausführen. Der Dienst wird installiert, aber noch nicht gestartet (da ja festgelegt wurde, dass er nur manuell gestartet werden soll). Nach der Installation finden Sie den Dienst unter `DIENSTE` in der Systemkonfiguration, wie in Abbildung 12.4 gezeigt.

Sie können den Dienst jetzt starten und beenden und werden sehen, dass in die angegebene Datei jede Minute ein Eintrag geschrieben wird.

Hierbei handelt es sich natürlich nur um ein sehr einfaches Beispiel. Es geht auch nur darum, Ihnen zu zeigen, wie ein Dienst erstellt wird. Es handelt sich eigentlich auch nur um ein Programm, das unter der Windows-Oberfläche läuft, aber in der Regel keine Bildschirmausgabe liefert. Oft laufen Dienste auch auf einem Server laufen, womit sich eine Ausgabe z.B. über einen modalen Dialog sogar verbietet – einen Dialog würde auf einem Server vermutlich niemand bestätigen und das System würde stillstehen.

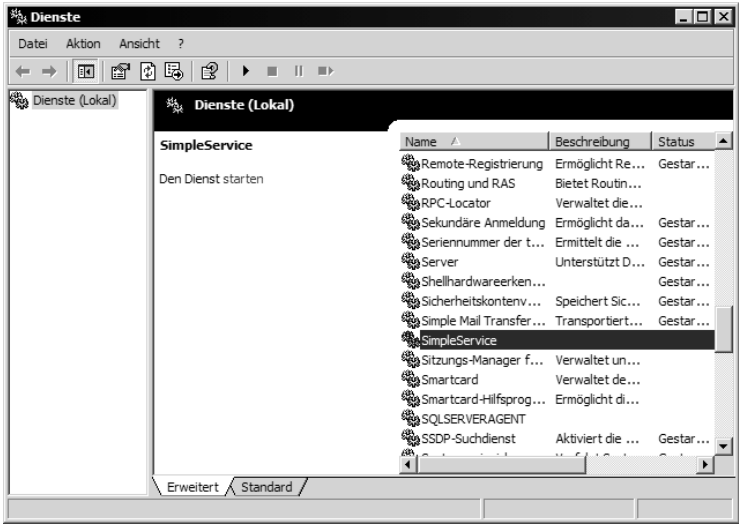


Abbildung 12.4: Der hinzugefügte Dienst

12.4 Die Klasse ServiceController

Sie können unter .NET nicht nur eigene Dienste erstellen, Sie können sie auch überwachen. Zu diesem Zweck gibt es die `ServiceController`-Klasse, die ebenfalls im Namespace `System.ServiceProcess` deklariert ist. Mit Hilfe dieser Klasse können Sie Informationen über einen Dienst bekommen oder auch herausfinden, welche Dienste aktuell installiert sind.

12.4.1 Eigenschaften und Methoden

Eine Instanz der Klasse `ServiceController` steht für einen Dienst, der im System installiert ist. Über diverse Eigenschaften erhalten Sie die gewünschten Informationen. Eine Liste der Eigenschaften finden Sie in Tabelle 12.5.

Eigenschaft	Bedeutung
<code>CanPauseAndContinue</code>	Liefert <code>True</code> , wenn Anforderungen zum Anhalten bzw. zum Fortsetzen an den Dienst gesendet werden können
<code>CanShutDown</code>	Liefert <code>True</code> , wenn der Dienst auf das Herunterfahren des Rechners reagieren kann

Tabelle 12.5: Eigenschaften der Klasse `ServiceController`

Eigenschaft	Bedeutung
CanStop	Liefert True, wenn der Dienst angehalten werden kann
DependentServices	Liefert ein Array von Diensten, von denen dieser Dienst abhängt
ServicesDependentOn	Liefert ein Array von Diensten, die von diesem Service abhängen
DisplayName	Liefert den angezeigten Namen des Dienstes
MachineName	Liefert den Namen des Computers, auf dem der Dienst läuft
ServiceName	Liefert den internen Namen des Dienstes
ServiceType	Liefert die Art des Dienstes
Status	Liefert den aktuellen Status des Dienstes

Tabelle 12.5: Eigenschaften der Klasse ServiceController (Fortsetzung)

Die Klasse `ServiceController` bietet auch die Möglichkeit, ohne Verwendung der Management-Konsole einen Dienst anzuhalten, fortzusetzen oder zu beenden. Die Methoden der Klasse finden Sie in Tabelle 12.6.

Methode	Bedeutung
Close()	Trennt die Verbindung zwischen der Instanz der Klasse und dem Dienst
Continue()	Sendet das Kommando zum Weitermachen an einen Dienst, nachdem dieser angehalten wurde
GetDevices()	<code>GetDevices()</code> ist eine statische (<code>Shared</code>) Methode. Mit ihr erhalten Sie eine Liste von <code>ServiceController</code> -Objekten, die Gerätetreiberdienste repräsentieren.
GetServices()	Auch <code>GetServices()</code> ist eine <code>Shared</code> Methode. Sie liefert ein Array aus <code>ServiceController</code> -Objekten. Mit dieser Methode können Sie alle Dienste auslesen, die auf diesem System installiert sind.
Pause()	Hält einen Dienst an
Refresh()	Aktualisiert alle Eigenschaften des Dienstes
Start()	Startet einen Dienst
Stop()	Beendet einen Dienst

Tabelle 12.6: Methoden der Klasse ServiceController

Methode	Bedeutung
WaitForStatus()	Wartet, bis ein Dienst den angegebenen Status erreicht hat
ExecuteCommand()	Ermöglicht es, eigene Kommandos an den Dienst zu senden, die über die Methode OnCustomCommand() innerhalb des Dienstes ausgewertet und dann entsprechend ausgeführt werden können.

Tabelle 12.6: Methoden der Klasse ServiceController (Fortsetzung)

12.4.2 Dienste auflisten

In diesem Beispiel sollen Dienste abhängig davon aufgelistet werden, ob sie laufen, gestoppt oder angehalten sind. Die Oberfläche des Programms besteht aus einer Listbox und vier Buttons, drei für die Auflistung und einer zum Beenden des Programms. In einem Label wird die Art des Dienstes angezeigt.

In der Methode FillListBox() wird eben diese Listbox mit den Diensten gefüllt. Über den Parameter state kann ausgewählt werden, welche Dienste hinzugefügt werden sollen. state ist vom Typ ServiceControllerStatus. Dabei handelt es sich um eine Aufzählung, die alle verwendbaren Statustypen beinhaltet.

Den Quelltext des Programms sehen Sie in Listing 12.5.

```
'=====
' ServiceLister.sln
' Copyright 2002 Frank Eller
' Listet Windows-Dienste auf
'=====

Imports System.ServiceProcess

Public Class Form1
    Inherits System.Windows.Forms.Form

    #Region " Windows Form Designer generated code "

        'Der Code, der von VS.NET eingefügt wurde, wird
        'aus Übersichtsgründen nicht dargestellt.

    #End Region

    Private Sub FillListBox( _
        ByVal state As ServiceControllerStatus)

        Dim allServices() As ServiceController
```

```
Dim actServices() As ServiceController
Dim actService As ServiceController
Dim count As Integer = 0

' --- Services holen
allServices = ServiceController.GetServices()

' --- Und checken
For Each actService In allServices
    If actService.Status = state Then
        ReDim Preserve actServices(count)
        actServices(count) = actService
        count = count + 1
    End If
Next

' --- Jetzt zuweisen
lboxServices.DataSource = actServices
lboxServices.DisplayMember = "DisplayName"
End Sub

Private Sub btnExit_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnExit.Click

    Me.Close()
End Sub

Private Sub btnRunning_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnRunning.Click

    FillListBox(ServiceControllerStatus.Running)
End Sub

Private Sub btnStopped_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnStopped.Click

    FillListBox(ServiceControllerStatus.Stopped)
End Sub

Private Sub btnPaused_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnPaused.Click

    FillListBox(ServiceControllerStatus.Paused)
```

```

End Sub

Private Sub lbxServices_SelectedIndexChanged( _
    ByVal sender As Object, _
    ByVal e As System.EventArgs) _
    Handles lbxServices.SelectedIndexChanged

    ' --- Daten anzeigen
    Dim thisService As ServiceController
    thisService = CType(lbxServices.SelectedItem, _
        ServiceController)
    lblServiceType.Text = _
        thisService.ServiceType.ToString()
End Sub
End Class

```

Listing 12.5: Die Methoden des Programms ServiceLister

Das Programm im Einsatz zeigt Abbildung 12.5.

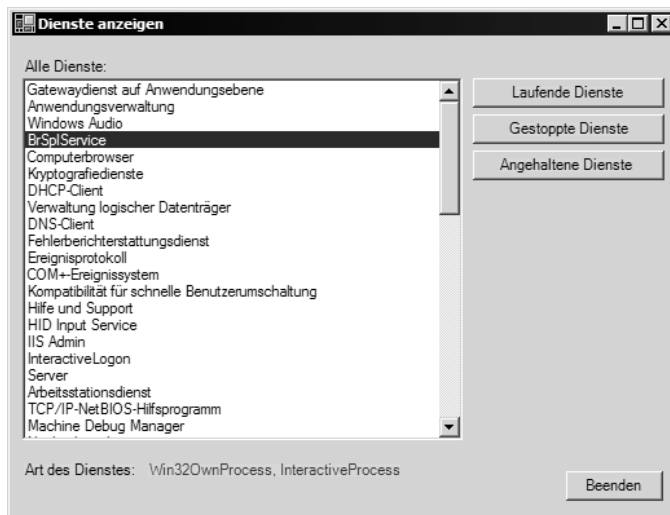


Abbildung 12.5: Die Dienste in einer Listbox

Die weiteren Möglichkeiten, die Sie haben, wenn ein Dienst einmal identifiziert ist und als Objekt vom Typ `System.ServiceProcess.ServiceController` vorliegt, sind eigentlich selbsterklärend. Wichtig ist nur, dass Sie stets den Status eines Dienstes kontrollieren, z. B. wenn Sie einen Dienst stoppen oder starten wollen.

13 Namespace Microsoft.Win32

In diesem Namensraum finden Sie eigentlich nur drei Klassen, die sich sinnvoll in eigenen Applikationen einsetzen lassen. Zum einen ist das die Klasse `SystemEvents`, mit der Sie auf Ereignisse reagieren können, die von Windows selbst ausgelöst werden; zum anderen sind es die Klassen `Registry` und `RegistryKey`, mit denen der Zugriff auf die Windows-Registrierdatenbank ermöglicht wird.

Der Name sagt bereits aus, dass es bei diesem Namensraum um Windows-spezifische Klassen geht. Falls Sie Ihre Applikation also auf einem anderen Betriebssystem laufen lassen wollen (das .NET-Framework ermöglicht dies ja), sollten Sie diese Klassen nicht einsetzen.



13.1 Die Klasse SystemEvents

Grundsätzlich besteht der Sinn und Zweck dieser Klasse darin, es Ihnen zu ermöglichen, auf Ereignisse des Systems zu reagieren. Einige gebräuchliche Ereignisse sind bereits definiert, weitere können Sie mit Hilfe der Methode `InvokeOnEventsThread()` abfangen.

**InvokeOnEvents
Thread()**

13.1.1 Standardereignisse

Alle Delegates für die Standard-Ereignisse und auch die `EventArgs`-Klassen für den jeweiligen zweiten Parameter (sofern es sich dabei nicht um einen Parameter vom Typ `EventArgs` selbst handelt) sind im gleichen Namespace deklariert. Eine detaillierte Auflistung der Ereignisse finden Sie auch in der Online-Dokumentation unter [VISUAL STUDIO .NET | .NET-FRAMEWORK | REFERENZ | KLASSEN-BIBLIOTHEK | MICROSOFT.WIN32](#). Die folgende Liste soll Ihnen nur eine kurze Übersicht geben.

- ▶ `DisplaySettingsChanged` tritt bei einem Wechsel der Bildschirm-auflösung oder der Farbtiefe auf.
- ▶ `EventThreadShutDown` tritt auf, wenn der Thread, der in einer Anwendung für die Behandlung der Systemereignisse zuständig ist, beendet wird.
- ▶ `InstalledFontsChanged` tritt auf, wenn neue Schriftarten installiert oder entfernt werden.
- ▶ `LowMemory` tritt auf, wenn der Speicherplatz knapp wird.
- ▶ `PaletteChanged` tritt auf, wenn sich die verwendete Farbpalette ändert. Das kann auch bei einem Wechsel der aktiven Anwendung der Fall sein.
- ▶ `PowerModeChanged` tritt auf, wenn der Rechner in den Schlafmodus geschickt oder wieder aufgeweckt wird, allerdings nur, wenn dies wirklich passiert.
- ▶ `SessionEnded` tritt auf, wenn der Benutzer sich abmeldet.
- ▶ `SessionEnding` tritt auf, wenn der Benutzer sich abmelden will, d.h. wenn die entsprechende Anfrage von Windows auf dem Bildschirm erscheint.
- ▶ `TimeChanged` tritt auf, wenn die Uhrzeit geändert wird.
- ▶ `TimerElapsed` tritt auf, wenn einer der Windows-Zeitgeber abgelaufen ist.
- ▶ `UserPreferenceChanged` tritt auf, wenn die Einstellungen eines Benutzers geändert wurden.
- ▶ `UserPreferenceChanging` tritt auf, wenn die Einstellungen eines Benutzers gerade geändert werden.

Installation von Schriftarten kontrollieren

Der folgende Programmcode zeigt die Anwendung eines solchen Ereignisses. In diesem Fall wird abgefangen, ob ein neuer Zeichensatz installiert bzw. ob einer entfernt wurde. Wenn dieses Ereignis auftritt, soll die Liste der Zeichensätze neu geladen werden.

Das Hauptformular der Anwendung besteht aus einer Combobox, die die System-Schriftarten enthalten soll. Ändert sich etwas an den Schriftarten, wird also eine hinzugefügt oder entfernt, wird diese Liste neu geladen.

Die Liste aller installierten Schriftarten finden Sie über die Klasse `InstalledFontCollection` aus dem Namespace `System.Drawing.Text`.
Listing 13.1 zeigt den Quellcode des Programms.

```
'=====
' FontCheck.sln
' Copyright 2002 Frank Eller
' Überwachen der Schriftarten
'=====

Imports Microsoft.Win32
Imports System.Drawing.Text

Public Class Form1
    Inherits System.Windows.Forms.Form

    #Region " Windows Form Designer generated code "

        ' Der Code, der von VS.NET eingefügt wurde, wird
        ' aus Übersichtsgründen nicht dargestellt.

    #End Region

    ' --- Methode zum Laden der Fonts
    ' --- Lädt die Fonts in eine Combobox mit Namen
    ' --- cbxFonts
    Private Sub LoadFonts()

        Dim ifc As New InstalledFontCollection()
        Dim fFamily() As FontFamily = ifc.Families
        Dim fonts(fFamily.Length - 1) As String
        Dim i As Integer

        For i = 0 To fonts.Length - 1
            fonts(i) = fFamily(i).Name
        Next

        cbxFonts.Items.Clear()
        cbxFonts.Items.AddRange(fonts)

        ifc.Dispose()

    End Sub

    ' --- EventHandler für Änderung der Fonts
    Private Sub FontsHaveChanged( _
        ByVal sender As Object, _
        ByVal e As EventArgs)

        LoadFonts()
    End Sub
```

```

Private Sub Form1_Load( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles MyBase.Load

    ' --- EventHandler verbinden
    AddHandler SystemEvents.InstalledFontsChanged, _
        AddressOf Me.FontsHaveChanged
    Me.LoadFonts()

End Sub

Private Sub btnExit_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnExit.Click

    Me.Close()

End Sub

End Class

```

Listing 13.1: Kontrolle der installierten Schriftarten [FontCheck.sln]

Wie Sie am Listing sehen können, ist die Verwendung der zur Verfügung gestellten Ereignisse nicht besonders schwierig. Die Methode `FontsHaveChanged()` wird der Aufrufliste des Ereignisses hinzugefügt, wodurch diese Methode immer dann aufgerufen wird, wenn sich die Anzahl der Schriftarten ändert.

Wechsel der Bildschirmauflösung kontrollieren

Ebenso einfach ist es, die Änderung der Bildschirmauflösung zu überwachen. Wenn der Anwender etwas daran ändert, wird ebenfalls ein Systemereignis ausgelöst, das auf die gleiche Art wie oben beschrieben abgefangen werden kann. Listing 13.2 zeigt den Quellcode des Programms.

```

'=====
' ScreenCheck.sln
' Copyright 2002 Frank Eller
' Überwachen der Bildschirmauflösung
'=====

Imports Microsoft.Win32

Public Class Form1
    Inherits System.Windows.Forms.Form

```



```

#Region " Windows Form Designer generated code "

    'Der Code, der von VS.NET eingefügt wurde, wird
    'aus Übersichtsgründen nicht dargestellt.

#End Region

    ' --- Der EventHandler für die Display-Änderungen
    Private Sub DisplayHasChanged( _
        ByVal sender As Object, _
        ByVal e As EventArgs)

        MessageBox.Show("Die Bildschirm Einstellungen "+ _
            "wurden geändert")

    End Sub

    Private Sub Form1_Load( _
        ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles MyBase.Load

        ' --- Handler hinzufügen
        AddHandler SystemEvents.DisplaySettingsChanged, _
            AddressOf DisplayHasChanged

    End Sub

    Private Sub button1_Click( _
        ByVal sender As System.Object, _
        ByVal e As System.EventArgs) _
        Handles button1.Click

        Me.Close()

    End Sub
End Class

```

Listing 13.2: Kontrolle der Bildschirmauflösung [ScreenCheck.sln]

13.2 Zugriff auf die Registry

Mit Erscheinen des .NET-Frameworks rückt die Registry ein wenig in den Hintergrund. Der Zugriff auf manche Werte mag zwar manchmal noch wichtig sein, dennoch ist die Registry ein Bestandteil von Windows und der Zugriff darauf macht eine Applikation daher betriebssystemabhängig. Sie sollten also nur dann auf die Registry zugreifen, wenn Sie sich sicher sind, dass dieser Zugriff

auch wirklich nötig ist bzw. wenn Sie wissen, dass Ihre Applikation nur unter Windows läuft. Andernfalls benutzen Sie die XML-Konfigurationsdateien Ihrer Applikation, in denen Sie relevante Werte ablegen können.

Interessant ist die Registry natürlich zum Herausfinden einiger Einstellungen von Windows oder deren Manipulation. Der Namespace `Microsoft.Win32` enthält alle notwendigen Klassen für den Zugriff auf die Registrierdatenbank.

13.2.1 Die Klasse Registry

Registry Die Klasse `Registry` enthält Eigenschaften, die die Hauptschlüssel der Registrierdatenbank repräsentieren. Anders als möglicherweise vermutet ist diese Klasse nicht für den Zugriff auf die Daten der Registrierdatenbank gedacht, sondern liefert lediglich den Teil derselben, auf den wir zugreifen wollen. Der zurückgelieferte Wert ist ein Objekt vom Typ `RegistryKey`, über das später auf die eigentlichen Daten bzw. Unterschlüssel zugegriffen wird. Tabelle 13.1 zeigt die Eigenschaften der Klasse `Registry` und ihre Bedeutung.

Eigenschaft	Bedeutung
ClassesRoot	Liefert den Schlüssel HKEY_CLASSES_ROOT
CurrentUser	Liefert den Schlüssel HKEY_CURRENT_USER
LocalMachine	Liefert den Schlüssel HKEY_LOCAL_MACHINE
Users	Der Schlüssel HKEY_USERS enthält Unterschlüssel für jeden Benutzer des Rechners. Der Zugriff auf diese Daten wird von allen Windows-Betriebssystemen ab Windows 98 unterstützt.
Performance Data	Bei diesem Schlüssel handelt es sich nicht wirklich um einen Schlüssel, da die Daten, auf die Sie zugreifen können, nicht physikalisch in der Registry gespeichert werden. Es handelt sich um dynamische Daten, die eine Applikation während ihrer Laufzeit erzeugt. Beim Zugriff auf die Daten werden diese vom System aktuell gesammelt und zurückgeliefert.
Current Config	Liefert den Schlüssel HKEY_CURRENT_CONFIG. Obwohl Sie diesen Schlüssel auch in der Registry finden, handelt es sich eigentlich um einen Unterschlüssel von HKEY_LOCAL_MACHINE.
DynData	Ermöglicht den Zugriff auf dynamische Daten unter Windows 98 / ME

Tabelle 13.1: Die Eigenschaften der Klasse Registry

Der zurückgelieferte Wert ist vom Typ `RegistryKey`. Eine Zuweisung sähe demnach in Visual Basic folgendermaßen aus:

```
Dim rKey As RegistryKey
rKey = Registry.CurrentUser
```

Das C#-Äquivalent wäre:

```
RegistryKey rKey = Registry.CurrentUser;
```

Während die Klasse `Registry` in den Eigenschaften den zu öffnen- den Basisschlüssel zurückliefert, gibt es auch noch einen Aufzählungstyp namens `RegistryHive`, der das Gleiche in Form eines `enum`-Datentyps enthält. Die Bezeichnungen der einzelnen Elemente sind die gleichen wie die der Eigenschaften von `Registry`. Den Datentyp `RegistryHive` benötigt man beispielsweise für die Methode `OpenRemoteBaseKey()`, die es ermöglicht, einen Schlüssel der Registry eines anderen Computers im Netzwerk zu öffnen.

RegistryHive

13.2.2 Die Klasse `RegistryKey`

Das `RegistryKey`-Objekt, das durch die Eigenschaften zurückgeliefert wird, können wir nun verwenden, um auf die einzelnen Unterschlüssel bzw. Daten zuzugreifen. `RegistryKey` bietet eine große Anzahl an Methoden, die einen komfortablen Zugriff auf die enthaltenen Daten ermöglichen. Tabelle 13.2 zeigt die Eigenschaften, Tabelle 13.3 die Methoden der Klasse `RegistryKey`.

RegistryKey

Eigenschaft	Bedeutung
Name	Liefert den Namen des Schlüssels zurück
SubKeyCount	Liefert die Anzahl der enthaltenen Unterschlüssel. Dabei wird allerdings nur eine Ebene berücksichtigt, d.h. Unterschlüssel eines Unterschlüssels werden nicht mitgezählt.
ValueCount	Liefert die Anzahl der unter diesem Schlüssel enthaltenen Werte

Tabelle 13.2: Die Eigenschaften von `RegistryKey`

Methode	Bedeutung
Close()	Schließt einen geöffneten Schlüssel
CreateSubKey()	Erzeugt oder öffnet einen Unterschlüssel des aktuellen Schlüssels
DeleteSubKey()	Löscht einen untergeordneten Schlüssel

Tabelle 13.3: Die Methoden von `RegistryKey`

Methoden	Bedeutung
DeleteSubKey-Tree()	Löscht den angegebenen untergeordneten Schlüssel inklusive der darin enthaltenen Unterschlüssel
DeleteValue()	Löscht den angegebenen Wert im aktuellen Schlüssel
Flush()	Schreibt die Änderungen physikalisch auf die Festplatte. Die Methode Close() ruft Flush() ebenfalls auf.
GetSubKeyNames()	Liefert die Namen aller Unterschlüssel zurück
GetValue()	Liefert den angegebenen Wert zurück
GetValueNames()	Liefert die Bezeichner aller Werte zurück, die der Schlüssel enthält
OpenRemoteBaseKey()	Ermöglicht das Öffnen eines Registry-Schlüssels auf einem anderen Computer
OpenSubKey()	Öffnet den angegebenen Unterschlüssel zum Lesen
SetValue()	Legt einen Wert fest

Tabelle 13.3: Die Methoden von RegistryKey (Fortsetzung)

Die Funktionen der Methoden aus Tabelle 13.3 sind nahezu selbst-erklärend. Eine genauere Betrachtung verdienen allerdings die Methoden `OpenSubKey()`, `CreateSubKey()` und `OpenRemoteBaseKey()`.

**OpenSubKey()
und Create-
SubKey()**

Die Methode `OpenSubKey()` öffnet einen Schlüssel für den Lesezu-griff – nicht für den Schreibzugriff, auch wenn der Bezeichner der Methode das suggeriert. Zum Öffnen eines Unterschlüssels, in den Sie auch Werte schreiben wollen, verwenden Sie `CreateSubKey()`. Diese Methode öffnet einen bestehenden Unterschlüssel für den lesenden und schreibenden Zugriff bzw. erstellt einen Schlüs-sel, wenn dieser noch nicht existiert.

**OpenRemote-
BaseKey()**

Etwas Besonderes ist die Methode `OpenRemoteBaseKey()`. Mit dieser Methode können Sie den Basisschlüssel eines anderen Computers öffnen. Es handelt sich um eine statische Methode, was logisch ist, wenn man sich die Funktionsweise ansieht. Da jedes Öffnen eines Unterschlüssels einen übergeordneten Schlüssel (also ein Objekt vom Typ `RegistryKey`) benötigt, ist dies hier nicht der Fall, da ein Basisschlüssel geöffnet werden soll. Die Funktion benötigt als Parameter einmal den gewünschten Schlüssel und den Namen des Computers, auf dem der Schlüssel geöffnet werden soll. Sie können auch einen Basisschlüssel des lokalen Rechners öffnen, wenn Sie für den Computernamen einen leeren String übergeben.

Existenz eines Schlüssels überprüfen

Kommen wir zu einigen Beispielen, um die Funktionsweisen der Klassen zu verdeutlichen. Im ersten Beispiel soll kontrolliert werden, ob ein Schlüssel existiert. Das ist relativ leicht zu bewerkstelligen. Die Methode `OpenSubKey()` liefert den Wert `Nothing` (null in C#) zurück, wenn der angegebene Unterschlüssel nicht existiert, und eben den Unterschlüssel in Form eines `RegistryKey`-Objekts, wenn er existiert.

Auf dem Formular befindet sich eine Combobox, die mit den verschiedenen möglichen Basisschlüsseln gefüllt wird. Ein Textfeld gibt dann den Unterschlüssel des Basisschlüssels an, der kontrolliert werden soll. Der Name der Combobox ist `cbxMainKeys`, der Name des Textfeldes `tbxSubKey`. Listing 13.3 zeigt den Quellcode des Beispielprogramms.

```
'=====
' RegistryAccess.sln
' Copyright 2002 Frank Eller
' Zugriff auf die Registry
'=====

Imports Microsoft.Win32

Public Class Form1
    Inherits System.Windows.Forms.Form

    #Region " Windows Form Designer generated code "

        'Der Code, der von VS.NET eingefügt wurde, wird
        'aus Übersichtsgründen nicht dargestellt.

    #End Region

    Private Sub Form1_Load( _
        ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles MyBase.Load

        ' --- Deklarationen
        Dim registryMainKeys() As String
        Dim rHive As RegistryHive
        Dim aKey As String

        cbxMainKeys.Items.Clear()
        registryMainKeys = _
            System.Enum.GetNames(rHive.GetType())

        For Each aKey In registryMainKeys
```

```

        cbxMainKeys.Items.Add(aKey)
    Next

    lblResult.Text = String.Empty

End Sub

Private Sub btnExists_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnExists.Click

    ' --- Deklarationen
    Dim mainKey As RegistryKey
    Dim cbxMainKeyValue As Integer
    Dim subKey As String

    mainKey = Registry.ClassesRoot
    cbxMainKeyValue = cbxMainKeys.SelectedIndex
    subKey = txtSubKey.Text

    Select Case cbxMainKeyValue
        Case -1 : Return
        Case 0 : mainKey = Registry.ClassesRoot
        Case 1 : mainKey = Registry.CurrentUser
        Case 2 : mainKey = Registry.LocalMachine
        Case 3 : mainKey = Registry.Users
        Case 4 : mainKey = Registry.PerformanceData
        Case 5 : mainKey = Registry.CurrentConfig
        Case 6 : mainKey = Registry.DynData
    End Select

    Try
        If mainKey.OpenSubKey(subKey) Is Nothing Then
            lblResult.ForeColor = Color.DarkRed
            lblResult.Text = "Der Unterschlüssel " + _
                "konnte nicht gefunden werden"
        Else
            lblResult.ForeColor = Color.DarkGreen
            lblResult.Text = "Der Unterschlüssel " + _
                "wurde gefunden"
        End If
    Catch
        MessageBox.Show("Ein Fehler ist "+ _
            "beim Zugriff aufgetreten!", _
            "Fehler", MessageBoxButtons.OK, _
            MessageBoxIcon.Stop)
    Finally
        mainKey.Close()
    End Try
End Sub

```

```
End Try
```

```
End Sub
```

```
End Class
```

Listing 13.3: Zugriff auf die Registry [RegistryAccess.sln]

Wie angesprochen enthält die Combobox die Basisschlüssel, von denen einer ausgewählt werden muss. Über die `Select Case`-Anweisung wird entsprechend des ausgewählten Elements der Combobox das entsprechende `RegistryKey`-Objekt ermittelt. Von diesem muss nun ein Unterschlüssel kontrolliert werden, was einfach dadurch erreicht wird, dass versucht wird, diesen Unterschlüssel, dessen Name in `txtSubKey` angegeben ist, zu öffnen. Falls der Unterschlüssel existiert, wird er zurückgeliefert, andernfalls ist der zurückgelieferte Wert `Nothing` (`null` in C#). Am Ende wird noch kontrolliert, ob eine Exception aufgetreten ist, und im `Finally`-Block werden alle Schlüssel in jedem Fall wieder geschlossen.

Eine mögliche Ausgabe des Programms sehen Sie in Abbildung 13.1.

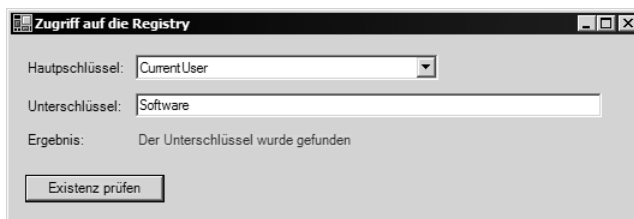


Abbildung 13.1: Existenzkontrolle eines Unterschlüssels (RegistryAccess)

Werte eines Schlüssels anzeigen

Eine sinnvolle Möglichkeit ist auch das Auslesen der Registry bzw. der Registry-Werte. Die Methode `GetValueNames()` liest hier lediglich die Namen aus, die auch gesetzt sind – der Standardwert eines Unterschlüssels, bei dem es sich üblicherweise um einen nicht gesetzten Wert handelt, wird nicht mit ausgelesen.

Die Funktion `GetValueNames()` liefert uns ein Array mit den Bezeichnern der Werte, die der Schlüssel enthält. `GetValue()` liefert den Wert eines bestimmten Bezeichners. Falls ein Bezeichner nicht existiert, liefert `GetValue` `Nothing()` zurück.

GetValue()

Wie vorher wird auch in diesem Beispielprogramm eine Combo-box namens `cbxMainKeys` verwendet, um den Hauptschlüssel festzulegen. Die Textbox `txtSubKey` dient zur Eingabe des gewünschten Unterschlüssels. Über einen Klick auf den Button `btnGetValues` erhalten Sie die Werte.

```
'=====
' RegistryReader.sln
' Copyright 2002 Frank Eller
' Lesen aus der Registry
'=====

Imports Microsoft.Win32

Public Class Form1
    Inherits System.Windows.Forms.Form

    #Region " Windows Form Designer generated code "

        'Der Code, der von VS.NET eingefügt wurde, wird
        'aus Übersichtsgründen nicht dargestellt.

    #End Region

    Private Sub Form1_Load( _
        ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles MyBase.Load

        ' --- Deklarationen
        Dim registryMainKeys() As String
        Dim rHive As RegistryHive
        Dim aKey As String

        cbxMainKeys.Items.Clear()
        registryMainKeys = _
            System.Enum.GetNames(rHive.GetType())

        For Each aKey In registryMainKeys
            cbxMainKeys.Items.Add(aKey)
        Next

    End Sub

    Private Sub btnGetValues_Click( _
        ByVal sender As System.Object, _
        ByVal e As System.EventArgs) _
        Handles btnGetValues.Click

        ' --- Deklarationen
        Dim mainKey As RegistryKey
```



```
Dim mainKeysValue As Integer
Dim subKey As RegistryKey

' --- Listbox leeren
lboxValues.Items.Clear()

' --- Hauptschlüssel einstellen

mainKey = Registry.ClassesRoot
mainKeysValue = cbxMainKeys.SelectedIndex

Select Case mainKeysValue
    Case -1 : Return
    Case 0 : mainKey = Registry.ClassesRoot
    Case 1 : mainKey = Registry.CurrentUser
    Case 2 : mainKey = Registry.LocalMachine
    Case 3 : mainKey = Registry.Users
    Case 4 : mainKey = Registry.PerformanceData
    Case 5 : mainKey = Registry.CurrentConfig
    Case 6 : mainKey = Registry.DynData
End Select

' --- Unterschlüssel öffnen und Werte hinzufügen
Try
    subKey = mainKey.OpenSubKey(txtSubKey.Text)

    If subKey Is Nothing Then
        MessageBox.Show("Dieser Unterschlüssel " + _
            "existiert nicht", _
            "Fehler", MessageBoxButtons.OK, _
            MessageBoxIcon.Hand)
    Else
        Dim vnames() As String
        Dim vName As String
        Dim valueStr As String
        Dim addStr As String

        vnames = subKey.GetValueNames()

        For Each vName In vnames
            valueStr = _
                subKey.GetValue(vName).ToString()
            addStr = vName + " : " + valueStr
            lboxValues.Items.Add(addStr)
        Next

        subKey.Close()
    End If
Catch
    MessageBox.Show("Ein Fehler ist beim " + _
        "Zugriff aufgetreten", _
```

```

        "Fehler", MessageBoxButtons.OK, _
        MessageBoxIcon.Hand)
    Finally
        mainKey.Close()
    End Try

End Sub

Private Sub btnExit_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnExit.Click

    Me.Close()

End Sub
End Class

```

Listing 13.4: Werte aus der Registry lesen [RegistryReader.sln]

Der Hauptschlüssel wird wieder eingestellt wie auch im vorherigen Beispiel, auch der Try...Catch-Block findet sich im Beispiel wieder, um etwaige Fehler beim Öffnen eines Schlüssels abzufangen. Existiert ein Schlüssel nicht oder wird ein Fehler festgestellt, erhält der Anwender eine Mitteilung über eine Messagebox.

Die Methode `GetValueNames()` des zum Lesen geöffneten Unterschlüssels liefert ein Array mit allen Bezeichnern der im Schlüssel enthaltenen Werte zurück. Der jeweils dazugehörige Wert kann über `GetValue()` ermittelt werden. `GetValue()` benötigt als Parameter den Namen des Werts, der zurückgeliefert werden soll.

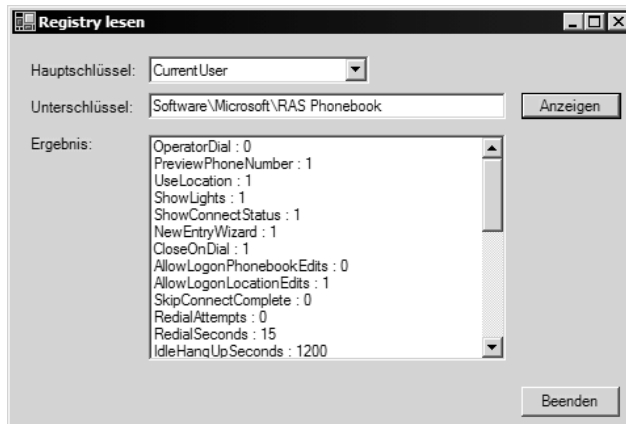


Abbildung 13.2: Lesen der Registry (RegistryReader)

Da nicht bekannt ist, wie viele Werte enthalten sind, verwenden Sie am besten die `For Each`-Schleife, um das Array mit den Bezeichnern zu durchlaufen und den jeweiligen Wert zu ermitteln.

Der Rest ist eigentlich einfach. Aus dem Wert und dem Bezeichner wird ein String gebildet und der Listbox hinzugefügt. Ein mögliches Resultat des Programms sehen Sie in Abbildung 13.2.

Schreiben in die Registry

Zum Schreiben von Daten in die Registry bietet die Klasse `RegistryKey` die Methode `SetValue()` an, die als Parameter den Bezeichner des Werts sowie den Wert selbst erwartet. Der Wert selbst wird als `System.Object` behandelt, Sie können also alle Arten von Daten übergeben.

SetValue()

Wenden Sie `SetValue()` auf einen bereits im Unterschlüssel existierenden Wert an, so wird dieser geändert. Existiert er nicht, wird er neu angelegt. `SetValue()` tut dies automatisch.

Der Schlüssel, in den geschrieben werden soll, muss jetzt mit Hilfe der Methode `CreateKey()` geöffnet werden. Der Unterschied zu `OpenKey()` besteht darin, dass der Unterschlüssel zum Lesen und zum Schreiben geöffnet und, falls er nicht existiert, automatisch angelegt wird. Eine kleine Routine zum Schreiben in die Registry sehen Sie in Listing 13.5.

CreateKey()

```
'=====
' RegistryWriter.sln
' Copyright 2002 Frank Eller
' Schreiben in die Registry
'=====

Imports Microsoft.Win32

Public Class Form1
    Inherits System.Windows.Forms.Form

    #Region " Windows Form Designer generated code "

        'Der Code, der von VS.NET eingefügt wurde, wird
        'aus Übersichtsgründen nicht dargestellt.

    #End Region

    Private Sub btnWriteValue_Click( _
        ByVal sender As System.Object, _
        ByVal e As System.EventArgs) _
```

```

Handles btnWriteValue.Click

' --- Deklarationen
Dim mainKey As RegistryKey
Dim subkey As RegistryKey
Dim subkeystr As String
Dim valueName As String
Dim valueValue As String

' --- Schlüssel öffnen
mainKey = Registry.CurrentUser
subkeystr = "Software\" + txtSubKey.Text
subkey = mainKey.CreateSubKey(subkeystr)

Try
    valueName = txtValueName.Text
    valueValue = txtValue.Text
    subkey.SetValue(valueName, valueValue)
    subkey.Close()

Catch
    MessageBox.Show("Ein Fehler ist aufgetreten", _
        "Fehler", MessageBoxButtons.OK, _
        MessageBoxIcon.Hand)

Finally
    subkey.Close()

End Try

End Sub

Private Sub btnExit_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnExit.Click

    Me.Close()

End Sub

End Class

```

Listing 13.5: Schreiben in die Registry [RegistryWriter.sln]

Sie sehen, dass es nicht besonders schwierig ist, mit den Daten der Registry zu arbeiten. Abbildung 13.3 zeigt einen Screenshot des Beispielprogramms.

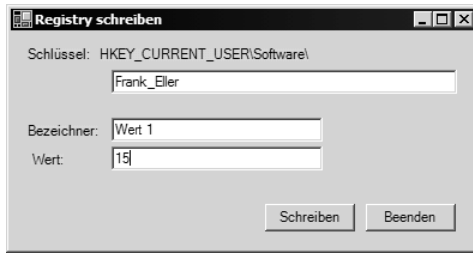


Abbildung 13.3: Schreiben in die Registry (RegistryWriter.exe)

Remote-Zugriff

Die Klassen `Registry` und `RegistryKey` ermöglichen auch den Zugriff auf die Registrierdatenbank anderer Computer im Netzwerk. Dazu müssen allerdings einige Grundvoraussetzungen gegeben sein:

- ▶ Beide Computer, sowohl der zugreifende als auch der Computer, auf den zugegriffen wird, müssen den Remote Registry-Dienst gestartet haben.
- ▶ Beide Computer müssen die Remote-Administration aktiviert haben.

Der Zugriff auf die fremde Registry geschieht über die Methode `OpenRemoteBaseKey`, ansonsten bleibt alles gleich, außer dass jetzt auf eine fremde Registry zugegriffen wird. Beachten Sie jedoch bei diesem Zugriff, dass Sie auch vieles zerstören können. Normalerweise sollten nur Administratoren auf diese Weise auf einen anderen Computer zugreifen können bzw. dürfen.

OpenRemoteBaseKey()

Der Hauptschlüssel, der geöffnet werden soll, wird in Form eines `RegistryHive`-Werts übergeben. Dieser Aufzählungstyp enthält alle Hauptschlüssel, die eine Registry enthalten kann, wobei nicht jede Windows-Plattform alle diese Schlüssel unterstützt. `OpenRemoteBaseKey()` gibt bei erfolgreicher Ausführung ein Objekt des Typs `RegistryKey` zurück, das zum Eintragen weiterer Schlüssel oder Werte verwendet werden kann.

RegistryHive

In diesem Beispiel wird auf die Registry eines Notebooks mit Namen »notebook« zugegriffen, das im Netzwerk hängt. In beiden Rechnern ist der Administrator eingeloggt, so dass es keine Probleme mit den Berechtigungen gibt. Der Schlüssel, der angelegt werden soll, wird unter `Software` eingetragen, danach folgen die Werte. Außer dem Einsatz von `OpenRemoteBaseKey()` hat sich im

Vergleich zum vorherigen Listing nicht viel geändert. Listing 13.6 zeigt den Quellcode des Programms.

```
'=====
' RegistryRemote.sln
' Copyright 2002 Frank Eller
' Remote-Zugriff auf die Registry
'=====

Imports Microsoft.Win32

Public Class Form1
    Inherits System.Windows.Forms.Form

    #Region " Windows Form Designer generated code "

        'Der Code, der von VS.NET eingefügt wurde, wird
        'aus Übersichtsgründen nicht dargestellt.

    #End Region

    Private Sub btnWriteValue_Click( _
        ByVal sender As System.Object, _
        ByVal e As System.EventArgs) _
        Handles btnWriteValue.Click

        ' --- Deklarationen
        Dim mainKey As RegistryKey
        Dim subkey As RegistryKey
        Dim subKeyStr As String
        Dim valueName As String
        Dim valueValue As String

        ' --- Remote-Schlüssel öffnen
        mainKey = RegistryKey.OpenRemoteBaseKey( _
            RegistryHive.LocalMachine, "notebook")
        subKeyStr = "SOFTWARE\" + txtSubKey.Text
        subkey = mainKey.CreateSubKey(subKeyStr)

        Try
            valueName = txtValueName.Text
            valueValue = txtValue.Text
            subkey.SetValue(valueName, valueValue)
            subkey.Close()

        Catch
            MessageBox.Show("Ein Fehler ist aufgetreten", _
                "Fehler", MessageBoxButtons.OK, _
                MessageBoxIcon.Hand)

        Finally
```

```
        subkey.Close()

    End Try

End Sub

Private Sub btnExit_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnExit.Click

    Me.Close()

End Sub

End Class
```

Listing 13.6: Remote-Zugriff auf die Registry [RegistryRemote.sln]

14 Namespace System.Diagnostics

Der Namespace `System.Diagnostics` bietet Klassen für die Arbeit mit drei wesentlichen Elementen in der Windows NT-Produktfamilie (NT4, Windows 2000, Windows XP, Windows .NET):

Zugriff auf Kernfunktionen von Windows

- ▶ Ereignisprotokolle (Eventlogs)
- ▶ Prozesse
- ▶ Leistungsindikatoren (Performance Counter)

Darüber hinaus gibt es in diesem FCL-Namespaces Klassen für folgende Funktionen:

- ▶ Versions- und Ursprungsinformationen über ausführbare Dateien (*.dll*, *.exe*)
- ▶ Debugging und Tracing von Anwendungen (Ausgabe von Protokollinformationen zur Entwicklungs- und Laufzeit)

14.1 Arbeit mit den Ereignisprotokollen

Der Namespace `System.Diagnostics` enthält Klassen, mit denen die Windows-Ereignisprotokolle verwaltet werden können. Insbesondere folgende Aufgaben können mit diesen Klassen ausgeführt werden:

Eventlogs verwenden

- ▶ Lesen von Ereignisprotokolleinträgen
- ▶ Überwachen eines Protokolls auf neue Einträge
- ▶ Schreiben von neuen Einträgen in das Protokoll
- ▶ Anlegen eines neuen, benutzerdefinierten Protokolls
- ▶ Löschen eines benutzerdefinierten Protokolls

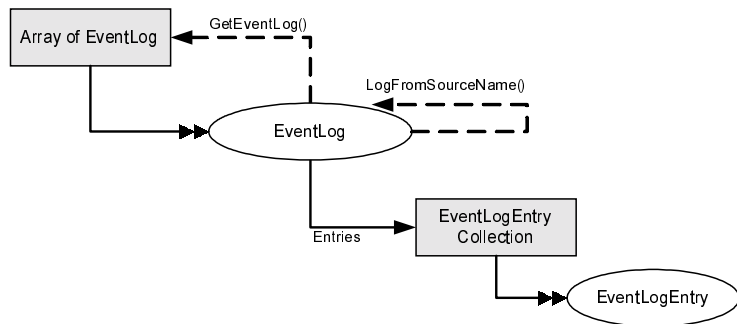
14.1.1 Überblick über die Ereignisprotokoll-Klassen

Klassenüberblick Die Arbeit mit Ereignisprotokollen konzentriert sich auf drei Klassen:

- ▶ `System.Diagnostics.EventLog`
Diese Klasse repräsentiert ein Ereignisprotokoll.
- ▶ `System.Diagnostics.EventLogEntryCollection`
Diese Klasse repräsentiert die Menge der Einträge in einem Ereignisprotokoll.
- ▶ `System.Diagnostics.EventLogEntry`
Diese Klasse repräsentiert einen einzelnen Eintrag in einem Ereignisprotokoll.

Objekthierarchie

Die folgende Grafik zeigt die Objekthierarchie der o.g. Klassen.



(C) Holger@Schwichtenberg.de 2002

Abbildung 14.1: Objekthierarchie der Ereignisprotokoll-bezogenen Klassen in »System.Diagnostics«

Weitere Klassen

Darüber hinaus gibt es Klassen für die Installation von Ereignisprotokollen und für die Änderung von Rechten auf Ereignisprotokolle. Diese werden hier nicht besprochen.

14.1.2 Auslesen eines Ereignisprotokolls

EventLog-Klasse Die Klasse `EventLog` besitzt ein Attribut `Entries`, das auf eine Collection vom Typ `EventLogEntryCollection` verweist. Diese Collection enthält einzelne Objekte des Typs `EventLogEntry` mit jeweils einem

Protokolleintrag. Die Einträge sind nach dem Eintragsdatum geordnet, wobei die ältesten Einträge vorne in der Collection stehen.

Beispiel

Das folgende Beispiel listet Einträge aus dem Anwendungsprotokoll (Name: »Application«) auf. Dabei wird die Anzahl der auszugebenden Einträge über einen Zähler auf die letzten x Einträge begrenzt.

Anwendungs- protokoll lesen

```
' =====
' Beispiel aus FCL-Buch.sln
' Auslesen von Einträgen aus einem
' Ereignisprotokoll
' (C) Holger@Schwichtenberg.de
' =====
Public Sub EventLog_Read()
    ' Name des Ereignisprotokolls
    Const logname As String = "Application"
    ' Anzahl der auszugebenden Einträge
    Const Anzahl As Long = 10

    Dim log As EventLog
    Dim eintrag As EventLogEntry
    Dim Count As Long = 0

    ' -- Zugriff auf das Ereignisprotokoll
    log = New EventLog(logname)

    say("--- Letzte " & Anzahl & _
        " Einträge von " & log.entries.Count & _
        " Einträgen aus dem Protokoll " & _
        log.Log & " auf dem Computer " & _
        log.machinename)

    ' --- Schleife über alle Einträge
    For Each eintrag In log.Entries
        count += 1
        If Count > log.Entries.Count - Anzahl Then
            say(eintrag.EntryType & ":" & _
                eintrag.EventID & ":" & _
                eintrag.Category & ":" & _
                eintrag.Message & ":" & _
                eintrag.Source & ":" & _
                eintrag.TimeGenerated & ":" & _
                eintrag.TimeWritten & ":" & _
```

```

        eintrag.UserName & ":")
    End If
Next

End Sub

```

Listing 14.1: Auslesen von Einträgen aus einem Ereignisprotokoll [eventlog.vb]

Zugriff auf Ereignisprotokolle auf anderen Computern

Fernzugriff Die Klasse `EventLog` unterstützt auch den Fernzugriff. Geben Sie dazu bei der Instanziierung neben dem Protokollnamen als zweiten Parameter einen Computernamen an.

```
EventLog("logname", "computername")
```

14.1.3 Liste aller Ereignisprotokolle

GetEventLogs() Alle Ereignisprotokolle eines Computers existieren in einem Array. Um dieses Array zu erhalten, muss man die statische Methode `GetEventLogs()` aus der Klasse `EventLog` aufrufen. Als Parameter kann ein Computernamen übergeben werden, so dass auch Fernzugriffe möglich sind.

Beispiel

Das Beispiel listet alle Ereignisprotokolle auf einem bestimmten Computer auf.

```

' =====
' Beispiel aus FCL-Buch.sln
' Liste der Ereignisprotokolle auf einem
' bestimmten Computer
' (C) Holger@Schwichtenberg.de
' =====
Public Sub EventLog_Liste()
    Const Computer As String = "Sonne2000"
    Dim Liste() As EventLog
    Dim e As EventLog
    Dim i As Integer
    out( _
        "Liste der EventLogs auf dem Computer " & _
        Computer)
    Liste = EventLog.GetEventLogs(Computer)
    For Each e In Liste
        out(e.Log)
    Next
End Sub

```

Listing 14.2: Liste der Ereignisprotokolle auf einem bestimmten Computer [eventlog.vb]

Ausgabe

Die obige Routine liefert für einen Windows 2000 Server mit aktiviertem Active Directory folgende Liste:

- 1 Liste der EventLogs auf dem Computer Sonne2000
- 2 Application
- 3 Directory Service
- 4 DNS Server
- 5 File Replication Service
- 6 Security
- 7 System

Auf einem Windows 2000 Professional- bzw. einem Windows XP Professional-Computer gibt es dagegen nur vier Ereignisprotokolle:

- 1 Liste der EventLogs auf dem Computer Mars
- 2 Application
- 3 Security
- 4 System

14.1.4 Anlegen eines neuen Ereignisprotokolls

Mit Hilfe der Methode `CreateEventSource()` aus der Klasse `EventLog` kann der Entwickler auch ein neues Ereignisprotokoll anlegen.

Protokoll anlegen

Zu beachten ist, dass lediglich die ersten acht Zeichen zur Unterscheidung verschiedener Ereignisprotokolle herangezogen werden. Die ersten acht Zeichen eines Ereignisprotokollnamens dürfen also nicht den ersten acht Zeichen eines bereits vorhandenen Protokolls entsprechen.



Beispiel

Das folgende Beispiel legt ein neues Ereignisprotokoll mit Namen »DOTNET« auf dem Computer »MARS« an.

```
' =====
' Beispiel aus FCL-Buch.sln
' Anlegen eines neuen Ereignisprotokolls
' (C) Holger@Schwichtenberg.de
' =====
Sub eventlog_newlog()
    Const LOGNAME As String = "DOTNET"
    Const COMPUTER = "Mars"

    Dim log As New EventLog()
    ' --- Prüfen, ob Protokoll existiert
    If log.Exists(logname) Then
```

```

        out("Log existiert bereits!")
    Else
        ' --- Protokoll anlegen
        log.CreateEventSource("Standardquelle", _
            LOGNAME, COMPUTER)
        out("Log wurde angelegt!")
    End If
End Sub

```

Listing 14.3: Anlegen eines neuen Ereignisprotokolls [eventlog.vb]

Ergebnis

Eintrag in der Registry

Das neue Ereignisprotokoll wird in die Registry unter `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Eventlog` eingetragen und ist dann in der MMC-Konsole »Computerverwaltung« sichtbar. Wenn Sie die Konsole bereits vor dem Anlegen geöffnet hatten, müssen Sie die Konsole schließen und wieder öffnen, da die Registry-Daten sonst nicht erneut eingelesen werden.

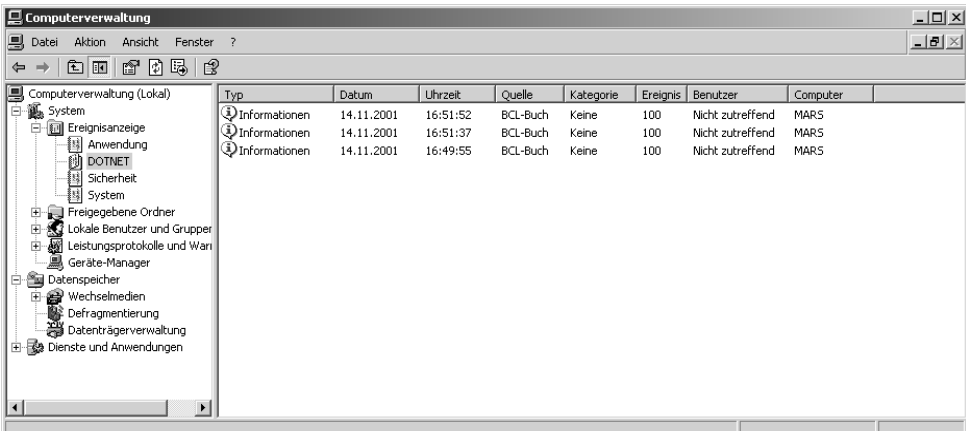


Abbildung 14.2: Die Abbildung zeigt das neue Ereignisprotokoll »DOTNET« in der MMC-Konsole »Computerverwaltung«.

14.1.5 Löschen eines Ereignisprotokolls

Delete() Das Löschen eines Ereignisprotokolls ist ein Zweizeiler:

- Instanziierung der Klasse `EventLog`
- Ausführung der Methode `Delete()`

Bei `Delete()` ist der erste Parameter (Protokollname) Pflicht, der zweite (Computername) optional.

Beispiel

Die folgende Routine löscht das Protokoll »DOTNET« auf dem Computer »MARS«.

```
' =====
' Beispiel aus FCL-Buch.sln
' Löschen eines Ereignisprotokolls
' (C) Holger@Schwichtenberg.de
' =====
Sub eventlog_dellog()
    Const LOGNAME = "DOTNET"
    Const COMPUTER = "Mars"
    Dim log As New EventLog()
    ' --- Protokoll löschen
    log.Delete(LOGNAME, COMPUTER)
    out("Log gelöscht!")
End Sub
```

Listing 14.4: Löschen eines Ereignisprotokolls [eventlog.vb]

14.1.6 Schreiben in das Ereignisprotokoll

Das Schreiben in das Ereignisprotokoll ist nach der Instanziierung eines `EventLog`-Objekts sehr einfach. Die Methode `WriteEntry()` erwartet folgende Parameter:

WriteEntry()

- ▶ Namen der Quelle (eine beliebige Zeichenkette)
- ▶ Inhalt des Eintrags (eine beliebige Zeichenkette)
- ▶ Typ des Eintrags (ein Wert aus der Enumeration `System.Diagnostics.EventLogEntryType`)
- ▶ eine Ereigniskennung (ein beliebiger Integer-Wert)

Beispiel

Das folgende Beispiel zeigt das Anlegen eines Eintrags in dem Protokoll »DOTNET«. Der Code-Block, der die Quelle anlegt, ist optional, da die Quelle automatisch angelegt wird, wenn sie nicht vorhanden ist.

Eintrag erzeugen

```
' =====
' Beispiel aus FCL-Buch.sln
' Erzeugung eines neuen Eintrags in
' einem Ereignisprotokoll
' (C) Holger@Schwichtenberg.de
' =====
Public Sub EventLog_Write()
```

```

Const logname As String = "DOTNET"
Const source As String = "FCL-Buch"
Dim log As New EventLog(logname)
' --- Dieser Block ist OPTIONAL ---
If (Not log.SourceExists(source)) Then
    log.CreateEventSource(source, logname)
    out("Quelle angelegt!")
End If
' ---- Eintrag schreiben
log.WriteEntry(source, "TestNachricht", _
System.Diagnostics.EventLogEntryType.Information, 100)
' --- Bildschirmausgabe
out("Eintrag geschrieben!")
End Sub

```

Listing 14.5: Erzeugung eines neuen Eintrags in einem Ereignisprotokoll [eventlog.vb]



Durch das Anlegen einer Quelle mit `CreateEventSource()` wird ein neuer Eintrag unterhalb von `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Eventlog\logname` erzeugt. Das explizite Anlegen der Quelle ist jedoch nicht notwendig: Wenn bei `Write Entry()` eine unbekannte Quelle angegeben wird, wird diese automatisch in die Registry eingetragen.

14.1.7 Überwachung eines Ereignisprotokolls

EventWritten() Die Klasse `EventLog` bietet auch ein Ereignis `EventWritten()`, das immer dann ausgelöst wird, wenn in das Ereignisprotokoll, das das `EventLog`-Objekt repräsentiert, ein neuer Eintrag geschrieben wird. Die Ereignisbehandlungsroutine bekommt über ein Objekt des Typs `EntryWrittenEventArgs` Informationen über den neuen Eintrag.

Beispiel

Das Beispiel zeigt die Überwachung des Protokolls »DOTNET«. Zwei neue Einträge werden mit der Routine `EventLog.Write()` aus dem vorherigen Kapitel erzeugt, nachdem mit `AddHandler` das Ereignis gebunden und mit `Thread.Sleep()` eine kurze Pause erzeugt wurde.

```

' =====
' Beispiel aus FCL-Buch.sln
' Überwachung eines Ereignisprotokolls
' (C) Holger@Schwichtenberg.de
' =====

```



```

Sub EventLog_Watcher()
    Dim Log As EventLog
    ' --- Instanziierung und Konfiguration
    Log = New EventLog()
    Log.Log = "DOTNET"
    Log.MachineName = "Mars"
    ' --- Verbindung der Ereignisbehandlungsroutine
    AddHandler Log.EntryWritten, _
    AddressOf EventLog_NeuerEintrag
    ' --- Starten der Überwachung
    Log.EnableRaisingEvents = True
    Thread.Sleep(5000)
    EventLog_Write()
    EventLog_Write()
    ' --- Warten auf Benutzeraktion
    MsgBox("Klicken Sie hier, um die Überwachung " & _
        "zu beenden!", , "Überwachung")
End Sub

' ### Ereignisbehandlungsroutine für
' Event-Log-Überwachung
Sub EventLog_NeuerEintrag(ByVal source As Object, _
    ByVal Ereignis As EntryWrittenEventArgs)

    Dim eintrag As EventLogEntry
    eintrag = Ereignis.Entry
    out(eintrag.EntryType & ":" & _
        eintrag.EventID & ":" & _
        eintrag.Category & ":" & _
        eintrag.Message & ":" & _
        eintrag.Source & ":" & _
        eintrag.TimeGenerated & ":" & _
        eintrag.TimeWritten & ":" & _
        eintrag.UserName)
End Sub

```

Listing 14.6: Überwachung eines Ereignisprotokolls [eventlog.vb]

14.2 Arbeit mit Prozessen

Der FCL-Namespace `System.Diagnostics` enthält auch Klassen, mit denen Prozesse verwaltet werden können. Insbesondere folgende Aufgaben können mit den Klassen ausgeführt werden:

**Prozesse und
Threads**

- Auflisten aller laufenden Prozesse
- Auflisten ausgewählter Prozesse

- ▶ Auflisten aller an einem Prozess beteiligten Dateien
- ▶ Auflisten aller Threads eines laufenden Prozesses
- ▶ Starten von Prozessen
- ▶ Beenden von Prozessen

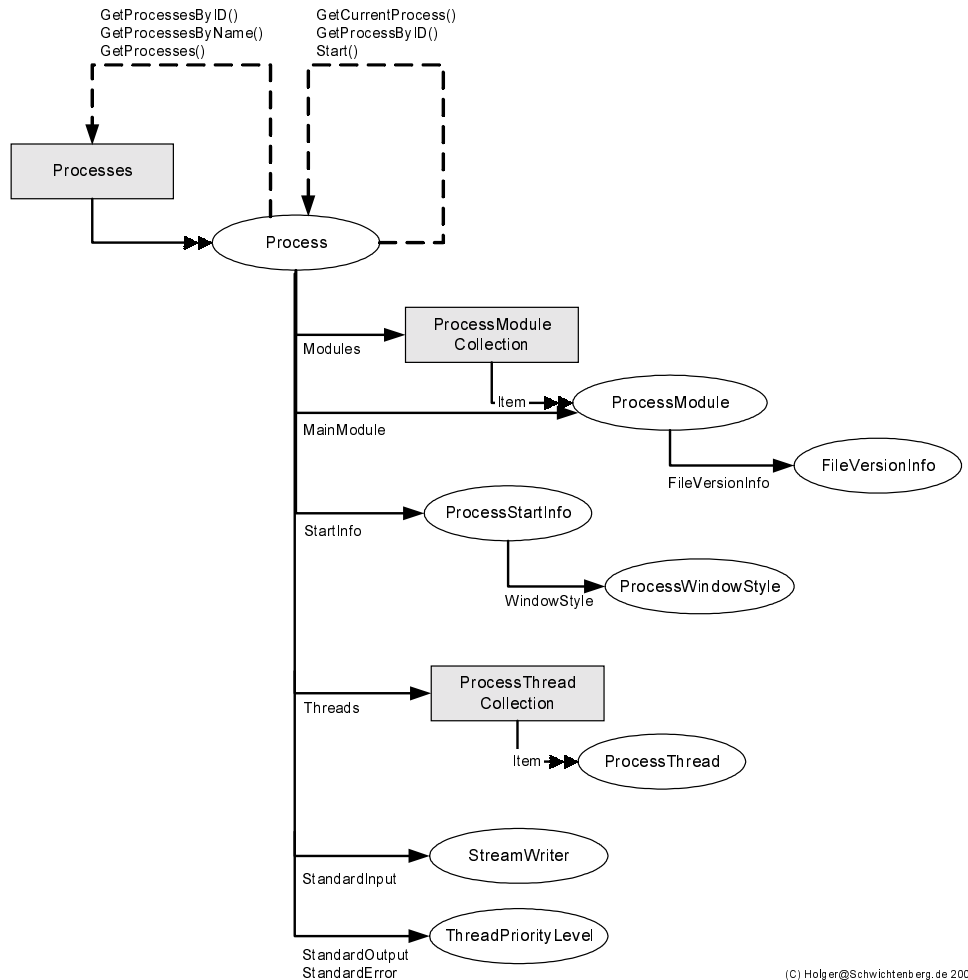
14.2.1 Überblick über die FCL-Klassen

Wichtige Klassen Wichtige FCL-Klassen für den Zugriff auf Prozesse sind:

- ▶ `Process`
Diese Klasse repräsentiert einen einzelnen laufenden Prozess und dient dazu, Prozesse zu starten und zu beenden.
- ▶ `ProcessModule`
Diese Klasse repräsentiert eine einzelne *.exe*- oder *.dll*-Datei, die im Rahmen des Prozesses verwendet wird.
- ▶ `FileVersionInfo`
Jedes `ProcessModule`-Objekt enthält einen Verweis auf ein `FileVersionInfo`-Objekt mit Informationen über die Dateiversion, den Hersteller, den Produktnamen etc.
- ▶ `ProcessModuleCollection`
Diese Collection enthält alle `ProcessModule`-Objekte eines Prozesses.
- ▶ `ProcessThread`
Diese Klasse repräsentiert einen einzelnen Thread eines Prozesses.
- ▶ `ProcessThreadCollection`
Diese Collection enthält alle `ProcessThread`-Objekte eines Prozesses.

Objektmodell

Die Abbildung 14.3 zeigt die Objekthierarchie der oben genannten Klassen.



(C) Holger@Schwichtenberg.de 2002

Abbildung 14.3: Objektmodell für die auf Prozesse und Threads bezogenen Klassen in `System.Diagnostics`

Zwei Sonderfälle sind zu beachten:

- Das **Process**-Objekt, das den Idle-Prozess (»Leerlaufprozess«) repräsentiert, besitzt weder eine `ProcessThreadCollection` noch eine `ProcessModuleCollection`. Diese Collections sind also für diesen spe-



ziellen Prozess nicht leer, sondern nicht existent. Der Zugriff auf die Attribute `Threads` und `Modules` führt zu einer Exception. Der Idle-Prozess ist leicht an der Prozess-ID »0« zu erkennen.

- Das `Process`-Objekt, das den System-Prozess (Prozess-ID »4«) repräsentiert, besitzt sowohl eine `ProcessThreadCollection` als auch eine `ProcessModuleCollection`. Die `ProcessModuleCollection` ist jedoch immer leer und das Attribut `MainModule` verweist auf `Nothing`.

14.2.2 Liste der Prozesse

Klasse »Process« Die Klasse `Process` bietet eine statische Methode mit Namen `GetProcess()`, die ein Array von `Process`-Objekten mit den aktuell laufenden Prozessen zurückliefert.

Beispiel

Der folgende Code listet alle laufenden Prozesse auf.

```
' =====
' Beispiel aus FCL-Buch.sln
' Liste aller laufenden Prozesse
' (C) Holger@Schwichtenberg.de
' =====
Sub Prozesse_Liste()
    Dim p As Process ' einzelner Prozess
    Dim pp() As Process ' Liste der Prozesse

    ' --- Liste der Prozesse holen
    pp = Process.GetProcesses

    ' --- Schleife über alle Prozesse
    For Each p In pp
        out("-----")
        out(p.Id & ":" & p.ProcessName)
        out("-----")
        If p.Id <> 0 Then
            ' Ausnahme für "Idle" (ID=0) !
            out("Handle Count      :" & _
                p.HandleCount)
            out("Main Window Title    :" & _
                p.MainWindowTitle)
            out("Min Working Set       :" & _
                p.MinWorkingSet.ToString)
            out("Max Working Set       :" & _
                p.MaxWorkingSet.ToString)
            out("Paged Memory Size     :" & _
                p.PagedMemorySize)
```

```

        out("Peak Paged Memory Size :" & _
            p.PeakPagedMemorySize)
    End If
Next
End Sub

```

Listing 14.7: Liste aller laufenden Prozesse [prozesse.vb]

Ausgabe

Das folgende Listing zeigt beispielhaft einen Teil der Ausgabe obiger Routine. Die Ausgabe ist normalerweise sehr lang, da pro Prozess neun Zeilen ausgegeben werden.

Bitte beachten Sie, dass der Name eines Prozesses dem Namen des Hauptmoduls der Datei ohne dessen Dateiextension entspricht.



```

1  -----
2  424:smss
3  -----
4  Handle Count           :21
5  Main Window Title      :
6  Min Working Set        :204800
7  Max Working Set        :1413120
8  Paged Memory Size      :167936
9  Peak Paged Memory Size :1974272
10 -----
11 1276:OUTLOOK
12 -----
13 Handle Count           :435
14 Main Window Title      :Postausgang - Microsoft Outlook
15 Min Working Set        :204800
16 Max Working Set        :1413120
17 Paged Memory Size      :7917568
18 Peak Paged Memory Size :8499200
19 -----
20 2584:IEXPLORE
21 -----
22 Handle Count           :293
23 Main Window Title      :Yahoo! Groups - Microsoft Internet
Explorer
24 Min Working Set        :204800
25 Max Working Set        :1413120
26 Paged Memory Size      :4583424
27 Peak Paged Memory Size :4804608
...

```

Selektion von Prozessen

GetProcessesByName()

Die Klasse `Process` besitzt mit `GetProcessesByName()` eine Methode zum Selektieren der Prozesse anhand ihres Namens. Im Namen darf dabei die Dateierweiterung nicht angegeben werden (also »IExplore« statt »IExplore.exe«).

Beispiel

Das folgende Beispiel listet alle laufenden Instanzen des Internet Explorers (Dateiname: *iexplorer.exe*, Prozessname: *iexplore*) auf. Zu jeder Instanz werden die Prozess-ID und die Startzeit ausgegeben.

```
' =====
' Beispiel aus FCL-Buch.sln
' Liste ausgewählter Prozesse
' (C) Holger@Schwichtenberg.de
' =====
Sub Prozesse_Liste_Selection()
    Dim p As Process ' einzelner Prozess
    Dim pp() As Process ' Liste der Prozesse

    Const PROZESSNAME = "IEXPLORE"

    ' --- Liste der Prozesse holen
    pp = Process.GetProcessesByName(PROZESSNAME)

    ' --- Ausgaben
    out("Prozesse mit Namen: " & PROZESSNAME)
    out("Anzahl von Prozessen: " & pp.Length)
    ' --- Schleife über alle Instanzen
    For Each p In pp
        out(p.Id & ":" & p.StartTime)
    Next
End Sub
```

Listing 14.8: Liste ausgewählter Prozesse [prozesse.vb]

Ausgabe

Das folgende Listing zeigt die Ausgabe obiger Routine, wenn vier Internet Explorer-Fenster geöffnet sind.

```
1 Prozesse mit Namen: IEXPLORE
2 Anzahl von Prozessen: 4
3 2348:18.11.2001 17:09:01
4 3488:18.11.2001 17:14:14
5 2156:18.11.2001 17:09:06
6 3116:18.11.2001 17:14:14
```

14.2.3 Liste der Module

Im Process-Objekt liefert das Attribut `Modules` eine `ProcessModule Collection` mit einzelnen `ProcessModule`-Objekten mit den `.exe`- und `.dll`-Dateien, die an dem Prozess beteiligt sind.

**ProcessModule
Collection**

Beispiel

Im folgenden Beispiel werden alle laufenden Prozesse mit ihren Modulen aufgelistet. Dabei werden zu jedem Modul der Name, der Pfad und die Speichernutzung ausgegeben.

**Prozesse und
Module**

Für den Prozess mit der ID »0«, den Idle-Prozess, wird abgefangen, dass er keine Module und das zugehörige Process-Objekt keine `ProcessModule Collection` besitzt.



```
' =====
' Beispiel aus FCL-Buch.sln
' Liste aller Prozesse mit ihren Modulen
' (C) Holger@Schwichtenberg.de
' =====
Sub Prozesse_ModulListe()
    Dim p As Process ' einzelner Prozess
    Dim pp() As Process ' Liste der Prozesse
    Dim m As ProcessModule ' ein Modul

    ' --- Liste der Prozesse holen
    pp = Process.GetProcesses

    ' --- Schleife über alle Prozesse
    For Each p In pp
        out("-----")
        out(p.Id & ":" & p.ProcessName)
        out("-----")
        If p.Id <> 0 Then ' Ausnahme für "Idle" (ID=0)
            out("Anzahl Module: " & p.Modules.Count)
            If Not p.MainModule Is Nothing Then
                out("Hauptmodul: " & _
                    p.MainModule.ModuleName)
            End If
            ' --- Schleife über alle Module
            For Each m In p.Modules
                out("- " & m.ModuleName & ":" & _
                    m.FileName & ":" & _
                    m.ModuleMemorySize)
            Next
        End If
    Next
End Sub
```

Listing 14.9: Liste aller Prozesse mit ihren Modulen [prozesse.vb]

Ausgabe

Die Modullisten vieler Prozesse sind sehr lang. Eine einfache .NET-Anwendung benötigt ca. 100 Dateien. Daher ist die Ausgabe hier nur Ausschnittsweise dargestellt.

```

1  -----
2  3368:FCL-Buch
3  -----
4  Anzahl Module: 104
5  Hauptmodul: FCL-Buch.exe
6  - FCL-Buch.exe:H:\Code\BUCH_FCL\bin\FCL-Buch.exe:147456
7  - ntdll.dll:C:\WINDOWS\System32\ntdll.dll:720896
8  - mscoree.dll:C:\WINDOWS\System32\mscoree.dll:135168
9  - KERNEL32.dll:C:\WINDOWS\system32\KERNEL32.dll:1011712
10 - ADVAPI32.dll:C:\WINDOWS\system32\ADVAPI32.dll:630784
11 - RPCRT4.dll:C:\WINDOWS\system32\RPCRT4.dll:479232
12 - mscorwks.dll:C:\WINDOWS\Microsoft.NET\
    Framework\v1.0.3328\mscorwks.dll:2277376
13 - USER32.dll:C:\WINDOWS\system32\USER32.dll:577536
14 - GDI32.dll:C:\WINDOWS\system32\GDI32.dll:262144
15 - MSVCR70.dll:C:\WINDOWS\Microsoft.NET\Framework\
    v1.0.3328\MSVCR70.dll:344064
16 - fusion.dll:C:\WINDOWS\Microsoft.NET\Framework\
    v1.0.3328\fusion.dll:212992
17 - SHLWAPI.dll:C:\WINDOWS\system32\SHLWAPI.dll:405504
18 - msvcrt.dll:C:\WINDOWS\system32\
    msvcrt.dll:339968
19 - mscorlib.dll:c:\windows\microsoft.net\
    framework\v1.0.3328\mscorlib.dll:1957888
20 - ole32.dll:C:\WINDOWS\system32\ole32.dll:1155072
21 - diasymreader.dll:C:\WINDOWS\Microsoft.NET\
    Framework\v1.0.3328\diasymreader.dll:434176
22 - mscorsn.dll:C:\WINDOWS\Microsoft.NET\Framework\
    v1.0.3328\mscorsn.dll:65536
23 - MSCTF.dll:C:\WINDOWS\System32\MSCTF.dll:307200
24 - MSCORJIT.DLL:C:\WINDOWS\Microsoft.NET\
    Framework\v1.0.3328\MSCORJIT.DLL:299008
25 - shell32.dll:C:\WINDOWS\system32\
    shell32.dll:8380416
26 - comctl32.dll:C:\WINDOWS\WinSxS\
    x86_Microsoft.Windows.Common-
    Controls_6595b64144ccf1df_6.0.0.0_x-
    ww_1382d70a\comctl32.dll:933888
27 - comctl32.dll:C:\WINDOWS\system32\
    comctl32.dll:569344
28 - system.windows.forms.dll:c:\windows\assembly\
    gac\system.windows.forms\1.0.3300.0__b77a5c561934e089\
    system.windows.forms.dll:1990656
...

```


14.2.4 Liste der Threads

In einem `Process`-Objekt liefert das Attribut `Threads` eine `Process ThreadCollection` mit einzelnen `ProcessThread`-Objekten mit den Threads eines Prozesses.

Beispiel

In dem folgenden Beispiel werden alle laufenden Prozesse mit ihren Threads aufgelistet. Dabei werden zu jedem Thread die Identifikationsnummer, der Status und die Startzeit ausgegeben.

Für den Prozess mit der ID »0«, den Idle-Prozess, wird abgefangen, dass er keine Threads und das zugehörige `Process`-Objekt keine `Process ThreadCollection` besitzt.



```
' =====
' Beispiel aus FCL-Buch.sln
' Liste aller Prozesse mit ihren Threads
' (C) Holger@Schwichtenberg.de
' =====
Sub Prozesse_ThreadListe()
    Dim p As Process ' einzelner Prozess
    Dim pp() As Process ' Liste der Prozesse
    Dim t As ProcessThread ' ein Thread

    ' --- Liste der Prozesse holen
    pp = Process.GetProcesses

    ' --- Schleife über alle Prozesse
    For Each p In pp
        out("-----")
        out(p.Id & ":" & p.ProcessName)
        out("-----")
        If p.Id <> 0 Then ' Ausnahme für "Idle"(PID=0)!
            ' --- Schleife über alle Threads
            out("Anzahl Threads: " & p.Threads.Count)
            For Each t In p.Threads
                out("- " & t.Id & ":" & _
                    t.ThreadState & ":" & _
                    t.StartTime)
            Next
        End If
    Next
End Sub
```

Listing 14.10: Liste aller Prozesse mit ihren Threads [prozesse.vb]

Ausgabe

Das folgende Listing zeigt beispielhaft einen Teil der Ausgabe der obigen Routine.

```

1  -----
2  1984:wmiprvse
3  -----
4  Anzahl Threads: 6
5  - 4080:5:18.11.2001 16:59:13
6  - 764:5:18.11.2001 16:59:13
7  - 2936:5:18.11.2001 16:59:13
8  - 2876:5:18.11.2001 16:59:13
9  - 3916:5:18.11.2001 16:59:13
10 - 864:5:18.11.2001 16:59:13
...
52 -----
53 904:OUTLOOK
54 -----
55 Anzahl Threads: 12
56 - 2008:5:18.11.2001 16:59:34
57 - 2836:5:18.11.2001 16:59:35
58 - 3728:5:18.11.2001 16:59:35
59 - 2980:5:18.11.2001 16:59:35
60 - 2204:5:18.11.2001 16:59:35
61 - 224:5:18.11.2001 16:59:35
62 - 896:5:18.11.2001 16:59:36
63 - 4092:5:18.11.2001 16:59:36
64 - 3816:5:18.11.2001 16:59:36
65 - 3244:5:18.11.2001 16:59:36
66 - 3764:5:18.11.2001 16:59:36
67 - 412:5:18.11.2001 16:59:37
...

```

14.2.5 Prozesse starten

Start() Um einen Prozess zu starten, müssen Sie lediglich die Methode `Start()` in der Klasse `Process` aufrufen. Die Methode erwartet im ersten Parameter den Namen der zu startenden Anwendung und im zweiten Parameter die Liste der Parameter für die zu startende Anwendung.

Beispiel

Das Beispiel zum Starten des Internet Explorers mit einem bestimmten URL ist ein Zweizeiler.

```

' =====
' Beispiel aus FCL-Buch.sln
' Start eines Prozesses
' (C) Holger@Schwichtenberg.de
' =====
Sub Prozess_Starten()
' --- Instanz erzeugen
Dim p As New System.Diagnostics.Process()
' --- Prozess starten
p.Start("IExplore.exe", _
"http://www.IT-Visions.de")
End Sub

```

Listing 14.11: Start eines Prozesses [prozesse.vb]

14.2.6 Prozesse beenden

Die Klasse `Process` bietet zwei verschiedene Methoden zum Beenden von Prozessen.

- Die sanfte Methode ist `CloseMainWindow()`. Diese hat den gleichen Effekt wie der Aufruf von DATEI/BEENDEN im Hauptfenster einer Windows-Anwendung durch den Benutzer. Wenn Änderungen an einem Dokument vorgenommen wurden, fragt die Anwendung den Benutzer zunächst, ob die Änderungen gespeichert werden sollen. Der Benutzer kann speichern oder den Beendigungs-Vorgang abbrechen. Die Methode liefert `True` zurück, wenn der Beendigungs-Vorgang angestoßen werden konnte. Ein `True` sagt aber nichts darüber aus, ob die Anwendung auch tatsächlich geschlossen wurde. `False` bedeutet, die Anwendung kann derzeit nicht normal beendet werden, z. B. weil ein Dialogfenster geöffnet ist.
- Im Fall von `Kill()` hat der Benutzer keinen Einfluss: Der Prozess wird sofort ohne Nachfrage beendet. Änderungen an offenen Dokumenten werden nicht gespeichert.

**CloseMain
Window()**

Kill()

Die Klasse `Process` kann nur lokale Prozesse beenden. Es ist hiermit nicht möglich, Prozesse auf entfernten Computern zu beenden. Dies kann mit der WMI-Klasse `Win32_Process` erfolgen.



Beispiel

Das folgende Beispiel beendet alle laufenden Instanzen des Internet Explorers (Dateiname: `iexplorer.exe`, Prozessname: `iexplore`). Zunächst wird versucht, den Prozess mit `CloseMainWindow()` zu beenden. Wenn das nicht funktioniert, wird `Kill()` aufgerufen.

**Internet Explorer
beenden**

```

' =====
' Beispiel aus FCL-Buch.sln
' Alle Internet Explorer-Prozesse stoppen
' (C) Holger@Schwichtenberg.de
' =====
Sub Prozesse_Stoppen()
    Const PNAME As String = "IEXPLORE"
    Dim p As System.Diagnostics.Process
    Dim pp As System.Diagnostics.Process()

    ' --- Prozesse dieses Namens ermitteln
    pp = Process.GetProcessesByName(PNAME)
    ' --- Schleife über diese Prozesse
    out("Beenden aller Prozesse mit Namen: " & PNAME)
    out("Anzahl Prozesse: " & pp.Length)
    For Each p In pp
        out("Prozess: " & p.Id & " wird beendet...")
        ' --- Beenden-Anfrage stellen...
        If p.CloseMainWindow Then
            ' --- Normales Ende
            out(" Prozess wurde normal beendet!")
        Else
            ' --- Keine Reaktion -> gewaltsames Ende
            p.Kill()
            out(" Prozess wurde gewaltsam beendet!")
        End If
    Next
End Sub

```

Listing 14.12: Alle Internet Explorer-Prozesse stoppen [prozesse.vb]

Ausgabe

Das folgende Listing zeigt beispielhaft einen Teil der Ausgabe obiger Routine, wenn vier Internet Explorer-Fenster geöffnet sind. Einer der vier Prozesse musste gewaltsam mit `Kill()` beendet werden, weil zum Zeitpunkt des Aufrufs von `CloseMainWindow()` das Dialogfenster DATEI/ÖFFNEN angezeigt wurde.

```

1 Beenden aller Prozesse mit Namen: IEXPLORE
2 Anzahl Prozesse: 4
3 Prozess: 1812 wird beendet...
4 Prozess wurde normal beendet!
5 Prozess: 3512 wird beendet...
6 Prozess wurde gewaltsam beendet!
7 Prozess: 3272 wird beendet...
8 Prozess wurde normal beendet!
9 Prozess: 2900 wird beendet...
10 Prozess wurde normal beendet!

```

14.2.7 Nicht reagierende Prozesse beenden

Mit Hilfe des Attributs `Responding` in der Klasse `Process` kann geprüft werden, ob ein Prozess noch reagiert. Die folgende Routine beendet alle nicht reagierenden Prozesse.

```
' =====
' Beispiel aus FCL-Buch.sln
' Alle hängenden Prozesse stoppen
' (C) Holger@Schwichtenberg.de
' =====
Sub Prozesse_ohne_Reaktion_stoppen()

    Dim p As System.Diagnostics.Process
    Dim pp As System.Diagnostics.Process()

    ' --- Prozesse dieses Namens ermitteln
    pp = Process.GetProcesses()
    ' --- Schleife über diese Prozesse
    out("Beenden aller nicht-reagierenden Prozesse")

    For Each p In pp
        If Not p.Responding Then
            out("Prozess hängt " & p.ProcessName & _
                " und wird beendet...")
            ' --- Keine Reaktion -> gewaltsames Ende
            p.Kill()
            out(" Prozess wurde gewaltsam beendet!")
        End If
    Next
End Sub
```

Listing 14.13: Alle hängenden Prozesse stoppen [prozesse.vb]

14.3 Informationen über Programmdateien

Die Klasse `FileVersionInfo` liefert Versions- und Ursprungsinformationen über eine Programmdatei (`.dll` oder `.exe`).

Beispiel

Das Beispiel gibt Informationen über die Datei `scrrun.dll`, die die COM-Komponente Scripting Runtime Library enthält. Diese Komponente wird mit vielen Produkten (z.B. Windows Script Host, Internet Explorer) ausgeliefert und existiert daher in vielen verschiedenen Versionen.

```

' =====
' Beispiel aus FCL-Buch.sln
' Informationen über eine Programmdatei
' (C) Holger@Schwichtenberg.de
' =====
Public Sub Datei_Info()

    Dim Vers As FileVersionInfo

    Vers = FileVersionInfo.GetVersionInfo _
        ("C:\Windows\system32\scrrun.dll")

    out("Info für Datei: " & Vers.FileName)

    out("Beschreibung; " & Vers.FileDescription)
    out("Version: " & Vers.FileVersion)
    out("Hersteller: " & Vers.CompanyName)
    out("Sprache: " & Vers.Language)
    out("Produktname: " & Vers.ProductName)
    out("Interner Name: " & Vers.InternalName)
    out("Kommentar: " & Vers.Comments())

End Sub

```

Listing 14.14: Informationen über eine Programmdatei [dateien.vb]

Ausgabe

Das folgende Listing zeigt beispielhaft einen Teil der Ausgabe der obigen Routine.

```

1 Info für Datei: C:\Windows\system32\scrrun.dll
2 Beschreibung; Microsoft (r) Script Runtime
3 Version: 5.6.0.6626
4 Hersteller: Microsoft Corporation
5 Sprache: Englisch (USA)
6 Produktname: Microsoft (r) Script Runtime
7 Interner Name:scrrun.dll
8 Kommentar:

```

14.4 Arbeit mit Leistungsindikatoren

Im Namespace `System.Diagnostics` gibt es mehrere Klassen für den Zugriff auf Windows-Leistungsindikatoren (engl. Performance Counter).

14.4.1 Grundlagen

Dieses Unterkapitel enthält einen sehr kurzen Exkurs zu den Grundlagen der Arbeit mit Leistungsindikatoren, deren Kenntnis für die folgenden Beispiele hilfreich ist.

**Performance
Counter**

Identifizierung von Leistungsindikatoren

Ein Leistungsindikator wird anhand von vier Daten eindeutig identifiziert:

- ▶ Computername
- ▶ Kategorie (alias »Datenobjekt«)
- ▶ Objektname
- ▶ Instanzname

Der eindeutige Name wird dabei in der Form

`\\Computername\Kategorie(Instanzname)\Objektname`
gebildet.

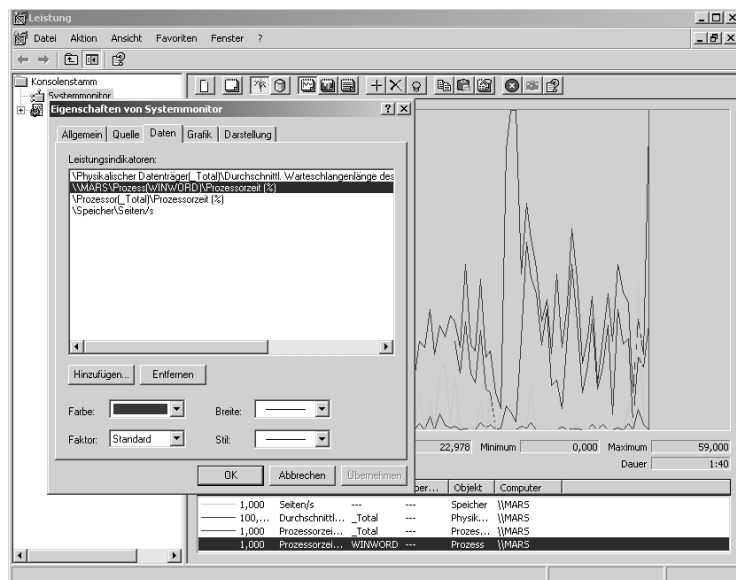


Abbildung 14.4: Anzeige von Leistungsindikatoren im Systemmonitor von Windows XP

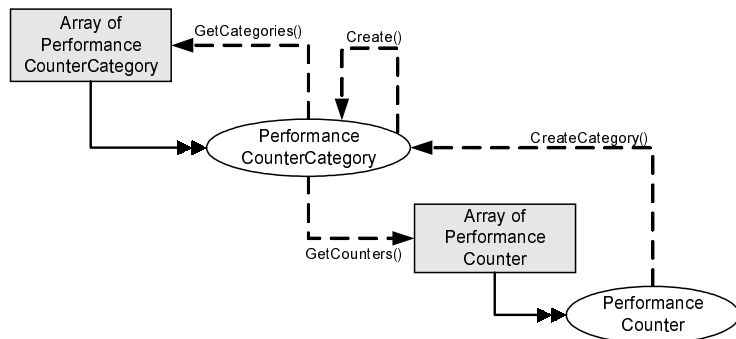
14.4.2 Überblick über die FCL-Klassen

Wichtige Klassen Wichtige FCL-Klassen für den Zugriff auf Leistungsindikatoren sind:

- ▶ `PerformanceCounter`
Diese Klasse dient sowohl dem Lesen von Leistungsindikatoren als auch dem Schreiben von Daten in Leistungsindikatoren.
- ▶ `PerformanceCounterCategory`
Diese Klasse dient dem Anlegen und Löschen von Kategorien und Indikatoren.
- ▶ `CounterCreationData`
Diese Klasse dient als Hilfsobjekt zum Anlegen eines Leistungsindikators.
- ▶ `CounterCreationDataCollection`
Diese Klasse nimmt beim Anlegen einer neuen Kategorie von Leistungsindikatoren eine Menge von `CounterCreationData`-Objekten auf.

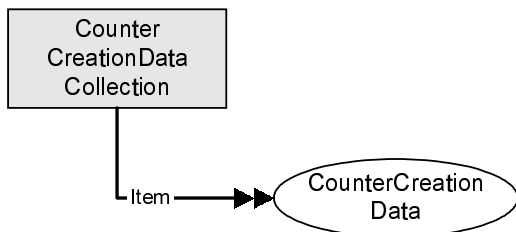
Objekthierarchie

Die folgenden Grafiken zeigen die Objekthierarchie dieser Klassen.



(C) Holger@Schwichtenberg.de 2002

Abbildung 14.5: Objektmodell für die auf Leistungsindikatoren bezogenen Klassen in *System.Diagnostics* (Teil 1)



(C) Holger@Schwichtenberg.de 2002

Abbildung 14.6: Objektmodell für die auf Leistungsindikatoren bezogenen Klassen in System.Diagnostics (Teil 2)

Weitere Klassen

Darüber hinaus gibt es Klassen für die Installation von Leistungsindikatoren und für die Änderung von Rechten auf Leistungsindikatoren. Diese werden hier nicht besprochen.

14.4.3 Lesen eines Leistungsindikators

Zum Lesen eines Indikators wird eine Instanz der Klasse `PerformanceCounter` benötigt. Beim Instanzieren werden der Name der Kategorie und der Name des Indikators erwartet. Optional sind der Name der Instanz (3. Parameter) und der Name des Computers (4. Parameter).

**Performance
Counter**

```
PC = New PerformanceCounter("KATEGORIE", "OBJEKTNAME",
    "INSTANZNAME", "COMPUTER")
```

Auslesen von Werten

Danach kann mit der Methode `NextValue()` jeweils der nächste Wert ermittelt werden. Alternativ bekommt man mit `NextSample()` eine `CounterSample`-Struktur. Diese enthält die Daten des Indikators in Rohform.

NextValue()

Beispiel

Das folgende Beispiel gibt die nächsten 100 Werte der Indikator-Instanz

```
Prozessor(_Total)\ Prozessorzeit (%)
```

aus.

```

' =====
' Beispiel aus FCL-Buch.sln
' Performance Counter lesen
' (C) Holger@Schwichtenberg.de
' =====

Sub PerfCount_Lesen()

    Const KATEGORIE As String = "Prozessor"
    Const OBJEKTNAME As String = "Prozessorzeit (%)"
    Const INSTANZNAME As String = "_Total"
    Const COMPUTER = "Sonne2000"

    Dim PC As PerformanceCounter
    Dim a As Integer
    Dim wert As Long

    ' --- Instanziieren
    PC = New PerformanceCounter(KATEGORIE, OBJEKTNAME, _
        INSTANZNAME, COMPUTER)

    ' --- Nächste 100 Werte lesen
    For a = 1 To 100
        ' --- Wert auslesen
        wert = PC.NextValue().ToString()
        ' --- Wert ausgeben
        out("(" & _
            PC.InstanceName() & ")/" & _
            PC.CounterName & " = " & wert)
    Next

End Sub

```

Listing 14.15: Performance Counter lesen [performance.vb]

14.4.4 Anlegen und Löschen eines Leistungsindikators

Performance-CounterCategory

Die Klasse `PerformanceCounterCategory` erlaubt das Anlegen einer neuen Leistungsindikator-Kategorie mit einem oder mehreren Leistungsindikatoren.

Dazu sind folgende Schritte durchzuführen:

- Für jeden Leistungsindikator ist zunächst ein Objekt vom Typ `CounterCreationData` anzulegen. Dort sind der Name und der Typ des Counters und optional eine Beschreibung zu hinterlegen.
- Nachdem dieser Schritt für jeden einzelnen Leistungsindikator durchgeführt wurde, sind alle `CounterCreationData`-Objekte zu einer `CounterCreationDataCollection` zusammenzufügen.

- Mit Hilfe der statischen Methode `Create()` in der Klasse `PerformanceCounterCategory` kann dann eine neue Kategorie mit den zuvor definierten Leistungsindikatoren angelegt werden. `Create()` erwartet einen Namen, eine Beschreibung und ein Objekt vom Typ `CounterCreationDataCollection`.

Eine Leistungsindikator-Kategorie kann nur angelegt werden, wenn sie noch nicht existiert. Das spätere Hinzufügen oder Löschen von einzelnen Leistungsindikatoren ist nicht direkt möglich. Dazu muss zunächst die Kategorie mit allen Indikatoren gelöscht werden.

Instanzen werden erst erzeugt, wenn Daten in den Indikator geschrieben werden. Instanzen können ohne Einfluss auf andere Indikatoren oder die Kategorie gelöscht werden.

Löschen eines Leistungsindikators

Wenn ein Leistungsindikator gelöscht werden soll, muss zunächst die Kategorie gelöscht werden. In der Klasse `PerformanceCounterCategory` kann mit der `Exists()`-Methode geprüft werden, ob eine Kategorie existiert, und mit der `Delete()`-Methode kann eine Kategorie gelöscht werden. In beiden Fällen wird nur der Kategoriename erwartet.

Delete()

Man kann nur benutzerdefinierte Kategorien löschen. Die vordefinierten Kategorien können nicht entfernt werden.



Beispiel

Das folgende Beispiel legt zwei Leistungsindikatoren »Seitenabfrage pro Sekunde« und »Aktuelle Benutzeranzahl« in der Kategorie »Website IT-Visions.de« an. Wenn die Kategorie existiert, wird sie vor dem Anlegen gelöscht.

```
' =====
' Beispiel aus FCL-Buch.sln
' Anlegen von Leistungsindikatoren
' (C) Holger@Schwichtenberg.de
' =====
Public Sub PerfCount_anlegen()
    Const KATEGORIE = "Website IT-Visions.de"
    Const OBJEKTNAME1 As String = _
        "Seitenabfrage pro Sekunde"
    Const OBJEKTNAME2 As String = _
        "Aktuelle Benutzeranzahl"
```

```

' --- Counter definieren
Dim Counter1 As New CounterCreationData()
Counter1.CounterName = OBJEKTNAME1
Counter1.CounterHelp = "Angelegt von HS"
Counter1.CounterType = _
    PerformanceCounterType.CountPerTimeInterval32

Dim Counter2 As New CounterCreationData()
Counter2.CounterName = OBJEKTNAME2
Counter2.CounterHelp = "Angelegt von HS"
Counter2.CounterType = _
    PerformanceCounterType.NumberOfItems64

' --- Liste der neuen Counter zusammenstellen
Dim neuecounter As New CounterCreationDataCollection()
neuecounter.Add(Counter1)
neuecounter.Add(counter2)

Dim PCCat As New PerformanceCounterCategory(KATEGORIE)

' Kategorie löschen, wenn sie bereits vorhanden ist
If PCCat.Exists(KATEGORIE) Then
    PCCat.Delete(KATEGORIE)
    out("Leistungsindikatorekategorie " & KATEGORIE & _
        " gelöscht!")
End If

' --- Anlegen
PCCat = PCCat.Create(KATEGORIE, _
    "Beispiel aus FCL-Buch!", neuecounter)
out("Leistungsindikatorekategorie " & _
    KATEGORIE & " angelegt!")
End Sub

```

Listing 14.16: Anlegen von Leistungsindikatoren [performance.vb]

Ergebnis des Beispiels

Wie die folgende Abbildung aus dem Windows 2000-Systemmonitor zeigt, existieren nach dem Anlegen eines Leistungsindikators noch keine Instanzen. Instanzen werden erst beim Schreiben von Werten angelegt.

14.4.5 Schreiben von Werten in einen Leistungsindikator

Die Klasse `PerformanceCounter` ermöglicht auch auf einfache Weise, Werte in benutzerdefinierte Indikatoren zu schreiben. Dabei können auch verschiedene Instanzen erzeugt werden.

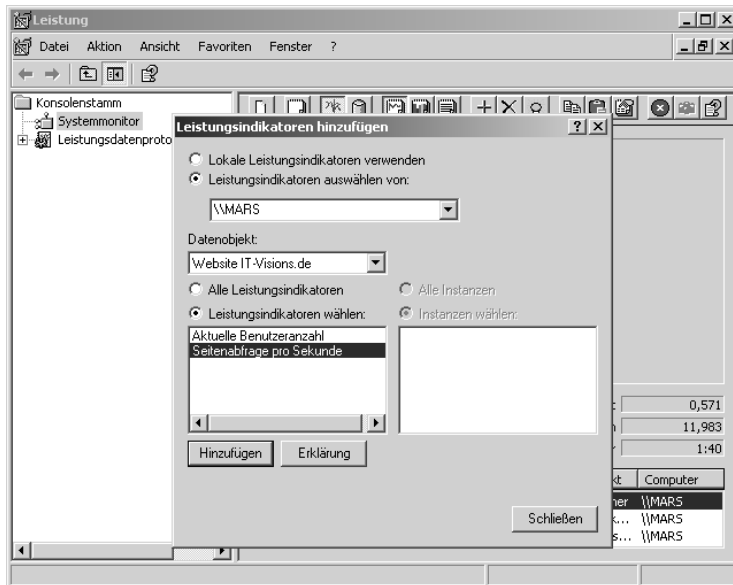


Abbildung 14.7: Anzeige der neuen Leistungsindikatoren unter »Hinzufügen« in der MMC-Konsole »Systemmonitor«

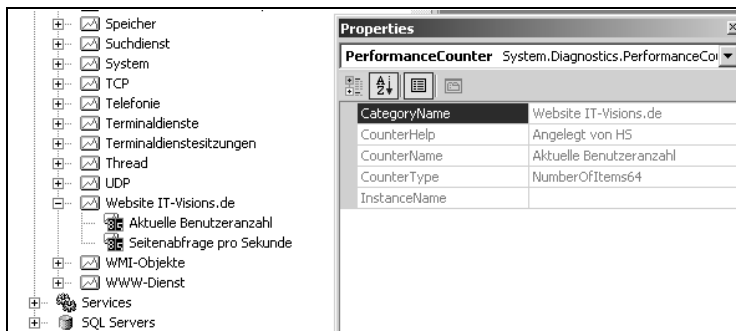


Abbildung 14.8: Anzeige der neuen Leistungsindikatoren im »Server Explorer« in Visual Studio .NET

Instanz anlegen

Um eine Instanz zu erzeugen, wird bei der Instanziierung der `PerformanceCounter`-Klasse einfach ein dritter Parameter angegeben. Wenn der Parameter fehlt oder mit einem Leerstring belegt ist, werden die Daten direkt in den Indikator geschrieben.

Instanzname als dritter Parameter

```
Dim PC As New
PerformanceCounter("KATEGORIE", "OBJEKTNAME", "INSTANZ")
```

Werte schreiben

Zähler verändern Im Standardfall ermöglicht ein `PerformanceCounter`-Objekt nur den Lesezugriff. Der Schreibzugriff muss erst mit Setzen des Attributs `ReadOnly` auf `False` aktiviert werden. Danach können die Werte mit folgenden Attributen und Methoden verändert werden:

- ▶ Das Attribut `RawValue` ermöglicht es, den Indikator auf einen bestimmten Wert zu setzen.
- ▶ Die Methode `Increment()` erhöht den Wert um eins.
- ▶ Die Methode `IncrementBy(ZAHL)` erhöht den Wert um eine bestimmte Zahl.
- ▶ Die Methode `Decrement()` vermindert den Wert um eins. Eine Methode, um den Wert direkt um mehr als eins zu vermindern, gibt es nicht.

Löschen einer Instanz

RemoveInstance() Eine Instanz wird durch Ausführung der Methode `RemoveInstance()` gelöscht, ohne dass dies Auswirkungen auf andere Instanzen, den zugehörigen Indikator oder dessen Kategorie hätte.

```
PC.RemoveInstance()
```

Beispiel

Das folgende Beispiel verändert die Indikator-Instanz

`\\Mars\Website IT-Visions.de(Geschützte Seiten für Leser)\Aktuelle Benutzeranzahl.`

Zunächst wird der Wert auf einen Startwert von 50 gesetzt, dann in fünf Schritten um jeweils eins erhöht. Nach einer Erhöhung um 20 erfolgt dann das schrittweise Runterzählen um fünf. Die Dialogboxen geben Ihnen die Möglichkeit, die Veränderungen im Systemmonitor zu verfolgen.

```
' --- Werte In Performance Counter schreiben
Sub PerfCount_Schreiben()
    Const KATEGORIE = "Website IT-Visions"
    Const OBJEKTNAME2 = "Aktuelle Benutzeranzahl"
    Const INSTANZ1 = "Geschützte Seiten für Leser"
    Const INSTANZ2 = "Chat"

    Dim a As Integer
```

```
' --- Zugriff auf Instanz
Dim PC As New PerformanceCounter(KATEGORIE, _
    OBJEKTNAME2, INSTANZ1)
' --- Schreibschutz aufheben
PC.ReadOnly = False

' --- Startwert vorgeben!
PC.RawValue = 50
MsgBox("Anzahl der Benutzer: " & PC.RawValue & _
    Chr(13) & "Betrachten Sie die Veränderung " & _
    "im Systemmonitor!")

' --- Wert verändern
For a = 1 To 5
    PC.Increment()
    MsgBox("Anzahl der Benutzer: " & PC.RawValue & _
        Chr(13) & "Betrachten Sie die Veränderung " & _
        "im Systemmonitor!")
Next

' --- Wert weiter erhöhen
PC.IncrementBy(20)
MsgBox("Anzahl der Benutzer: " & PC.RawValue & _
    Chr(13) & "Betrachten Sie die Veränderung " & _
    "im Systemmonitor!")

' --- Wert verändern
For a = 1 To 5
    PC.Decrement()
    MsgBox("Anzahl der Benutzer: " & PC.RawValue & _
        Chr(13) & "Betrachten Sie die Veränderung " & _
        "im Systemmonitor!")
Next

' --- Entfernt die Instanz
PC.RemoveInstance()
End Sub
```

Listing 14.17: Werte in Performance Counter schreiben [performance.vb]

Ergebnis des Beispiels

Die folgende Grafik zeigt die Veränderung der Kurve im Systemmonitor beim Ablauf der obigen Routine.

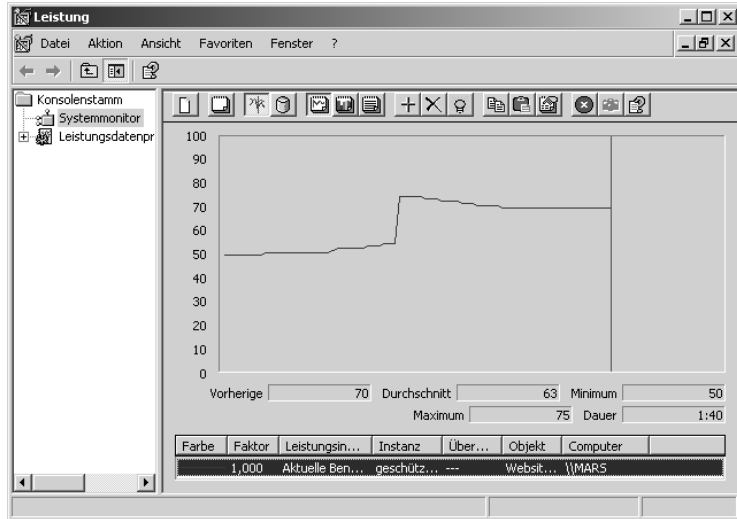


Abbildung 14.9: Anzeige der Veränderung der Nutzeranzahl im Systemmonitor

15 Namespace System.DirectoryServices

Die Klassen des FCL-Namespace `System.DirectoryServices` sind eine Kapselung des Active Directory Service Interfaces (ADSI). ADSI ist eine COM-Komponente zum Zugriff auf verschiedene Verzeichnisdienste. Allgemeine Informationen zu ADSI und der ADSI-COM-Komponente erhalten Sie in Anhang B. Sie werden in diesem Kapitel feststellen, dass Sie auch unter .NET noch gute Kenntnisse der ADSI-COM-Komponente benötigen.

ADSI

Microsoft hat auch in diesem Fall das Objektmodell stark verändert. Dabei ist das neue Objektmodell nur eine zusätzliche Schicht über ADSI; im Hintergrund arbeiten die bisherigen ADSI-Klassen. Dies bedeutet, dass die Klassen im Namespace `System.DirectoryService` nicht funktionieren, wenn die ADSI-COM-Komponente nicht installiert ist.

Auf die ADSI-COM-Komponente wird in diesem Buch auch mit dem Begriff Classic-ADSI Bezug genommen.



15.1 Architektur

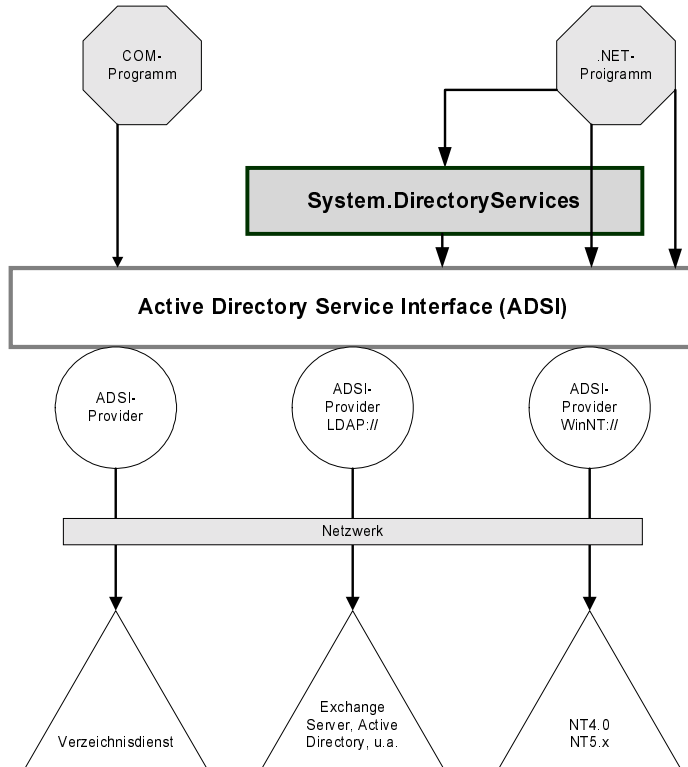
Die Klassen im FCL-Namespace `System.DirectoryServices` bieten nur sehr allgemeine Mechanismen für den Zugriff auf Verzeichnisdienste. Es gibt keine spezifischen Klassen mehr für einzelne Verzeichnisdienste, wie sie in der ADSI-COM-Komponente vorhanden waren. Bestimmte Operationen (z.B. Ändern des Kennworts in einem Benutzer-Objekt) müssen daher direkt oder indirekt über die ADSI-COM-Komponente aufgerufen werden.

**Allgemeine
Mechanismen**

Die folgende Grafik zeigt die Architektur von ADSI unter .NET. Ein .NET-Programm (Managed Code) hat drei Möglichkeiten, auf ADSI zuzugreifen:

**Zugriffs-
möglichkeiten**

1. Verwendung von Objekten im Namespace `System.DirectoryServices` zur Ausführung von Verzeichnisdienst-Operationen
2. Verwendung von Objekten im Namespace `System.DirectoryServices` für den Aufruf von Operationen in der ADSI-COM-Komponente
3. Direkte Verwendung der ADSI-COM-Komponente via COM-Interoperabilität



(C) Holger@Schwichtenberg.de 2002

Abbildung 15.1: Architektur von ADSI unter .NET

Weiterreichen an COM-ADSI

Kapselung Den Beweis dafür, dass alle Aufrufe in `System.DirectoryServices` in COM-ADSI umgesetzt werden, liefern die Fehlermeldungen der FCL. Zum Beispiel liefert die Klasse `DirectoryEntry` beim Aufruf von `CommitChanges()` folgende Fehlermeldung, wenn das anzulegende Objekt bereits vorhanden ist:

Unhandled Exception: The program '[1520] FCL-Buch.exe' has exited with code 0 (0x0).

System.Runtime.InteropServices.COMException (0x80071392): Das Objekt ist bereits vorhanden.

at System.DirectoryServices.Interop.IAds.SetInfo()

at System.DirectoryServices.DirectoryEntry.CommitChanges()

Dies bedeutet nichts anderes, als dass der Aufruf `CommitChanges()` in der Klasse `DirectoryEntry` intern weitergereicht wurde an die Methode `SetInfo()` in der Schnittstelle `System.DirectoryServices.Interop.IAds`. Dabei ist `SetInfo()` die aus der klassischen ADSI-COM-Komponente bekannte Methode, um den Property Cache an den Verzeichnisdienst zurückzuliefern und damit alle Änderungen persistent zu machen.

Der Namespace `System.DirectoryServices.Interop` ist undokumentiert und im Objektkatalog von Visual Studio .NET nicht sichtbar. In diesem Namespace sind die aus dem klassischen ADSI bekannten Schnittstellen `IAds`, `IAdsContainer` etc. definiert. Da in .NET eine Instanziierung von Schnittstellen nicht mehr möglich ist, mussten die Schnittstellen zu Klassen zusammengefasst werden.



15.2 Objektmodell

Die Klassen im Namespace `System.DirectoryServices` lassen sich in zwei Gruppen einteilen:

- ▶ allgemeine Klassen für den Zugriff auf Blätter und Container
- ▶ Klassen für die Ausführung von LDAP-Suchanfragen

15.2.1 Allgemeine Klassen

Die beiden zentralen Klassen in diesem Namespace sind `DirectoryEntry` und `DirectoryEntries`.

**Blätter und
Container**

Klasse »DirectoryEntry«

Die Klasse `DirectoryEntry` repräsentiert einen beliebigen Verzeichniseintrag, egal ob es sich um ein Blatt oder einen Container handelt. Diese Klasse besitzt ein Attribut `Children` vom Typ `DirectoryEntries`. Diese Collection ist nur dann gefüllt, wenn das Objekt ein Container ist, also Unterobjekte besitzt. Die Collection existiert aber auch in einem Blatt-Objekt; sie ist dann allerdings leer. Die

**Property
Collection**

DirectoryEntry-Klasse besitzt im Attribut `Property` eine Collection vom Typ `PropertyCollection`, die die Menge der Verzeichnisattribute des Verzeichnisobjekts repräsentiert. Die `PropertyCollection` besitzt drei untergeordnete Objektmengen:

- ▶ `PropertyNames` zeigt auf ein `KeysCollection`-Objekt, das Strings mit den Namen aller Verzeichnisattribute enthält.
- ▶ `Values` zeigt auf eine `ValueCollection`, die wiederum einzelne Collections vom Typ `PropertyValueCollection` enthält. Dies ist notwendig, da jedes Verzeichnisattribut mehrere Werte haben kann. Die `ValueCollection` repräsentiert die Menge der Werte aller Verzeichnisattribute, die `PropertyValueCollection` die einzelnen Werte eines Verzeichnisattributs.
- ▶ Das Attribut `Item(ATTRIBUTNAME)` liefert für einen als Parameter zu übergebenden Attributnamen die zugehörige `PropertyValueCollection`.



Der Zugriff über das Attribut `Values` kommt in der Regel nicht vor, da man normalerweise die Werte ohne die Namen der Attribute benötigt. Der normale Weg ist entweder die direkte Verwendung von `Item()`, wenn der Attributname bekannt ist, oder aber die Iteration über `PropertyNames` und darauf folgend die Verwendung von `Item()`, wenn alle Attribute mit ihren Werten aufgelistet werden sollen.

NativeObject

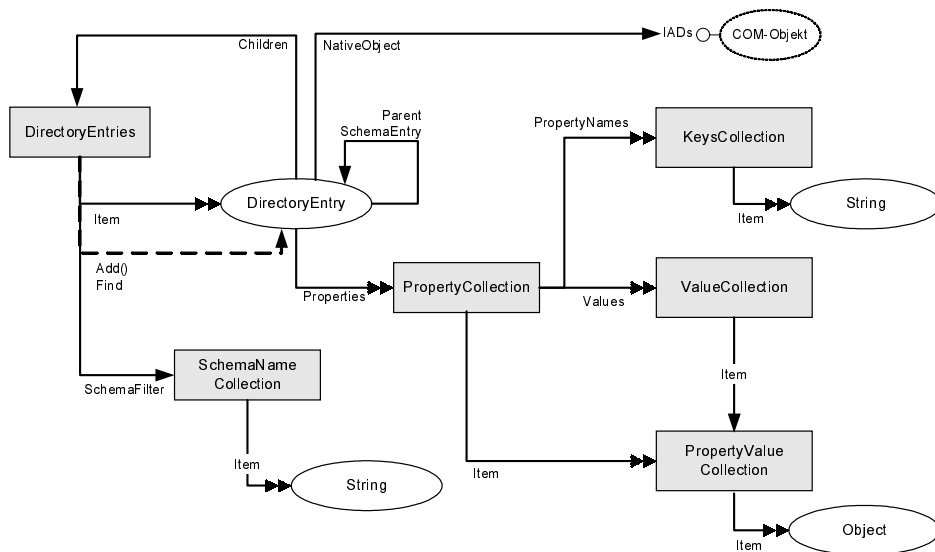
Jedes `DirectoryEntry`-Objekt besitzt ein Attribut mit Namen `NativeObject`, das einen Verweis auf das zugehörige ADSI-COM-Objekt liefert. Damit ist ein schneller Wechsel zur klassischen ADSI-Programmierung möglich.

Klasse »DirectoryEntries«

Container-Objekte

Die Klasse `DirectoryEntries` unterstützt die Schnittstelle `IEnumerable` und ermöglicht daher die Auflistung ihrer Mitglieder über eine `For Each`-Schleife. Die Menge kann gefiltert werden, indem über die `SchemaNameCollection` eine Menge von Verzeichnisdienst-Klassen spezifiziert wird, die berücksichtigt werden soll. Die Methode `Find()` liefert ein `DirectoryEntry`-Objekt. Wenn das anhand des Namens spezifizierte Objekt nicht in diesem Container vorhanden ist, gibt es eine `InvalidOperationException`.

Die Klasse `DirectoryEntries` kann nicht instanziiert werden. Sie erhalten ein `DirectoryEntries`-Objekt immer nur über das Attribut `Children` eines `DirectoryEntry`-Objekts.



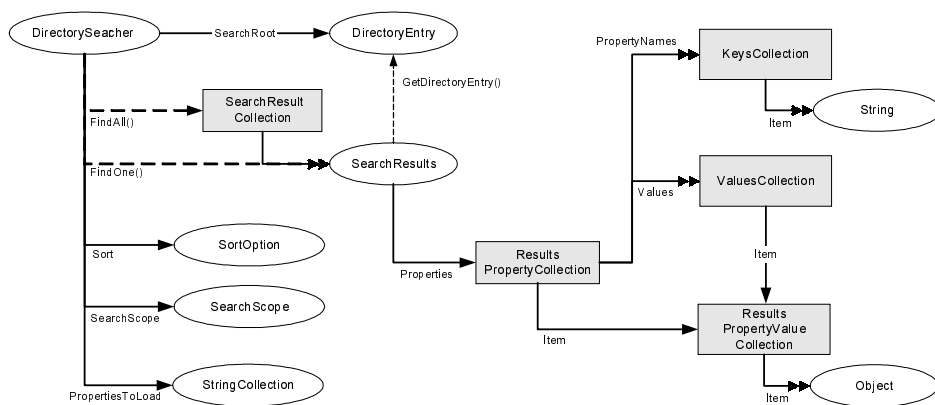
(C) Holger@Schwichtenberg.de 2002

Abbildung 15.2: Objektmodell der Klassen im Namespace »System.DirectoryServices« – Teil I

15.2.2 Klassen für die Ausführung von Suchanfragen

Suchanfragen wurden in COM-ADSI über die ActiveX Data Objects (ADO) bzw. einen OLE DB-Provider ausgeführt. In .NET gibt es nun eigene Klassen für die Ausführung von LDAP-Suchanfragen, die unabhängig von ADO.NET sind.

LDAP-Suchanfragen



(C) Hohere@Schwiebenberg.de 2002

Abbildung 15.3: Objektmodell der Klassen im Namespace »System.DirectoryServices« – Teil 2

15.2.3 Vergleich zwischen .NET-ADSI und COM-ADSI

Die folgende Tabelle zeigt, dass es für viele Schnittstellen aus der ADSI-COM-Komponente keine entsprechende spezifische Klasse im .NET-ADSI mehr gibt.

Verzeichnisobjekt-Klasse im Active Directory	ADSI in COM	ADSI in .NET (System.DirectoryServices)
Blatt-Klassen	Schnittstelle IADs	Klasse DirectoryEntry
Container-Klassen	Schnittstelle IADsContainer	Klasse DirectoryEntries
Klasse »User«	Schnittstelle IADsUser	--- (DirectoryEntry)
Klasse »Computer«	Schnittstelle IADsComputer	--- (DirectoryEntry)
Klasse »Group«	Schnittstelle IADsGroup	--- (DirectoryEntry)
---	Klasse ADODB.Connection	Klasse DirectorySearcher
beliebige Klassen	Klasse ADODB.RecordSet	Klasse SearchResultCollection

Tabelle 15.1: NET-ADSI vs. COM-ADSI

15.3 Überblick über die Programmiermechanismen

Dieses Kapitel dokumentiert die wichtigsten Mechanismen der Verzeichnisdienst-Programmierung mit System.DirectoryServices.

15.3.1 Objektidentifikation und Bindung

ADSI-Pfade ADSI bedient sich auch unter .NET der COM-Moniker, um einzelne Einträge in verschiedenen Verzeichnisdiensten anzusprechen und einen Zeiger auf das Stellvertreter-Objekt zu erhalten. Der Moniker hat die Form

<namespace-ID>:<providerspezifischer Teil>

und wird in diesem Zusammenhang ADSI-Pfad genannt.

Der providerspezifische Teil enthält in der Regel den Distinguished Name (DN) des Verzeichnisobjekts und dazu einen Servernamen. Es gibt jedoch auch andere Formen, gerade im Zusammenhang mit dem Active Directory. Bei der Namespace-ID werden Groß- und Kleinschreibung berücksichtigt. Beispiele für vollständige ADSI-Pfade zeigt die folgende Tabelle.

DN und RDN

Namespace	Beispiele für ADSI-Pfade
LDAP	LDAP://server/CN=Mitarbeiter,DC=ITVISIONS,DC=DE LDAP://sonne2000.it-visions.de/CN=HS, OU=Marketing,DC=DEV,DC=ITVISIONS,DC=DE
NT 4.0	WinNT://Domaene/Computer/Benutzer WinNT://Computername/Gruppenname WinNT://domaene/benutzer
Novell 3.x	NWCOMPAT://NWServer/Druckername
Novell 4.x (NDS)	NDS://Server/O=ITVisions/OU=Essen/ CN=Mitarbeiter
IIS ab 4.0	IIS://ComputerName/tw3svc/1

Tabelle 15.2: Beispiele für ADSI-Pfade in verschiedenen Verzeichnisdiensten

Bindung an ein Verzeichnisobjekt

Voraussetzung für den Zugriff auf Objekte des Verzeichnisdienstes ist die Bindung eines ADSI-Objekts an ein Verzeichnisobjekt. Während unter Classic-ADSI der Bindungsvorgang über die Methode `GetObject()` stattfand, wird dies in .NET-ADSI über einen Parameter bei der Instanziierung der Klasse `DirectoryEntry` erledigt.

Instanziierung von DirectoryEntry

```
o = New DirectoryEntry("ADSI_PFAD")
```

Impersonifizierung

Das Wort *Impersonifizierung* (engl. impersonification) sucht man in Wörterbüchern (noch) vergeblich. In Fachkreisen (so auch in Microsoft-Dokumentationen) wird dieser Begriff für einen Wechsel des Benutzerkontextes im laufenden Betrieb verwendet. Ein Benutzer kann also in die Rolle eines anderen Benutzers wechseln, ohne sich neu am System anmelden zu müssen. Diese Funktion ist besonders wichtig für Administratoren, die nur gelegentlich bestimmte administrative Aufgaben ausführen müssen. Nicht ständig unter einem administrativen Konto angemeldet zu sein, erhöht den Schutz vor Viren und anderen Angriffen.

Wechsel des Benutzerkontextes

Der ADSI-Client verwendet den ADSI-Impersonifizierungsmodus durch Angabe eines Benutzernamens und eines Kennworts bei der Instanziierung der Klasse `DirectoryEntry`.

```
o = New DirectoryEntry("ADSI_PFAD", "ADMINUSER", _
    "ADMINPASSWORD")
```

Prüfung auf Existenz

Exists() Das Classic-ADSI hatte keine eingebaute Methode, um die Existenz eines Verzeichnisobjekts zu überprüfen. Man war dort auf die (zeitaufwendige) Try-and-Fail-Methode angewiesen. Unter .NET bietet die Klasse `DirectoryEntry` die statische Methode `Exists()` an, mit der geprüft werden kann, ob ein anhand seines ADSI-Pfades spezifiziertes Verzeichnisobjekt existiert.

```
Ja_oder_Nein = DirectoryEntry.Exists("ADSI_PFAD")
```

Beispiel Die folgende Routine gibt für einen übergebenen ADSI-Pfad aus, ob das Verzeichnisobjekt existiert oder nicht.

```
' ### Prüfung, ob ein Verzeichnisobjekt vorhanden ist
Function ADSI_exists(ByVal pfad As String)
    out("Der Eintrag " & pfad & " ist " & _

    IIf(DirectoryEntry.Exists(pfad), _
        "vorhanden", "nicht vorhanden"))

    ADSI_exists = DirectoryEntry.Exists(pfad)
End Function
```

Listing 15.1: Prüfung, ob ein Verzeichnisobjekt vorhanden ist [ADSI_allgemein.vb]

15.3.2 Zugriff auf Attribute und Methoden

Invoke() Im Vergleich zum Classic-ADSI ist die Verzeichnisdienst-Programmierung unter .NET mit etwas mehr Kodierungsaufwand verbunden. Die Hauptursache ist, dass es keinen direkten Zugriff mehr auf Verzeichnisattribute und Verzeichnisoperationen gibt, sondern alle Verzeichnisattribute über die `Properties-Collection` angesprochen und alle Verzeichnisoperationen über `Invoke()` durch spätes Binden aufgerufen werden müssen.

Im Classic-ADSI konnten – dank der verschiedenen ADSI-Schnittstellen – alle Verzeichnisoperationen und viele Verzeichnisattribute direkt (über frühes Binden) aufgerufen werden. Nur die verzeichnisdienstspezifischen Verzeichnisattribute mussten etwas umständlicher über `Get()` und `Put()` verwendet werden.

Attribute lesen

Während das Classic-ADSI nur diejenigen Attribute, die mehrwertig sind, als Array zurückliefert, gibt .NET-ADSI immer eine `PropertyValueCollection` zurück. Dies hat zur Folge, dass auch bei einwertigen Attributen immer explizit das erste Element der Liste angesprochen werden muss.

Während ein Attributzugriff im Classic-ADSI in der Regel so aussah

```
xy = obj.Attributname
```

ist also in .NET-ADSI mehr Aufwand nötig:

```
xy = obj.Properties("Attributname")(0)
```

Wenn ein Attribut keinen Wert hat, dann führt der Zugriff auf das Attribut in .NET genauso zu einem Fehler wie in COM-ADSI. Während man dies in COM nur mit der Laufzeitfehlerbehandlung (on error...) abfangen konnte, bietet die FCL nun eine elegantere Möglichkeit über die Abfrage des Attributs `Count` in der `PropertyValueCollection`.

```
obj.Properties("Attributname").Count
```

Wenn `Count` den Wert 0 hat, dann besitzt das Attribut keinen Wert.

Bei mehrwertigen Attributen liefert `Count` die Anzahl der vorhandenen Werte. Auf die einzelnen Werte kann dann über den Index in der `PropertyValueCollection` zugegriffen werden:

```
obj.Properties("Attributname")(INDEX)
```

Die verschiedenen Prüfungsvorgänge beim Auslesen eines Verzeichnisattributs kann man gut in eine Hilfsroutine packen. Die Funktion `getAtt()` erwartet als Parameter einen Zeiger auf ein `DirectoryEntry`-Objekt und den Namen eines Attributs. Sie liefert eine Zeichenkette (`System.String`) mit dem Attributwert zurück. Bei mehrwertigen Attributen werden die Einzelwerte durch ein Semikolon voneinander getrennt.

```
' ### Liefert die Werte eines Attributs in einem durch
Semikola getrennten String
Function getAtt(ByVal se As Object, ByVal attributname As
String) As String
    Dim werteliste As Object
    'ResultPropertyValueCollection oder
    'PropertyValueCollection
```

**Attribute lesen
über die Proper-
ties-Collection**

Count

**Mehrwertige
Attribute**

**Hilfsroutine
getAtt()**

```

Dim wert As Object ' einzelner Wert
Dim ergebnis As String ' Rückgabestring

' --- Prüfung, ob Attribut vorhanden
If se.Properties.Contains(attributname) Then
    ' --- Zugriff auf Wertemenge
    werteliste = se.Properties(attributname)
    ' --- Schleife über alle Werte
    For Each wert In werteliste
        ' --- Ergebnis zusammensetzen
        If Len(ergebnis) = 0 Then
            ergebnis = wert
        Else
            ergebnis = ergebnis & ";" & wert
        End If
    Next
End If

Return (ergebnis)
End Function

```

Listing 15.2: Diese Hilfsroutine liefert die Werte eines Attributs in einem durch Semikola getrennten String [ADSI_allgemein.vb]



Damit diese Routine auch mit einem SearchResult-Objekt bzw. dessen ResultPropertyValueCollection funktioniert, verwendet die Routine spätes Binden (as Object).

Attribute schreiben

Attribute schreiben über die Properties-Collection

Soll ein Attributwert gesetzt werden, muss unterschieden werden, ob das Attribut bereits einen Wert besitzt oder nicht. Um einen bestehenden Wert zu ändern, verwendet man

```
obj.Properties("Attributname")(INDEX) = xy
```

Dabei ist INDEX die laufende Nummer des Werts. Dieser Zugriff ist aber nur erlaubt, wenn es überhaupt INDEX-1 Werte gibt.

Bei einwertigen Attributen ist INDEX immer 0, also:

```
obj.Properties("Attributname")(0) = xy
```

Add () Besitzt das Attribut noch keinen Wert oder soll zu einem mehrwertigen Attribut ein Wert hinzugefügt werden, muss man die Add()-Methode verwenden.

```
obj.Properties("Attributname").Add(Wert)
```

Bei einem einwertigen Attribut, das bereits einen Wert besitzt, führt die Ausführung der Methode `Add()` zu einem Fehler. Wenn ein einwertiges Attribut gesetzt werden soll, muss also zunächst geprüft werden, ob es schon einen Wert gibt. Dafür kann man `Count` verwenden.

```
obj.Properties("Attributname").Count
```

Wenn es einen Wert gibt (`count > 0`), muss über den Index zugegriffen werden. Wenn es noch keinen Wert gibt (`count = 0`), muss man den Wert über `Add()` anlegen. Man kann sich die Arbeit etwas vereinfachen, indem man zunächst die Methode `Clear()` aufruft, um alle Werte zu löschen.

```
obj.Properties("Attributname").Clear
```

Danach kann man mit Sicherheit `Add()` verwenden, ohne einen Laufzeitfehler zu riskieren.

Methoden aufrufen

Alle Verzeichnisoperationen müssen über

Invoke()

```
obj.invoke("Methodenname", Parameter1, Parameter2, etc)
```

aufgerufen werden, wobei »Methodenname« der Name einer Methode im Classic-ADSI ist. Der Aufruf einer Verzeichnisoperation erfolgt also in der FCL immer durch spätes Binden. Ein direkter Aufruf der Methoden ist (noch) nicht möglich. Die Namen und Parameter der verfügbaren Verzeichnisoperationen entsprechen den Methoden der COM-Schnittstellen des Classic-ADSI.

Classic-ADSI versus .NET-ADSI

Die folgende Tabelle stellt die Vorgehensweise in Classic-ADSI und .NET-ADSI zusammenfassend gegenüber.

	Classic-ADSI	.NET-ADSI
Attribut lesen	xy = obj.Attributname xy = obj.Get ("Attributname")	xy = obj.Properties ("Attributname ")(0)
Attribut setzen	obj.Attributname = xy obj.Put "Attributname", xy	obj.Properties ("Attributname ")(0) = xy obj.Properties ("Attributname").Add(xy)
Methodenaufrufe	obj.methodenname (Parameter1, Parameter2.etc)	obj.invoke ("Methodenname", Parameter1, Parameter2,etc)

Tabelle 15.3: Attributzugriffe und Methodenaufrufe in Classic-ADSI und .NETADSI

Basiseigenschaften

Mitglieder der DirectoryEntry-Klasse

Die Meta-Klasse DirectoryEntry besitzt einige wenige Attribute, die Basiseigenschaften eines Verzeichnisdienstobjekts enthalten. Dies sind:

- ▶ Name: Relative Distinguished Name des Objekts
- ▶ Path: Distinguished Name des Objekts
- ▶ SchemaClassName: Name der Verzeichnisdienst-Klasse im Schema des Verzeichnisdienstes
- ▶ Guid: Global Unique Identifier (GUID) des Meta-Objekts
- ▶ NativeGuid: der Global Unique Identifier (GUID) für das Verzeichnisdienst-Objekt

Auflisten der Verzeichnisattribute

Iteration über die Properties-Collection

Auf die Verzeichnisdienstattribute (Attribute von Verzeichnisobjekten) kann man nicht direkt zugreifen. Hierzu muss die Properties-Collection verwendet werden. Dabei ist zunächst über die der Properties-Collection untergeordnete Values-Collection zu iterieren, um die Namen der Attribute zu ermitteln. Die Wertemenge eines Verzeichnisattributs erhält man anschließend über item("Attributname"). Für jedes Attribut ist dann noch eine Iteration über die PropertyValueCollection notwendig, um die einzelnen Attributwerte auszugeben.

Die folgende Routine listet für ein beliebiges Verzeichnisdienstobjekt zunächst die Basiseigenschaften auf und dann alle Attribute des Verzeichnisdienstobjekts über die `Properties-Collection`.

Beispiel

```
' =====
' Beispiel aus FCL-Buch.sln
' Alle Eigenschaften eines Verzeichniseintrags ausgeben
' (C) Holger@Schwichtenberg.de
' =====
Sub ADSI_EintragLesen(ByVal pfad As String)
    Dim o As DirectoryEntry
    Dim werteliste As PropertyValueCollection
    Dim attributname As String
    Dim wert As Object

    out("# Lesen des Eintrags: " & pfad)

    ' --- Zugriff auf Eintrag
    o = New DirectoryEntry(pfad)
    ' --- Basisdaten des Eintrags
    out("Name: " & o.Name)
    out("Pfad: " & o.Path)
    out("Klasse: " & o.SchemaClassName)
    out("GUID: " & o.Guid.ToString)
    out("Native GUID: " & o.NativeGuid)

    ' --- Alle Verzeichnisattribute des Eintrags auflisten
    out("Alle Verzeichnisattribute:")
    out(o.Properties.Count)
    For Each attributname In o.Properties.PropertyNames
        out(attributname)
        ' --- Werteliste holen
        werteliste = o.Properties.Item(attributname)
        ' --- Werte einzeln ausgeben
        For Each wert In werteliste
            out("    " & wert.ToString())
        Next
    Next
Next

End Sub
```

Listing 15.3: Alle Eigenschaften eines Verzeichniseintrags ausgeben [ADSI_Allgemein.vb]

NativeObject

Interoperabilität

Das Attribut `NativeObject` enthält einen Verweis auf das entsprechende ADSI-COM-Objekt, genau genommen auf dessen `IADs-Schnittstelle`.

```
iads = o.NativeObject
```

ADSI Property Cache

Caching Da ADSI-Objekte nur Stellvertreter für Verzeichniseinträge sind, werden die Attributwerte in einem Property Cache verwaltet. Beim ersten Zugriff auf ein Attribut lädt ADSI alle Attributwerte in den Cache. Schreibzugriffe sind durch Zuweisungen an die Attribute möglich.

GetInfo(), SetInfo() Alle Schreibzugriffe müssen mit einem Aufruf der Methode `CommitChanges()` (`SetInfo()` unter Classic-ADSI) abgeschlossen werden. Erst dann wird der Cache an den zugrunde liegenden Verzeichnisdienst übergeben. Damit wird auch die Transaktions-sicherheit gewährleistet: Entweder werden alle Änderungen ausgeführt oder keine. Auch für das Einlesen der Attribute in den Cache gibt es eine Methode: `RefreshCache()` (entspricht `GetInfo()` unter Classic-ADSI). Das Programm sollte sie explizit aufrufen, wenn nicht sicher ist, ob die Werte im Cache noch aktuell sind. Mit `RefreshCache()` können auch Änderungen verworfen werden, wenn zwischen den Änderungen und dem `RefreshCache()` kein `CommitChanges()` steht. Durch Angabe eines Arrays mit Attributnamen bei `RefreshCache(ARRAY_OF_STRING)` können vor einem ersten Attributzugriff gezielt einzelne Werte in den Cache gelesen werden, um zur Verringerung der Netzwerklast die Übertragung aller Attribute zu vermeiden.

Abschalten des Property Caches Im Gegensatz zu COM-ADSI bietet .NET-ADSI die Möglichkeit, den Property Cache auszuschalten. Dazu ist nach der Instanziierung des `DirectoryEntry`-Objekts folgender Befehl notwendig:

```
obj.UsePropertyCache = False
```

Wichtiger Hinweis Die Abschaltung des Property Caches funktioniert nicht beim Anlegen von Verzeichnisobjekten von Verzeichnisklassen, die Pflichtattribute haben, da der Verzeichnisdienst den Eintrag erst erzeugt, wenn alle Pflichtattribute übergeben wurden.

15.3.3 Zugriff auf Container-Objekte

Container-Objekte Die Bindung an Container-Objekte und der Zugriff auf deren Verzeichnisattribute erfolgt vollkommen identisch zum Zugriff auf Blatt-Objekte, also über die Klasse `DirectoryEntry`. Sollen die Unterobjekte des Containers aufgelistet werden, muss jedoch das Unterobjekt `Children` angesprochen werden, das ein `DirectoryEntries`-Objekt liefert.

Die Iteration durch ADSI-Container erfolgt in Visual Basic analog zur Iteration durch Collections mit der `For Each`-Schleife. Sie bindet eine Laufvariable vom Typ `DirectoryEntry` nacheinander an die Objekte im Container.

**Iteration mit
For Each**

```
' =====
' Beispiel aus FCL-Buch.sln
' Liste der Unterobjekte eines Containers
' (C) Holger@Schwichtenberg.de
' =====
Sub ADSI_ContainerAuflisten(ByVal pfad As String)
    Dim o As DirectoryEntry
    Dim c As DirectoryEntries

    out("# Auflisten des Containers: " & pfad)

    ' --- Zugriff auf IADs
    o = New DirectoryEntry(pfad)
    ' --- Wechsel zu IADsContainer
    c = o.Children
    ' --- Schleife über alle Containerelemente
    For Each o In c
        out(o.Name)
    Next
End Sub
```

Listing 15.4: Liste der Unterobjekte eines Containers [ADSI_allgemein.vb]

15.3.4 Instanzenverwaltung

Die `DirectoryEntries`-Klasse stellt die Methoden zur Verwaltung von Objekten bereit: `Add()` erzeugt neue Objekte, `Remove()` löscht bestehende Objekte.

Vor der Ausführung von `Add()` und `Remove()` sollten Sie die Existenz des jeweiligen Objekts prüfen, um einen Laufzeitfehler zu vermeiden.

Objekt anlegen

Ein Verzeichnisobjekt wird über den übergeordneten Container angelegt, weil nur dieser weiß, ob er eine bestimmte Verzeichnis-klassse als Unterobjekt überhaupt zu akzeptieren bereit ist. Die Methode `Add()` erwartet im ersten Parameter den Relative Distinguished Name (RDN) des neuen Objekts und im zweiten Parameter den Namen der Verzeichnisdienst-Klasse, die als Schablone für

Add()

das Objekt verwendet werden soll. Nach dem Setzen eventuell vorhandener Pflichtattribute muss noch `CommitChanges()` aufgerufen werden.

```
obj = New DirectoryEntry("CONTAINER_PFAD")
container = obj.Children
neues_obj = container.Add("RDN", "KLASSE")
neues_obj.Properties("ATTRIBUTNAME").Add("WERT")
neues_obj.CommitChanges()
```

Objekt löschen

Remove() Ein Objekt wird nicht durch einen Methodenaufruf auf sich selbst, sondern über die Ausführung von `Remove()` auf einem Container-Objekt gelöscht. Dabei ist als Parameter das `DirectoryEntry`-Objekt, das das zu löschende Verzeichnisobjekt repräsentiert, anzugeben. Der Aufruf von `CommitChanges()` ist nicht nötig.

```
obj = New DirectoryEntry("CONTAINER_PFAD")
neues_obj = container.Remove(DE_OBJECT)
```

Beispiel Die folgende allgemeine Hilfsroutine löscht ein Objekt, das keine Unterobjekte besitzt. Als Parameter übergeben werden der DN des Containers und der RDN des Objekts.

```
' =====
' Beispiel aus FCL-Buch.sln
' Löschen eines Verzeichnisobjekts, das keine Kinder hat
' (C) Holger@Schwichtenberg.de
' =====

Sub ADSI_loeschen(ByVal container As String, ByVal oname As String)

    Dim o As DirectoryEntry
    Dim c As DirectoryEntries
    ' --- Zugriff auf IADs

    o = New DirectoryEntry(container)
    ' --- Zugriff auf IADsContainer
    c = o.Children
    ' --- Objekt löschen!
    c.Remove(c.Find(oname))

    out(oname & "gelöscht!")

End Sub
```

Listing 15.5: Löschen eines Verzeichnisobjekts, das keine Kinder hat
[ADSI_allgemein.vb]

Rekursives Löschen

Üblicherweise können Container-Objekte erst dann gelöscht werden, wenn sie leer sind. Die Routine `ADS_loeschen_rekursiv()` löscht in dem Fall, dass der übergebene ADSI-Pfad einen Container darstellt, zunächst rekursiv alle Unterobjekte. Die Routine ist aber auch eine Erleichterung für Blatt-Objekte, da ein Verzeichnisobjekt direkt über seinen Pfad gelöscht werden kann.

Rekursives Löschen

Die folgende allgemeine Hilfsroutine löscht ein Verzeichnisobjekt, egal ob es Unterobjekte besitzt oder nicht. Als Parameter wird der DN des zu löschenden Verzeichnisobjekts übergeben. Wenn Unterobjekte vorhanden sind, ruft sich diese Routine rekursiv selbst auf.

Beispiel

```
' ### Löscht alle eventuell vorhandenen Unterobjekte und
' ### schließlich das angegebene Objekt selbst
Sub ADSI_Loeschen_Rekursiv(ByVal dn As String)
    Dim o As DirectoryEntry
    Dim kind As DirectoryEntry
    Dim vater As DirectoryEntries
    Dim rdn As String

    out("# Rekursives Löschen von: " & dn)

    ' --- Zugriff auf IADs
    o = New DirectoryEntry(dn)
    ' --- Relativen Namen ermitteln
    rdn = o.Name

    ' --- Zuerst alle Unterobjekte rekursiv löschen
    For Each kind In o.Children
        ADSI_Loeschen_Rekursiv(kind.Path)
    Next

    ' --- Zugriff auf IADsContainer
    vater = o.Parent.Children

    ' --- Dieses Objekt löschen!
    vater.Remove(o)

    ' --- Ausgabe
    out("Gelöscht: " & dn)
End Sub
```

Listing 15.6: Rekursives Löschen in einem Verzeichnisdienst [*ADSI_allgemein.vb*]

15.4 Active Directory-Programmierung

Dieses Kapitel erläutert die Verwendung der Klassen im Namespace `System.DirectoryServices` für den Zugriff auf das Microsoft Active Directory im Windows 2000 Server bzw. Windows .NET Server via Lightweight Directory Access Protocol (LDAP). ADSI bietet dafür den LDAP-Provider.

LDAP-Provider

Mit dem LDAP-Provider kann nicht nur das Active Directory, sondern jeder LDAP-Server (z. B. die Novell NDS oder ein LDAP-Server im Internet) angesprochen werden. Aber: Weder ältere Windows-Versionen noch ein Windows 2000 Professional oder ein Windows 2000 Server ohne Active Directory kann auf die in diesem Unterkapitel dargestellte Weise verwendet werden. Benutzen Sie in diesen Fällen den WinNT-Provider (siehe folgende Abbildung).

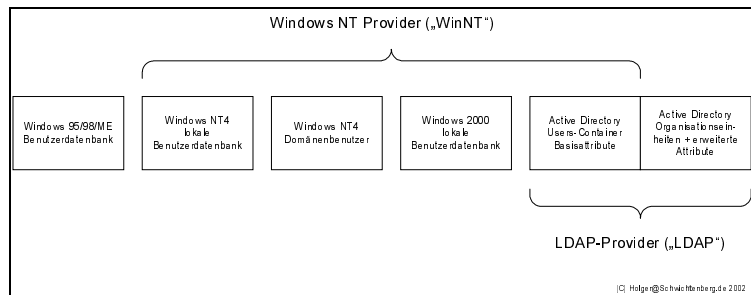


Abbildung 15.4: Überblick über die Benutzerverwaltung mit dem WinNT-Provider und dem LDAP-Provider

15.4.1 Grundlagen zur Active Directory-Programmierung

Umfangreiches Active Directory

Das Active Directory (AD), auch Active Directory Service (ADS) genannt, ist der Verzeichnisdienst, den Microsoft erstmalig mit Windows 2000 Server ausgeliefert und in Windows .NET Server verbessert hat. Das Active Directory ist ein sehr leistungsstarker, aber auch sehr komplizierter Verzeichnisdienst. Dies zeigt sich schon am Umfang des AD-Schemas: Nach einer Standardinstallation unter Windows 2000 Server ohne Schema-Erweiterungen umfasst das Schema 142 Klassen und 863 Attribute. Im Vergleich dazu hat Windows NT 4.0 nur 15 Klassen und 77 Attribute. Das Zugriffsprotokoll für das Active Directory ist LDAP. Sie müssen in ADSI daher den ADSI-LDAP-Provider verwenden.

Durch die Installation eines Exchange Servers 2000, der auf dem Active Directory aufsetzt, steigt die Zahl der Klassen unter Windows 2000 Server auf 299 und die Zahl der Attribute auf 1705 an.

Exchange 2000

Active Directory-Grundlagen

Ein Active Directory hat eine komplexe Struktur. Grundbaustein sind die aus NT4 bekannten Domänen. Eine Domäne besitzt einen oder mehrere im Wesentlichen gleichberechtigte Domänen-Controller (DC). Eine Unterscheidung in Primary Domain Controller (PDC) und Backup Domain Controller (BDC) findet nicht mehr statt. Nach dem Multi-Master-Prinzip kann auf jedem Domänen-Controller das Verzeichnis nicht nur gelesen, sondern auch verändert werden. Lediglich die Modifikation des Schemas ist auf einem bestimmten ausgewählten DC, dem Schema-Master, möglich.

**Aufbau eines
Active Directory**

Das AD kennt einige wenige Sonderrollen, die nur ein Server haben kann. Diese werden Flexible Single Master Operations (FSMO) genannt. In der deutschen MMC findet man den Begriff »Betriebsmaster«.

Sonderrollen

Domänen können zu so genannten *Trees* (Bäumen) und diese wiederum zu *Forests* (Wäldern) zusammengefasst werden. Eine Site ist dagegen kein logisches Strukturierungsmittel für den Verzeichnisdienst, sondern spiegelt die physikalische Struktur des Rechnernetzes wider. Die Definition von Sites dient vor allem dazu, die Replikationslast zwischen den Domain Controllern eines ADs zu regulieren.

**Bäume in Wälder
zusammenfassen**

Auf das Active Directory kann auch mit Hilfe des WinNT-Providers zugegriffen werden. Für die Benutzerverwaltung ist davon jedoch abzuraten, da dieses Vorgehen großen Einschränkungen unterworfen ist:

**Zugriff mit dem
WinNT-Provider**

- ▶ Es werden nur die im WinNT-Schema bekannten Klassen angezeigt.
- ▶ Es besteht nur Zugriff auf die im WinNT-Schema bekannten Attribute.
- ▶ Die Hierarchisierung in Organisationseinheiten ist nicht sichtbar: Alle Benutzer und Gruppen werden wie in NT4 üblich in einer flachen Liste angezeigt.

Diese Einschränkung gilt jedoch nur für die Verwaltung von Benutzern, Gruppen, Computern und Organisationseinheiten sowie für die im Active Directory abgebildeten Konfigurationsinformationen. Die Verwaltung von anderen Verzeichnisobjekten wie Windows NT-Diensten, Freigaben, Sessions etc. ist im Active Directory nicht möglich, so dass dafür weiterhin ausschließlich der WinNT-Provider verwendet werden kann.

Neuerungen Neuerungen im Active Directory gegenüber dem NT4-Verzeichnis (zur Frage, ob NT4 einen Verzeichnisdienst besitzt, siehe Kapitel 1.5) sind insbesondere:

- ▶ wesentlich mehr Klassen von Verzeichniseinträgen
- ▶ wesentlich mehr Attribute für die einzelnen Einträge
- ▶ die beliebig tiefe hierarchische Strukturierung der Einträge. Strukturierungsmöglichkeiten bestehen insbesondere durch die vordefinierten Container-Klassen `OrganizationalUnit` und `Container`.
- ▶ fein granulierte Zugriffsrechte auf jeder Ebene und auch für jedes Attribut, so dass eine Delegation von administrativen Aufgaben möglich wird
- ▶ ein erweiterbares, objektorientiertes Schema

Active Directory-Verzeichnisobjektmodell

Oberstes Element jedes Active Directory ist das Objekt »rootDSE«. Der Verzeichnisdienst teilt sich darunter in drei Partitionen:

DefaultNamingContext	▶ Der DefaultNamingContext ist die Sicht auf den Verzeichnisdienst, wie ihn das MMC Snap-In »Active Directory-Benutzer und Computer« bietet.
ConfigurationNamingContext	▶ Im ConfigurationNamingContext werden Einstellungen zu dem Active Directory zuarbeitenden Diensten, den Sites und dem Inter-Site-Transport, den erweiterten Rechten, den <i>Display Specifiers</i> (ein <i>Display Specifier</i> legt fest, mit welchen Eigenschaftsfenstern eine Klasse in der MMC angezeigt wird) und den <i>Well Known Security Principals</i> (z.B. <i>Everybody</i> , <i>System</i>) abgelegt.
SchemaNamingContext	▶ Der <code>SchemaNamingContext</code> enthält das Schema des Active Directory.

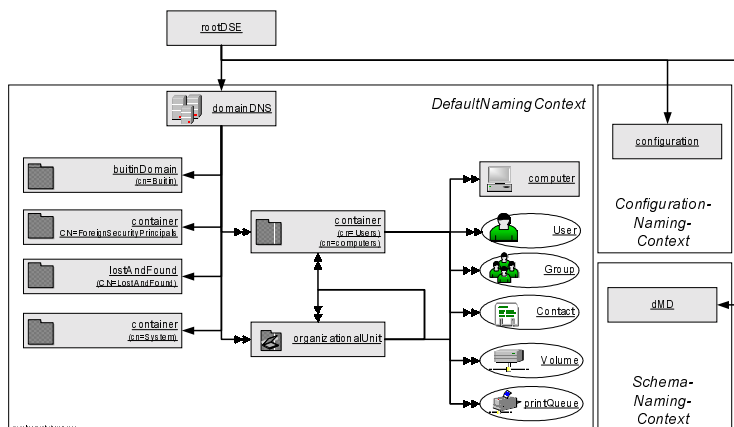


Abbildung 15.5: Ein kleiner Ausschnitt aus dem komplexen Objektmodell des Active Directory

Klassen im Active Directory

Alle vordefinierten Klassen im Active Directory sind Container-Klassen und können daher Unterobjekte besitzen. Die nachfolgende Tabelle zeigt eine Auswahl der wichtigsten der 142 AD-Klassen mit ihren Schlüsselattributen (fast immer cn).

Ausgewählte Klassen

Verzeichniseintrag	Klassenname (LDAP-Name)	Schlüsselattribut
Benutzer	"user"	"cn"
Gruppe	"group"	"cn"
Container	"container"	"cn"
Organisationseinheit	"organizationalUnit"	"ou"
Domänenbestandteil	"domainDNS"	"cn"
Verzeichnisfreigabe	"Volume"	"cn"
Computer	"Computer"	"cn"
Drucker	"printQueue"	"cn"
Kontakt	"Contact"	"cn"
AD-Schema	"dmD"	"cn"
Attribut im Schema	"attributeSchema"	"cn"
Klasse im Schema	"classSchema"	"cn"
Konfigurationscontainer	"configuration"	"cn"

Tabelle 15.4: Wichtigste Verzeichnisklassen im Active Directory

**Konvention**

Zur Unterscheidung von Klassen aus dem Active Directory und Klassen aus der FCL werden die Klassen aus dem Active Directory in diesem Buch immer in Anführungszeichen geschrieben. Gleiches gilt für die Unterscheidung zwischen den Attributen der Verzeichnisobjekte und den Attributen der Meta-Klassen.

Identifikation und Bindung**LDAP-Pfade**

Für den Zugriff auf das Active Directory verwendet ADSI Pfade der Form `LDAP://server:port/DN`. Dabei sind alle Bestandteile optional:

- ▶ Ohne Servernamen wird der so genannte *Locator Service* verwendet. Beim serverlosen Binden sucht der Active Directory Locator Service mit Hilfe des Domain Name Services (DNS) den besten Domain Controller für den angegebenen Verzeichniseintrag. Dabei erhalten Domain Controller, zu denen eine schnelle Verbindung besteht, den Vorzug.
- ▶ Ohne Portangabe wird der Standard-LDAP-Port 389 verwendet.
- ▶ Ohne Distinguished Name (DN) wird der `DefaultNamingContext` in der aktuellen Domäne angesprochen.

Bindung über Global Unique Identifier (GUIDs)**GUID-Bindung**

Bei der Adressierung über einen Textpfad besteht die Gefahr, dass Verzeichnisobjekte umbenannt wurden. Das Active Directory ermöglicht daher die Bindung über einen GUID, der für ein Verzeichnisobjekt unveränderlich ist. Der GUID muss natürlich für ein Objekt vorher bekannt sein.

```
LDAP://sonne2000/<GUID=228D9A87C30211CF9AA400AA004A5691>
```

Well Known Objects**Bindung über Well Known-GUIDs**

Für die Standardcontainer in einem Active Directory gibt es eine besondere Unterstützung. Für diese so genannten *Well Known Objects* besteht ein vordefinierter GUID, der in jedem Active Directory gleich ist.

```
LDAP://<WKGUID=a9d1ca15768811d1aded00c04fd8d5cd,dc=it-visions,dc=de>
```

Bitte beachten Sie, dass hierbei die Ansprache über `WKGUID` erfolgt und der dahinter angegebene GUID nicht der wirkliche GUID des Objekts ist. Auch die Standardcontainer erhalten bei der Installation eines Active Directory einen individuellen GUID; der `WKGUID` ist ein allgemein gültiger Alias.

Well Known Object	GUID
<code>CN=Deleted Objects</code>	<code>18E2EA80684F11D2B9AA00C04F79F805</code>
<code>CN=Infrastructure</code>	<code>2FBAC1870ADE11D297C400C04FD8D5CD</code>
<code>CN=LostAndFound</code>	<code>AB8153B7768811D1ADED00C04FD8D5CD</code>
<code>CN=System</code>	<code>AB1D30F3768811D1ADED00C04FD8D5CD</code>
<code>OU=Domain Controllers</code>	<code>A361B2FFFFD211D1AA4B00C04FD7D83A</code>
<code>CN=Computers</code>	<code>AA312825768811D1ADED00C04FD8D5CD</code>
<code>CN=Users</code>	<code>A9D1CA15768811D1ADED00C04FD8D5CD</code>

Tabelle 15.5: Liste der Well Known Objects

Bei einigen Operationen (z.B. beim Anlegen neuer Objekte) kommt es zu Problemen, wenn die Bindung über einen `WKGUID` erfolgt ist. Nutzen Sie in diesem Fall den `WKGUID` nur, um den DN des Objekts zu ermitteln. Binden Sie danach neu über den DN.



15.4.2 Testumgebung

In den folgenden Beispielen werden Konstanten für die anzusprechenden Verzeichnisdienst-Objekte verwendet. Sie müssen diese Konstanten anpassen, damit die Routinen in Ihrem Active Directory funktionieren können.

Verwendete Konstanten

```
' --- Pfade für Active Directory
Const DOMAIN As String = "DC=IT-Visions,DC=de"
Const SERVER As String = "LDAP://sonne2000/"
Const USER As String = "cn=H.Schwichtenberg"
Const OU As String = "ou=Basta"
Const GROUP As String = "cn=Sprecher"

Const LDAProot As String = SERVER & DOMAIN
Const LDAPusers As String = SERVER & "CN=users," & _
DOMAIN

Const LDAPhs As String = SERVER & USER & "," & OU & "," &
DOMAIN
Const LDAPbasta As String = SERVER & OU & "," & DOMAIN
Const LDAPsprecher As String = SERVER & GROUP & "," & _
```

```
OU & ", " & DOMAIN
```

```
Const ADMINUSER As String = "sonnensystem\admin"
Const ADMINPASSWORD As String = "egal"
```

Listing 15.7: Die in den folgenden Listings verwendeten Konstanten

15.4.3 Ermittlung der Partitionen

Zugriff auf rootDSE

Wenn der Domänen-Controller und der Domänenname nicht bekannt sind oder wenn vermieden werden soll, dass der Name von Server und Domäne entweder ins Programm hineinkodiert werden muss oder als Konfigurationsparameter dem Programm übergeben werden soll, dann ist es hilfreich, den Namen des Domänen-Controllers und der Domäne dynamisch ermitteln zu können. ADSI bietet diese Möglichkeit über den Zugriff auf *LDAP://rootDSE*. Das zurückgelieferte Verzeichnisobjekt bietet vier Attribute, deren Namen sich selbst erklären:

- ▶ »ServerName«
- ▶ »DefaultNamingContext«
- ▶ »ConfigurationNamingContext«
- ▶ »SchemaNamingContext«

Beispiel

Mit dem folgenden Programm ermittelt man die ADSI-Pfade der Partitionen in einem Active Directory und den Namen eines Domänen-Controllers.

```
' =====
' Beispiel aus FCL-Buch.sln
' Ausgabe der Partitionen eines ADs
' (C) Holger@Schwichtenberg.de
' =====
Sub ADS_partitionen()

    Dim de As DirectoryEntry

    ' --- Zugriff auf RootDSE (serverloses Binden)
    de = New DirectoryEntry("LDAP://rootDSE")

    ' --- Ausgabe des Servernamens
    out("Servername: " & de.Properties("Servername")(0))

    ' --- Ausgabe der Partitionen
```



```

out("defaultNamingContext: " & _
  de.Properties("defaultNamingContext")(0))
out("ConfigurationNamingContext: " & _
  de.Properties("ConfigurationNamingContext")(0))
out("SchemaNamingContext: " &
  de.Properties("SchemaNamingContext")(0))

```

End Sub

Listing 15.8: Ausgabe der Partitionen eines ADs [ADS.vb]

Ausgabe

Das Skript liefert in der Testumgebung die folgende Ausgabe:

```

1  Servername:
   CN=SONNE2000,CN=Servers,
   CN=Standardname-des-ersten-Standorts,CN=Sites,
   CN=Configuration,DC=IT-Visions,DC=de
2  defaultNamingContext:
   DC=IT-Visions,DC=de
3  ConfigurationNamingContext: CN=Configuration,DC=IT-
   Visions,DC=de
4  SchemaNamingContext: CN=Schema,CN=Configuration,DC=IT-
   Visions,DC=de
5  Pfad zum Computer-Container:
   CN=Computers,DC=IT-Visions,DC=de

```

15.4.4 Informationen über einen Benutzer

Ein Benutzer-Objekt im Active Directory (AD-Klasse »user«) besitzt zahlreiche Verzeichnisattribute. Ein Pflichtattribut, das alle Benutzer-Objekte besitzen, ist »SAMAccountName«, das den NT4-kompatiblen Anmeldennamen enthält.

Die folgende Tabelle zeigt weitere Verzeichnisattribute eines Benutzer-Objekts im Active Directory.

Name	Pflicht	Mehrwertig	Datentyp (Stringlänge)
"cn"	Ja	Nein	DirectoryString (1-64)
"nTSecurityDescriptor"	Ja	Nein	ObjectSecurityDescriptor (0-132096)
"objectCategory"	Ja	Nein	DN
"objectClass"	Ja	Ja	OID

Tabelle 15.6: Ausgewählte Attribute der Active Directory-Klasse »user«

Name	Pflicht	Mehr- wertig	Datentyp (Stringlänge)
"ObjectSid"	Ja	Nein	OctetString (0-28)
"SAMAccountName"	Ja	Nein	DirectoryString (0-256)
"accountExpires"	Nein	Nein	INTEGER8
"accountNameHistory"	Nein	Ja	DirectoryString
"badPwdCount"	Nein	Nein	INTEGER
"comment"	Nein	Nein	DirectoryString
"company"	Nein	Nein	DirectoryString (1-64)
"createTimeStamp"	Nein	Nein	GeneralizedTime
"department"	Nein	Nein	DirectoryString (1-64)
"description"	Nein	Ja	DirectoryString (0-1024)
"desktopProfile"	Nein	Nein	DirectoryString
"displayName"	Nein	Nein	DirectoryString (0-256)
"displayNamePrintable"	Nein	Nein	PrintableString (1-256)
"DistinguishedName"	Nein	Nein	DN
"division"	Nein	Nein	DirectoryString (0-256)
"employeeID"	Nein	Nein	DirectoryString (0-16)
"EmployeeType"	Nein	Nein	DirectoryString (1-256)
"expirationTime"	Nein	Nein	UTCTime
"FacsimileTelephoneNumber"	Nein	Nein	DirectoryString (1-64)
"givenName"	Nein	Nein	DirectoryString (1-64)
"homeDirectory"	Nein	Nein	DirectoryString
"HomeDrive"	Nein	Nein	DirectoryString
"homeMDB"	Nein	Nein	DN
"Initials"	Nein	Nein	DirectoryString (1-6)
"internationalISDNNumber"	Nein	Ja	NumericString (1-16)
"1"	Nein	Nein	DirectoryString (1-128)
"lastLogoff"	Nein	Nein	INTEGER8
"LastLogon"	Nein	Nein	INTEGER8
"logonCount"	Nein	Nein	INTEGER
"LogonHours"	Nein	Nein	OctetString
"logonWorkstation"	Nein	Nein	OctetString
"manager"	Nein	Nein	DN
"middleName"	Nein	Nein	DirectoryString (0-64)
"Mobile"	Nein	Nein	DirectoryString (1-64)

Tabelle 15.6: Ausgewählte Attribute der Active Directory-Klasse »user« (Fortsetzung)

Name	Pflicht	Mehrwertig	Datentyp (Stringlänge)
"name"	Nein	Nein	DirectoryString (1-255)
"objectGUID"	Nein	Nein	OctetString (16-16)
"ObjectVersion"	Nein	Nein	INTEGER
"otherFacsimileTelephone Number"	Nein	Ja	DirectoryString (1-64)
"OtherHomePhone"	Nein	Ja	DirectoryString (1-64)
"physicalDeliveryOffice- Name"	Nein	Nein	DirectoryString (1-128)
"PostalAddress"	Nein	Ja	DirectoryString (1-4096)
"postalCode"	Nein	Nein	DirectoryString (1-40)
"PostOfficeBox"	Nein	Ja	DirectoryString (1-40)
"profilePath"	Nein	Nein	DirectoryString
"SAMAccountType"	Nein	Nein	INTEGER
"scriptPath"	Nein	Nein	DirectoryString
"street"	Nein	Nein	DirectoryString (1-1024)
"streetAddress"	Nein	Nein	DirectoryString (1-1024)
"TelephoneNumber"	Nein	Nein	DirectoryString (1-64)
"title"	Nein	Nein	DirectoryString (1-64)
"userWorkstations"	Nein	Nein	DirectoryString (0-1024)
"whenChanged"	Nein	Nein	GeneralizedTime
"whenCreated"	Nein	Nein	GeneralizedTime
"wWWHomePage"	Nein	Nein	DirectoryString (1-2048)

Tabelle 15.6: Ausgewählte Attribute der Active Directory-Klasse »user« (Fortsetzung)

Beispiel

Die folgende Routine listet Informationen zu einem durch einen übergebenen LDAP-Pfad spezifizierten Benutzer-Objekt auf. Nach der Bindung des `DirectoryEntry`-Objekts an das Verzeichnisobjekt wird zunächst geprüft, ob das verwendete Objekt zur AD-Klasse »user« gehört. Nach der Ausgabe der Basisdaten werden vier benutzerspezifische Attribute (Anmeldename, Beschreibung, Telefonnummer, Stadt) ausgegeben.

Verzeichnisattribute ausgeben

```
' =====
' Beispiel aus FCL-Buch.sln
' Informationen zu einem Benutzer ausgeben
' (C) Holger@Schwichtenberg.de
' =====
```

```

Sub ADS_Benutzer_Lesen(ByVal pfad As String)

    Dim o As DirectoryEntry

    out("# Informationen über den Benutzer: " & pfad)

    ' --- Zugriff auf Eintrag
    o = New DirectoryEntry(pfad)
    If o.SchemaClassName <> "user" Then Exit Sub

    ' --- Basisdaten des Eintrags
    out("Name: " & o.Name)
    out("Pfad: " & o.Path)
    out("Klasse:" & o.SchemaClassName)
    out("GUID: " & o.Guid.ToString)
    out("Native GUID:" & o.NativeGuid)

    ' --- Ausgabe der Verzeichnisattribute
    out("NT4-Anmeldename: " & _
        o.Properties("SAMAccountName")(0))
    out("Beschreibung: " & getAtt(o, "Description"))
    out("Telefonnummer: " & getAtt(o, "telephoneNumber"))

    out("Stadt: " & getAtt(o, "l"))

End Sub

```

Listing 15.9: Informationen zu einem Benutzer ausgeben [ADS.vb]

15.4.5 Benutzer ändern

Verzeichnisattribute des Benutzers ändern

Nach der Bindung eines Benutzer-Objekts kann über die Klasse `DirectoryEntry` auch schreibend auf die Attribute zugegriffen werden. Zu beachten ist dabei, dass Attribute, die bereits einen Wert besitzen, anders zu behandeln sind als solche, die noch keinen Wert besitzen. Wie bereits in den allgemeinen Programmier-techniken in diesem Kapitel beschrieben, kann dies umgangen werden, indem die `PropertyValueCollection` vor dem Beschreiben mit `Clear()` gelöscht wird.

Beispiel

Im folgenden Beispiel wird die Stadt des Benutzers »H.Schwichtenberg« geändert.

```

' =====
' Beispiel aus FCL-Buch.sln
' Benutzerdaten ändern
' (C) Holger@Schwichtenberg.de
' =====

```

```

Sub ADS_Benutzer_aendern()

    Dim o As DirectoryEntry
    Dim p As PropertyValueCollection
    out("# Ändern des Benutzers: " & LDAPhs)

    ' --- Zugriff auf Eintrag
    o = New DirectoryEntry(LDAPhs)
    ' --- Zugriff auf Attribut
    p = o.Properties("l")
    ' --- bisherigen Eintrag löschen
    p.Clear()
    ' --- Stadt neu eintragen
    p.Add("Essen")
    ' --- Änderungen speichern
    o.CommitChanges()

End Sub

```

Listing 15.10: Benutzerdaten ändern [ADS.vb]

15.4.6 Organisationseinheit anlegen

Eine hervorstechende Eigenschaft des Active Directory im Vergleich zum Windows NT4-Benutzerverwaltungssystem ist es, beliebige Organisationsstrukturen in Form von Verzeichnis-Containern nachzubilden. Ein solcher Container heißt im AD Organisationseinheit (Klasse "OrganizationalUnit").

Beachten Sie bei der Anlage von Organisationseinheiten im Vergleich zur Anlage von Benutzern den anderen Klassennamen (»organizationalUnit«) im ersten Parameter und den anderen Attributnamen (OU) im zweiten Parameter bei Add(). Es gibt keine Pflichtattribute.

Anlegen von OUs

Beispiel

In dem folgenden Beispiel wird eine Organisationseinheit »BASTA« angelegt. Als freiwillige Attribute werden die Stadt (»l«) und die Beschreibung (»Description«) festgelegt.

```

' =====
' Beispiel aus FCL-Buch.sln
' Anlegen einer Organisationseinheit im AD
' (C) Holger@Schwichtenberg.de
' =====
Sub ADS_OUanlegen()
    Dim r As DirectoryEntry
    Dim o As DirectoryEntry

```

```

out("# Anlegen der OU: " & LDAPbasta)

' --- Zugriff auf Container
r = New DirectoryEntry(LDAProot)
' --- Untereintrag anlegen
o = r.Children.Add(OU, "organizationalUnit")

' --- freiwillige Attribute setzen
o.Properties("l").Add("Frankfurt")
o.Properties("Description")._
.Add("Entwickler-Fachkonferenz")
' --- Änderungen speichern
o.CommitChanges()
out("OU angelegt! " & o.Path)
End Sub

```

Listing 15.11: Anlegen einer Organisationseinheit im Active Directory

15.4.7 Organisationseinheit löschen

OU löschen Eine Organisationseinheit kann nur entfernt werden, wenn sie keine Unterobjekte mehr enthält. Daher sollte für diese Aufgabe die Unterroutine `ADS_Loeschen_Rekursiv()` verwendet werden.

Beispiel

Im folgenden Beispiel wird die Organisationseinheit »BASTA« gelöscht.

```

' =====
' Beispiel aus FCL-Buch.sln
' Löschen einer OU
' (C) Holger@Schwichtenberg.de
' =====
Sub ADS_ouloeschen()
    Dim o As DirectoryEntry
    Dim c As DirectoryEntries

    out("# Löschen der OU: " & LDAPbasta)

    ' --- Zugriff auf IADS
    o = New DirectoryEntry(LDAProot)
    ' --- Zugriff auf IADSContainer
    c = o.Children
    ' --- Objekt löschen!
    c.Remove(c.Find(OU))

    out("OU gelöscht!")
End Sub

```

Listing 15.12: Löschen einer OU [ADS.vb]

15.4.8 Benutzer anlegen

Da das Anlegen eines Objekts vom übergeordneten Container ausgeht, muss im ersten Schritt der Container an `DirectoryEntry` gebunden werden. Die Erzeugung eines neuen Objekts erfolgt mit `Add()`, wobei im ersten Parameter der RDN des neuen Objekts und im zweiten Parameter der AD-Klassenname »user« angegeben sind.

»user«-
Verzeichnisobjekt
anlegen

Das Setzen der Eigenschaft »SAMAccountName« ist Pflicht. Sofern der Property Cache nicht ausgeschaltet wurde, muss nach dem Setzen aller Eigenschaften `CommitChanges()` ausgeführt werden, da sonst das Benutzer-Objekt nicht angelegt wird.

`CommitChanges()`

Im Standard ist ein neues Benutzerkonto im Active Directory deaktiviert. Die einfachste Möglichkeit zur Aktivierung ist der Zugriff auf das Attribut `AccountDisabled` in der COM-Schnittstelle `IADsUser`.

Beim Anlegen eines Benutzers muss der Impersonifizierungsmodus verwendet werden, selbst wenn der Benutzer, der die Anwendung startet, die notwendigen Rechte hätte. Dies war unter Classic-ADSI nicht der Fall und es stellt sich die Frage, ob dies unter .NET ein Bug oder ein Feature ist.



Bug oder Feature?

Beispiel

In der folgenden Routine wird ein Benutzerkonto »H.Schwichtenberg« mit NT4-Anmeldename »HSch« angelegt. Als optionales Attribut wird nur die Stadt ("1") gesetzt.

```
' =====
' Beispiel aus FCL-Buch.sln
' Anlegen eines User-Objekts im Active Directory
' (C) Holger@Schwichtenberg.de
' =====
Sub ADS_Benutzer_anlegen()
    Dim o As DirectoryEntry
    Dim c As DirectoryEntries

    out("# Anlegen des Benutzerkontos: " & USER)

    ' --- Zugriff auf IADS
    o = New DirectoryEntry(LDAPbasta, _
        ADMINUSER, ADMINPASSWORD)
```

```

' --- Zugriff auf IADSContainer
c = o.Children()

' --- Neues Objekt erzeugen
o = c.Add(USER, "user")
' --- Verzeichnisattribute festlegen
o.Properties("SAMAccountName").Add("HSch")
o.Properties("l").Add("Essen-Byfang")
o.CommitChanges()
' --- Konto aktivieren
o.NativeObject.AccountDisabled = False
o.CommitChanges()
End Sub

```

Listing 15.13: Anlegen eines User-Objekts im Active Directory [ADS.vb]

15.4.9 Benutzerkennwort setzen

**Kennwort
festlegen mit
SetPassword()**

Das Kennwort eines Benutzerkontos kann erst gesetzt werden, nachdem das Benutzerkonto im Verzeichnisdienst angelegt wurde. Auch bei dieser Operation ist unter .NET die Impersonifizierung notwendig. Hier kommt der Methodenaufruf mit `Invoke()` zum Einsatz, da die .NET-Klassen keine Möglichkeit bieten, das Kennwort zu ändern. Mit `Invoke()` wird die in der COM-Schnittstelle `IADSIUser` definierte Methode `SetPassword()` aufgerufen. Als zweiter Parameter bei `Invoke()` ist das neue Kennwort in Form eines Strings zu übergeben.

```
o.Invoke("setpassword", "NEUES_KENNWORT")
```

Beispiel

In der folgenden Routine wird das Kennwort für den zuvor angelegten Benutzer auf »12345678« festgelegt.

```

' =====
' Beispiel aus FCL-Buch.sln
' Kennwort für ein AD-Benutzerkonto setzen
' (C) Holger@Schwichtenberg.de
' =====
Sub ADS_BenutzerPwd_aendern()
    Dim o As DirectoryEntry

    out("# Kennwortänderung für Benutzer " & LDAPhs)

    ' --- Zugriff auf Eintrag
    o = New DirectoryEntry(LDAPhs, _
        ADMINUSER, ADMINPASSWORD)
    o.Invoke("setpassword", "12345678")

```



```

' --- Basisdaten des Eintrags
out("Kennwort für " & o.Name & " geändert!")
End Sub

```

Listing 15.14: Kennwort für ein AD-Benutzerkonto setzen [ADS.vb]

15.4.10 Benutzer löschen

Um einen Benutzer zu löschen, gibt es zwei Alternativen:

Alternative 1

- ▶ Bindung an den Container über den DN des Containers
- ▶ Wechsel zur `DirectoryEntries`-Collection über das Attribut `Children`
- ▶ Suche nach dem »user«-Objekt in der Collection über den RDN des Benutzers mit der Methode `Find()`
- ▶ Löschen des Benutzers durch Ausführung von `Remove()` auf dem Container unter Angabe des `DirectoryEntry`-Objekts für den Benutzer

Alternative 1

```

' =====
' Beispiel aus FCL-Buch.sln
' Löschen eines Benutzers (Variante 1)
' (C) Holger@Schwichtenberg.de
' =====
Sub ADS_Benutzer_loeschen()
    Dim o As DirectoryEntry
    Dim c As DirectoryEntries
    Dim u As DirectoryEntry

    out("# Löschen des Benutzerkontos: " & LDAPhs)

    ' --- Zugriff auf IADS
    o = New DirectoryEntry(LDAPbasta)
    ' --- Zugriff auf IADSContainer
    c = o.Children
    ' --- Suche nach dem Benutzer
    u = c.Find(USER)
    ' --- Objekt löschen!
    c.Remove(u)
    ' --- Bestätigung
    out("Benutzer gelöscht!")
End Sub

```

Listing 15.15: Löschen eines Benutzers (Variante 1) [ADS.vb]

Alternative 2

- Alternative 2**
- Bindung an das »user«-Objekt über den DN des Benutzers
 - Wechsel zum übergeordneten Objekt über Parent
 - Wechsel zur DirectoryEntries-Collection des übergeordneten Objekts über das Attribut Children
 - Löschen des Benutzers durch Ausführung von Remove() auf dem Container unter Angabe des DirectoryEntry-Objekts für den Benutzer

```
' =====
' Beispiel aus FCL-Buch.sln
' Löschen eines Benutzers (Variante 2)
' (C) Holger@Schwichtenberg.de
' =====

Sub ADS_Benutzer_loeschen2()
    Dim o As DirectoryEntry
    Dim c As DirectoryEntries

    out("# Löschen des Benutzerkontos: " & LDAPhs)

    ' --- Zugriff auf IADS
    o = New DirectoryEntry(LDAPhs)
    ' --- Zugriff auf IADSContainer
    c = o.Parent.Children
    ' --- Objekt löschen!
    c.Remove(o)
    ' --- Bestätigung
    out("Benutzer gelöscht!")
End Sub
```

Listing 15.16: Löschen eines Benutzers (Variante 2) [ADS.vb]



Zum Löschen eines Benutzers kann auch die bereits in diesem Kapitel vorgestellte allgemeine Routine ADS_loeschen_rekursiv() verwendet werden.

15.4.11 Benutzer umbenennen

- Rename()** Für das Umbenennen eines Verzeichnisdienstobjekts bietet die Klasse DirectoryEntry mit der Methode Rename() ein sehr einfaches Verfahren. Unter Classic-ADSI musste dazu die IADsContainer-Methode MoveHere() verwendet werden.

Beispiel

Im folgenden Beispiel wird das Benutzerkonto »H.Schwichtenberg« in »Holger.Schwichtenberg« umbenannt.

```
' =====
' Beispiel aus FCL-Buch.sln
' Umbenennen eines AD-Benutzerkontos
' (C) Holger@Schwichtenberg.de
' =====
Sub ADS_benutzer_umbenennen()
    Dim de As DirectoryEntry

    out("# Umbenennen des Benutzerkontos: " & LDAPhs)

    ' --- Zugriff auf Benutzer
    de = New DirectoryEntry(LDAPhs)
    de.UsePropertyCache = False
    ' --- Verzeichnisnamen ändern
    de.Rename("cn=Holger.Schwichtenberg")
    ' --- Änderungen speichern
    de.CommitChanges()
End Sub
```

Listing 15.17: Umbenennen eines AD-Benutzerkontos [ADS.vb]

15.4.12 Benutzer verschieben

Als Äquivalent zu der COM-Methode `IADSTContainer.MoveHere()` gibt es in der FCL-Klasse `DirectoryEntry` die Methode `MoveTo()`. Sie verschiebt ein Verzeichnisobjekt in einen anderen Container. Der Zielcontainer ist in Form eines zweiten `DirectoryEntry`-Objekts als Parameter zu übergeben.

MoveTo()

Beispiel

In der folgenden Routine wird der Benutzer »H.Schwichtenberg« aus der Organisationseinheit »BASTA« in den Standard-Benutzer-Container »Users« verschoben.

```
' =====
' Beispiel aus FCL-Buch.sln
' Verschieben eines AD-Benutzerkontos
' (C) Holger@Schwichtenberg.de
' =====
Sub ADS_benutzer_verschieben()
    Dim de As DirectoryEntry
    Dim con As DirectoryEntry

    out("# Verschieben des Benutzerkontos: " & LDAPhs)
```

```

' --- Zugriff auf Benutzer
de = New DirectoryEntry(LDAPhs)
' --- Zugriff auf neuen Container
con = New DirectoryEntry(LDAPusers)

' --- Verzeichnisnamen ändern
de.MoveTo(con)
End Sub

```

Listing 15.18: Verschieben eines AD-Benutzerkontos [ADS.vb]

15.4.13 Gruppe anlegen

Verzeichnisklasse
»group«

Das Anlegen einer Gruppe ist fast analog zum Anlegen eines Benutzerkontos. Unterschiede sind:

- Die zu erzeugende Active Directory-Klasse ist »group« statt »user«.
- Es darf kein Kennwort vergeben werden.

Beispiel

In der folgenden Routine wird eine Gruppe »Sprecher« erzeugt.

```

' =====
' Beispiel aus FCL-Buch.sln
' Gruppe anlegen im AD
' (C) Holger@Schwichtenberg.de
' =====
Sub ADS_Gruppe_anlegen()
    Dim r As DirectoryEntry
    Dim gr As DirectoryEntry

    out("# Anlegen der Gruppe: " & LDAPsprecher)

    ' --- Bindung an Container
    r = New DirectoryEntry(LDAPbasta)
    ' --- Gruppenobjekt erzeugen
    gr = r.Children.Add(GROUP, "group")
    ' --- Pflichtattribute setzen
    gr.Properties("SAMAccountName").Add("Sprecher")
    ' --- Optionale Attribute setzen
    gr.Properties("Description")._
        .Add("Redner auf der BASTA")
    ' --- Änderungen speichern
    gr.CommitChanges()
    ' --- Ausgabe
    out("Gruppe angelegt: " & gr.Path)
End Sub

```

Listing 15.19: Gruppe anlegen im AD [ADS.vb]

15.4.14 Benutzer einer Gruppe hinzufügen

Um einen Benutzer zu einer Gruppe hinzuzufügen, bindet man eine Instanz der Klasse `DirectoryEntry` an das Gruppen-Objekt und führt dort die in der COM-Schnittstelle `IADsGroup` definierte Methode `Add()` aus. Als Parameter ist der LDAP-Pfad des hinzuzufügenden Benutzer-Objekts anzugeben.

Add()

Für diese Operation gibt es noch keine Methode in einer .NET-Klasse.



Beispiel

In der folgenden Routine wird der zuvor angelegte Benutzer »H.Schwichtenberg« der angelegten Gruppe »Sprecher« hinzugefügt.

```
' =====
' Beispiel aus FCL-Buch.sln
' AD-Benutzer einer AD-Gruppe hinzufügen
' (C) Holger@Schwichtenberg.de
' =====
Sub ADS_Benutzer_zu_Gruppe()
    Dim o As DirectoryEntry
    out("# Hinzufügen des Benutzers " & LDAPhs & _
        " zu Gruppe: " & LDAPsprecher)
    ' --- Zugriff auf Eintrag
    o = New DirectoryEntry(LDAPsprecher)
    ' --- IADsGroup::Add() aufrufen
    o.Invoke("Add", LDAPhs)
    ' --- Bestätigung
    out("Benutzer zu Gruppe hinzugefügt!")
End Sub
```

Listing 15.20: AD-Benutzer einer AD-Gruppe hinzufügen [ADS.vb]

15.4.15 Benutzer aus Gruppe entfernen

Um einen Benutzer aus einer Gruppe zu entfernen, muss bis auf ein Wort der gleiche Programmcode wie beim Hinzufügen ausgeführt werden: Statt `Add()` wird `Remove()` via `Invoke()` aufgerufen.

Remove()

Beispiel

In der folgenden Routine wird der Benutzer »H.Schwichtenberg« aus der Gruppe »Sprecher« entfernt.

```

' =====
' Beispiel aus FCL-Buch.sln
' AD-Benutzer aus einer AD-Gruppe entfernen
' (C) Holger@Schwichtenberg.de
' =====
Sub ADS_Benutzer_aus_Gruppe()
    Dim o As DirectoryEntry
    out("# Entfernen des Benutzers " & LDAPhs & _
        " aus Gruppe: " & LDAPsprecher)
    ' --- Zugriff auf Eintrag
    o = New DirectoryEntry(LDAPsprecher)
    ' --- IADSGroup::Add() aufrufen
    o.Invoke("Remove", LDAPhs)
    ' --- Bestätigung
    out("Benutzer aus Gruppe entfernt!")
End Sub

```

Listing 15.21: AD-Benutzer aus einer AD-Gruppe entfernen [ADS.vb]

15.4.16 Gruppe löschen

Löschen einer Benutzergruppe

Das Löschen einer Gruppe geschieht mit den gleichen Befehlen wie das Löschen eines Benutzers. Beim Löschen einer Gruppe werden auch die Gruppenzugehörigkeiten gelöscht. Die Benutzerkonten, die zu der Gruppe gehörten, bleiben jedoch erhalten.

Beispiel

In der folgenden Routine wird die Gruppe »Sprecher« gelöscht.

```

' =====
' Beispiel aus FCL-Buch.sln
' Löschen einer Gruppe
' (C) Holger@Schwichtenberg.de
' =====
Sub ADS_Gruppe_loeschen()
    Dim o As DirectoryEntry
    Dim c As DirectoryEntries
    out("# Löschen der Gruppe: " & LDAPsprecher)
    ' --- Zugriff auf IADS
    o = New DirectoryEntry(LDAPsprecher)
    ' --- Zugriff auf IADSTContainer
    c = o.Parent.Children
    ' --- Objekt löschen!
    c.Remove(o)
    ' --- Bestätigung
    out("Gruppe gelöscht!")
End Sub

```

Listing 15.22: Löschen einer Gruppe [ADS.vb]

15.4.17 Containerinhalt auflisten

Es gibt im Active Directory verschiedene Container-Typen. Eine Organisationseinheit ist nur ein möglicher Typ von Containern. Die Vorgehensweise ist in allen Fällen jedoch gleich:

Inhalt eines Containers

- ▶ Bindung an das Container-Objekt über `DirectoryEntry()`
- ▶ Wechsel zur `DirectoryEntries-Collection` über das Attribut `Children`
- ▶ Iteration mit `For...Each` über die `DirectoryEntries-Collection`

Beispiel

Die folgende Routine listet alle Unterobjekte des Standardcontainers »Users« auf.

```
' =====
' Beispiel aus FCL-Buch.sln
' Liste der Unterobjekte eines Containers
' (C) Holger@Schwichtenberg.de
' =====
Sub ADS_ContainerAuflisten_Benutzer()
    Dim o As DirectoryEntry
    Dim c As DirectoryEntries
    out("# Inhalt des Containers: " & LDAPusers)
    ' --- Zugriff auf IADS
    o = New DirectoryEntry(LDAPusers)
    ' --- Wechsel zu IADSContainer
    c = o.Children
    ' --- Schleife über alle Containerelemente
    For Each o In c
        out(o.Name)
    Next
End Sub
```

Listing 15.23: Liste der Unterobjekte eines Containers [ADS.vb]

Filtern von Klassen in einem Container

Ebenso wie im klassischen ADSI kann auch im .NET-ADSI der Containerinhalt durch einen Filter auf bestimmte Klassennamen eingeschränkt werden. Dazu muss vor der Schleife über den Containerinhalt die Collection `SchemaFilter` im `DirectoryEntries`-Objekt mit denjenigen Klassennamen gefüllt werden, auf die eingeschränkt werden soll.

Ergebnismenge einschränken

```
' --- Setzen des Filters
Dim snc As SchemaNameCollection
snc = c.SchemaFilter
snc.Add("User")
snc.Add("Group")
```

15.4.18 Computerkonto anlegen

**Verzeichnisdienst-
Klasse
»computer«**

Das Anlegen eines Computerkontos im Active Directory ist dem Anlegen eines Benutzers sehr ähnlich. Dies ist auch nicht verwunderlich, wenn man weiß, dass die Active Directory-Klasse »computer« von der Klasse »user« abgeleitet ist. Die Vererbungshierarchie im Active Directory ist

```
top->person->organizationalPerson->user->Computer
```

Beispiel

In der folgenden Routine wird in dem Standardcontainer »Computers« ein neues Computerkonto mit dem Namen »NeuerComputer« angelegt. Wenn dieses Computerkonto schon vorhanden ist, wird es mit `ADSI_Loeschen_Rekursiv()` vorher gelöscht.

```
' =====
' Beispiel aus FCL-Buch.sln
' Computerkonto anlegen im AD
' (C) Holger@Schwichtenberg.de
' =====
Sub ADS_Computerkonto_anlegen()
    Dim con As DirectoryEntry
    Dim c As DirectoryEntry

    Const DNC = "dc=IT-Visions,dc=de"
    ' DefaultNamingContext
    Const COMPUTER_CONTAINER = "LDAP://cn=computers," _
    & DNC
    Const COMPUTER_RDN = "cn=NeuerComputer"
    Const COMPUTER_DN = "LDAP://" & COMPUTER_RDN & _
    ",cn=computers," & DNC

    out("# Anlegen des Computerkontos : " & COMPUTER_DN)

    ' --- Löschen des Computerkontos,
    ' --- wenn es bereits vorhanden ist
    If DirectoryEntry.Exists(COMPUTER_DN) Then
        out("Computerkonto ist bereits vorhanden!")
        ADSI_Loeschen_Rekursiv(COMPUTER_DN)
    End If
```



```

' --- Bindung an Computer-Container
con = New DirectoryEntry(COMPUTER_CONTAINER)

' --- Gruppenobjekt erzeugen
c = con.Children.Add(COMPUTER_RDN, "Computer")
' --- Pflichtattribute setzen
c.Properties("SAMAccountName").Add("NEUERCOMPUTER")
' --- Optionale Attribute setzen
c.Properties("Description").Add("Mein neuer Computer")
' --- Änderungen speichern
c.CommitChanges()
' --- Ausgabe
out("Computerkonto angelegt: " & c.Path)
End Sub

```

Listing 15.24: Computerkonto anlegen im AD [ADS.vb]

15.4.19 Computerkonto löschen

Um ein Computerkonto zu löschen, kann die Hilfsroutine `ADSI_Loeschen_Rekursiv()` verwendet werden.

**Computerkonto
entfernen**

```
ADSI_Loeschen_Rekursiv(COMPUTER_DN)
```

Beispiel

Das Löschen eines Computerkontos wird in dem Beispiel zum Anlegen eines Computerkontos mitbehandelt.

15.4.20 Suche im Active Directory

Das LDAP-Protokoll unterstützt neben der schrittweisen Iteration durch Verzeichnis-Container auch die gezielte Suche nach Verzeichnisobjekten. Im Classic-ADSI wurde diese Funktionalität durch einen OLE DB-Provider gekapselt. Dieser steht grundsätzlich natürlich auch in ADO.NET über den Managed Provider für OLE DB noch zur Verfügung. Allerdings bietet der Namespace `System.DirectoryServices` eine elegantere Möglichkeit zur Ausführung von LDAP-Suchanfragen.

Suche via LDAP

Eine LDAP-Suchanfrage hat gegenüber den bisher vorgestellten Möglichkeiten folgende Vorteile:

- ▶ Die Menge der Verzeichniseinträge kann bereits auf der Serverseite eingeschränkt werden.
- ▶ Die Abfrage kann Einträge aus mehreren Ebenen im Verzeichnisbaum zurückgeben.



Während der OLE DB-Provider für ADSI Anfragen sowohl in LDAP-Query-Syntax als auch als SQL-Befehle unterstützt, können mit den in der FCL eingebauten Klassen nur LDAP-Query-Syntax-Anfragen gestellt werden.

Ebenso wie mit dem OLE DB-Provider können auch mit den FCL-Klassen nur LDAP-fähige Verzeichnisdienste abgefragt werden.

LDAP-Query-Syntax

LDAP-Syntax Die LDAP-Query-Syntax ist in [RFC1960] und [RFC2254] spezifiziert. Die allgemeine Syntax lautet:

```
"Start; [Filter]; Attribute [; Scope]"
```

Dabei ist:

- ▶ **Start** ein LDAP-Pfad inkl. *LDAP://*. Der Pfad kann sowohl in Little Endian- als auch in Big Endian-Form angegeben werden (siehe auch Erläuterungen in Anhang B).
Beispiel: *LDAP://sonne2000/dc=It-Visions,dc=DE*
- ▶ **Filter** eine Bedingung in umgekehrt polnischer Notation (UPN), auch Postfix-Notation. Diese Notation zeichnet sich dadurch aus, dass die Operatoren am Anfang stehen. Der Sternoperator (*»*«*) ist als Platzhalter erlaubt.
Beispiel für eine Und-Verknüpfung:
(&(objectclass=user)(name=h))*
- ▶ **Attribute** eine durch Kommata getrennte Attributliste der gewünschten Verzeichnisattribute, die in die Tabelle aufgenommen werden sollen. Diese Angabe ist nicht optional und der Sternoperator (*»*«*) wie bei SQL ist nicht erlaubt.
Beispiel: *AdsPath,Name,SAMAccountname*
- ▶ **Scope** eine der in der folgenden Tabelle genannten Konstanten.

Konstante (LDAP-Syntax)	Erläuterung
BASE	Es wird nur der angegebene Eintrag gesucht. Die Ergebnismenge umfasst keinen oder einen Datensatz.
ONELEVEL	Es wird in den Einträgen gesucht, die dem angegebenen Eintrag untergeordnet sind.

Tabelle 15.7: Suchtiefen bei LDAP-Queries

Konstante (LDAP-Syntax)	Erläuterung
SUBTREE	Es werden alle darunter liegenden Ebenen durchsucht.

Tabelle 15.7: Suchtiefen bei LDAP-Queries (Fortsetzung)

Beispiel für eine Abfrage

Die folgende Abfrage sucht im ganzen Active Directory alle Benutzerkonten, deren Namen mit dem Buchstaben »H« beginnen.

```
<LDAP://sonne2000/dc=IT-Visions,dc=DE>;
(&(objectclass=user)(name=h*));
adspath,SAMAccountname;subtree
```

**Beispiel für eine
LDAP-Suchanfrage**

Eine Abfrage, die nur aus der Bedingung class= besteht, funktioniert nicht. Um alle Verzeichnisobjekte zurückzuliefern, muss der Sternoperator auf ein anderes Attribut angewendet werden.*



Ausführung einer Abfrage

Eine LDAP-Abfrage wird mit FCL-Klassen in folgenden Schritten ausgeführt:

Suche definieren

- ▶ Instanziierung der Klasse `DirectorySearcher`
- ▶ Festlegung des Ausgangspunkts der Anfrage durch Zuweisung eines Zeigers auf ein `DirectoryEntry`-Objekt, das an den Ausgangspunkt gebunden ist, an das Attribut `SearchRoot`.
- ▶ Setzen des Filter-Teils der LDAP-Abfrage im Attribut `Filter`
- ▶ Festlegung der Attribute durch Füllen der Collection `PropertiesToLoad`
- ▶ Festlegung des Scopes in dem Attribut `SearchScope`
- ▶ Starten der Anfrage durch die Methode `FindAll()`
- ▶ `FindAll()` liefert eine Collection vom Typ `SearchResultCollection` zurück.
- ▶ Die `SearchResultCollection` enthält einzelne `SearchResult`-Objekte.
- ▶ Von einem `SearchResult`-Objekt kann man entweder über die `ResultsPropertyCollection` lesend auf die abgefragten Attribute zugreifen oder aber man lässt sich von der Methode `GetDirec`

Suche starten

**Ergebnis
auswerten**

DirectoryEntry() ein DirectoryEntry-Objekt für den gefundenen Verzeichniseintrag liefern. Das so ermittelte DirectoryEntry-Objekt ermöglicht auch den Schreibzugriff.

Beispiel

Suchbeispiel In dem folgenden Beispiel werden im ganzen Active Directory alle Benutzerkonten gesucht, deren Verzeichnisname mit dem Buchstaben »H« beginnt. Für diese Objekte werden der Verzeichnisname und die Stadt ausgegeben. Außerdem wird für alle diese Objekte die Beschreibung auf den Text »Benutzer mit 'H'« geändert.

```
' =====
' Beispiel aus FCL-Buch.sln
' Ausführen einer LDAP-Suche im AD
' (C) Holger@Schwichtenberg.de
' =====
Sub ADS_suche()
    Dim suche As DirectorySearcher
    Dim ergebnisliste As SearchResultCollection
    Dim ergebnis As SearchResult

    out("# Suchanfrage im ADS")

    ' --- Suchanfrage
    Const suchanfrage = "(&(objectclass=user)(cn=h*))"
    ' --- Instanziierung der Klasse
    suche = New DirectorySearcher()
    ' --- Festlegung des Ausgangspunkts
    suche.SearchRoot = New DirectoryEntry(LDAProot)
    ' --- Festlegung der LDAP-Query
    suche.Filter = suchanfrage

    ' --- Ort einbeziehen
    suche.PropertiesToLoad.Add("l")
    suche.PropertiesToLoad.Add("Description")
    ' --- Suchtiefe festlegen
    suche.SearchScope = SearchScope.Subtree
    ' --- Suche starten
    ergebnisliste = suche.FindAll()
    ' --- Ergebnismenge ausgeben
    For Each ergebnis In ergebnisliste
        ' --- Ergebnis lesen
        out(getAtt(ergebnis, "Name") & " wohnt In " & _
            getAtt(ergebnis, "l"))
        out(getAtt(ergebnis, "Description"))
        ' --- Ergebnis verändern
        Dim de As DirectoryEntry = ergebnis.GetDirectoryEntry
```

```
de.Properties("description").Clear()  
de.Properties("description").Add("Benutzer mit 'H'")  
de.CommitChanges()  
Next  
End Sub
```

Listing 15.25: Ausführen einer LDAP-Suche im AD [ADS.vb]

15.5 Verzeichnisdienstprogrammierung mit dem WinNT-Provider

Der ADSI-Provider »WinNT« dient nicht nur dem Zugriff auf den Windows NT 4.0-Verzeichnisdienst, sondern auch dem Zugriff auf die lokalen Benutzerdatenbanken folgender Systeme:

WinNT-Provider

- ▶ Windows 2000 Professional
- ▶ Windows XP Home
- ▶ Windows XP Professional
- ▶ Windows 2000 Server ohne Active Directory
- ▶ Windows .NET Server ohne Active Directory

Im Gegensatz zum LDAP-Provider ermöglicht der WinNT-Provider auch die Verwaltung von Ressourcen wie Druckern, Diensten, Benutzersitzungen und geöffneten Dateien. Da das .NET-ADSI aber keine Kapselung der im klassischen ADSI vorhandenen Schnittstellen bietet, ist hier stets der Aufruf des `NativeObject` geboten.

Erweiterte Möglichkeiten

Mit dem WinNT-Provider kann man auch auf das Active Directory zugreifen: Dabei besitzt man allerdings über zahlreiche Verzeichnisattribute keine Kontrolle.

15.5.1 Grundlagen zum NT4-Verzeichnisdienst

Mit dem WinNT-Provider können folgende Objekte im NT-Verzeichnis verwaltet werden:

WinNT-Verzeichnisobjekte

- ▶ NT-Domänen
- ▶ Computer
- ▶ Benutzer (sowohl Domänen- als auch lokale Benutzer)

- ▶ Benutzergruppen (sowohl Gruppen in Domänen als auch Gruppen auf Computern)
- ▶ NT-Dienste
- ▶ Druckerwarteschlangen
- ▶ Druckaufträge
- ▶ Verzeichnisfreigaben
- ▶ Benutzersitzungen
- ▶ Dateien, die in Benutzung sind



Verzeichnisdienst oder nicht Verzeichnisdienst

Ob Windows NT 4.0 überhaupt einen Verzeichnisdienst hat, ist eine Frage, an der sich die Geister scheiden, weil die Verwaltung von Benutzern und Ressourcen sehr »flach« ist. Es gibt nur wenige Container und es können keine eigenen (Unter-)Container angelegt werden. Das Schema ist nicht erweiterbar. In diesem Buch wird zur Vereinfachung dennoch der Begriff Verzeichnisdienst verwendet.

WinNT-Objektmodell

Wo kommt das Schema her?

Die nachfolgende Grafik stellt die hierarchische Anordnung der Objekte im WinNT-Verzeichnis dar. Dies ist das Basis-Objektmodell des WinNT-Verzeichnisdienstes. Der WinNT-Provider ist insofern eine Ausnahme, als dieses Basis-Objektmodell nicht durch den Verzeichnisdienst, sondern durch den Provider selbst realisiert wird. NT4 besitzt kein Schema. Der WinNT-Provider kapselt jedoch die NT4-API-Funktionen so, dass ADSI-Clients ein Schema sehen.

Objekthierarchie

»Domain«-Objekte enthalten »Computer«, »User«- und »Group«-Objekte. Ein Computer-Objekt enthält »User«, »Group«, »Service«, »File service«, »PrintQueue«- und »PrintJob«-Objekte. »Fileservice«-Objekte enthalten wiederum »FileShare«, »Session«- und »Resource«-Objekte. Auffallend ist, dass »User«- und »Group«-Objekte direkt unter der Domain und auch innerhalb jedes »Computer«-Containers zu finden sind.

»Service«-Objekte sind in den meisten Fällen Blatt-Objekte, ein »FileService« ist jedoch auch selbst ein Container.

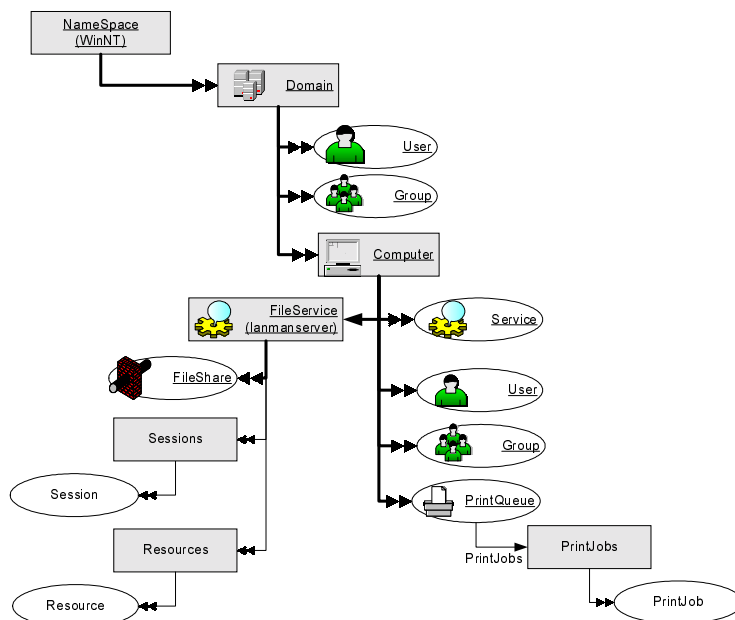


Abbildung 15.6: Objekte im WinNT-Namespace

Identifikation und Bindung

Der WinNT-Provider benutzt die Namespace-ID »WinNT«, wobei Groß- und Kleinschreibung exakt zu beachten sind. Der provider-spezifische Teil entspricht der Form */obersteEbene/ebene1/ebene2*.

**Objekt-
identifikation**

WinNT-Objekttyp	ADSI-Pfad
WinNT-Root	WinNT://
Domäne	WinNT://DomainName
Computer	WinNT://DomainName/ComputerName oder WinNT://ComputerName
Domänen-Benutzer (entsprechend für Gruppen)	WinNT://DomainName/BenutzerName oder WinNT://PDCName/BenutzerName
Computer-Benutzer (entsprechend für Gruppen)	WinNT://ComputerName/BenutzerName
NT-Dienste	WinNT://ComputerName/DienstName
Druckerwarteschlangen	WinNT://ComputerName/DruckerName

Tabelle 15.8: ADSI-Pfade im WinNT-Provider

WinNT-Objekttyp	ADSI-Pfad
Verzeichnisfreigabe	WinNT:// ComputerName/lanmanserver/Freigabe- bename
WinNT-Schema	WinNT://ComputerName/Schema oder WinNT://DomainName/Schema

Tabelle 15.8: ADSI-Pfade im WinNT-Provider (Fortsetzung)

Angabe des
Klassennamens

Da Doppeldeutigkeiten möglich sind, weil eine Domain einen Computer und einen Benutzer mit dem gleichen Namen enthalten kann, ist es möglich, hinter dem ADSI-Pfad durch ein Komma getrennt den Klassennamen anzugeben:

- Der Computer mit dem Namen »Sonne«

WinNT://Sonnensystem/Sonne,computer

- Der Benutzer mit dem Namen »Sonne«

WinNT://Sonnensystem/Sonne,user

15.5.2 Benutzerliste ausgeben

Iteration über
Domänen-/
Computer-
container

Um die Benutzer eines Computers oder einer Domäne aufzulisten, ist eine Iteration über den jeweiligen Container notwendig. Der Container enthält aber nicht nur »User«-Objekte, sondern auch andere Objekte (siehe Objekthierarchie). Um die Ergebnismenge auf die Klassen »User« und »Group« zu beschränken, muss die in dem Attribut SchemaFilter enthaltene SchemaNameCollection gefüllt werden.

```
' --- Demo-Daten für WinNT-Provider
Const COMPUTER = "mars"

' =====
' Beispiel aus FCL-Buch.sln
' Liste der Benutzer und Gruppen
' (C) Holger@Schwichtenberg.de
' =====

Sub WINNT_Benutzer_Liste()
    Dim o As DirectoryEntry
    Dim c As DirectoryEntries

    Dim iadscontainer As ActiveDs.IADsContainer

    outtitle("Liste der Domänenbenutzer")
```



```

' --- Zugriff auf IADs
o = New DirectoryEntry("WinNT://" & COMPUTER)

' --- Wechsel auf Kinderliste
c = o.Children

' --- Setzen des Filters
Dim snc As SchemaNameCollection
snc = c.SchemaFilter
snc.Add("User")
snc.Add("Group")

' --- Liste
For Each o In c
    out(o.Name)
Next

End Sub

```

Listing 15.26: Ausgabe einer Liste aller Benutzer und Gruppen

15.5.3 Benutzer verwalten

Das Anlegen eines Benutzers mit dem WinNT-Provider ist dem Anlegen eines Benutzers mit dem LDAP-Provider sehr ähnlich. Allerdings gibt es keine Pflichtattribute, die zu setzen sind. Das Löschen ist völlig analog und kann daher mit der allgemeinen Hilfsroutine `ADSI_Loeschen_Rekursiv("Benutzer_DN")` erfolgen.

**Benutzer anlegen
und löschen**

```

' --- Demo-Daten für WINNT-Provider
Const COMPUTER = "mars"
Const BENUTZER = "HSch"

' =====
' Beispiel aus FCL-Buch.sln
' Anlegen eines Benutzers mit dem WinNT-Provider
' (C) Holger@Schwichtenberg.de
' =====
Sub WINNT_Benutzeranlegen()

    Dim o As DirectoryEntry
    Dim c As DirectoryEntries

    Dim Benutzer_DN As String

    Benutzer_DN = "WinNT://" & COMPUTER & "/" & BENUTZER

    outtitle("Benutzer anlegen:" & Benutzer_DN)

    ' -- Prüfung, ob Objekt vorhanden ist
    If DirectoryEntry.Exists(Benutzer_DN) Then

```

```
        out("Benutzer bereits vorhanden...")
        ADSI_Loeschen_Rekursiv(Benutzer_DN)
    End If

    ' --- Zugriff auf IADs
    o = New DirectoryEntry("WinNT://" & COMPUTER)
    ' --- Zugriff auf IADsContainer
    c = o.Children()
    ' --- Neues Objekt erzeugen
    o = c.Add(BENUTZER, "user")
    ' --- Verzeichnisattribute festlegen
    o.Invoke("setpassword", "123456789")
    o.CommitChanges()

    o.NativeObject.AccountDisabled = False
    o.CommitChanges()

    out("Benutzer angelegt!")

End Sub
```

Listing 15.27: Anlegen eines Benutzers mit dem WinNT-Provider

15.5.4 Dienste verwalten

Ein NT-Dienst wird im WinNT-Verzeichnis durch Verzeichnisobjekte der Klasse »Service« repräsentiert (eine Ausnahme ist der Dienst »lanmanserver«, siehe nächstes Unterkapitel). Ein Dienst ist ein Unterobjekt eines »Computer«-Containers.

COM-Schnittstellen

Im klassischen ADSI gibt es zwei Schnittstellen zur Verwaltung der Verzeichnisklasse »Service«: IADsService und IADsServiceOperations. Letztere Schnittstelle bietet die Möglichkeit, den Dienst zu starten, zu stoppen, anzuhalten und fortzusetzen.

Attribut	Erläuterung
StartType	Legt den Zeitpunkt fest, zu dem der Dienst startet: automatisch (2), manuell (3), deaktiviert (4)
ServiceType	Repräsentiert die Art des Prozesses
DisplayName	Angezeigter Name im Dienstmanager
Path	Pfad zu der zugehörigen EXE-Datei
ErrorControl	Legt die Maßnahmen im Fehlerfall fest
HostComputer	ADSI-Pfad des Computers, auf dem der Dienst läuft
LoadOrderGroup	Legt die Load Order Group fest

Tabelle 15.9: Mitglieder der COM-Schnittstelle IADsService

Attribut	Erläuterung
ServiceAccount Name	Name des NT-Benutzers, unter dem sich dieser Dienst am System anmeldet. Der Wert »Local-System« repräsentiert den Systemaccount.
Dependencies	Namen der Dienste, von denen dieser Dienst abhängig ist
Description	Nur bei Fileservices: textliche Beschreibung
MaxUserCount	Nur bei Fileservices: maximale Anzahl der Benutzer; -1 = unbestimmt

Tabelle 15.9: Mitglieder der COM-Schnittstelle IADsService (Fortsetzung)

Mitglied	Erläuterung
Status	Aktueller Betriebszustand des Dienstes: nicht gestartet (1), gestartet (4), angehalten (7). Dieses Attribut kann nur gelesen werden. Um den Status zu verändern, stehen die Methoden Start(), Stop(), Pause() und Continue() zur Verfügung.
Start()	Starten des Dienstes
Stop()	Stoppen des Dienstes
Pause()	Anhalten des Dienstes
Continue()	Fortsetzen des Dienstes
SetPassword()	Setzen des Passworts des Dienstkontos

Tabelle 15.10: Mitglieder der COM-Schnittstelle IADsServiceOperations

Beispiel

Die folgende Unterroutine kehrt den Startzustand des Webserver-Dienstes (IIS) um: Ist der Dienst gestartet, wird er gestoppt. Ist der Dienst gestoppt, wird er gestartet.

**Dienst starten/
stoppen**

```
' =====
' Beispiel aus FCL-Buch.sln
' Status eines Dienstes umkehren
' (C) Holger@Schwichtenberg.de
' =====
Sub WINNT_Dienste_veraendern()

    Dim Dienst As DirectoryEntry
    Dim iads_sop As ActiveDs.IADsServiceOperations

    ' --- Bindung an Dienstobjekt
    Dienst = New DirectoryEntry("WinNT://mars/w3svc")
    ' --- Bindung an COM-Schnittstelle
    iads_sop = Dienst.NativeObject
```

```

' --- Informationen über den Dienst
out("Dienstname: " & getAtt(Dienst, "Name"))
out("Starttyp: " & getAtt(Dienst, "Starttype"))
out("Status: " & iads_sop.Status)

' --- Dienstzustand umkehren
If iads_sop.Status = 4 Then ' Dienst läuft
' --- Dienst beenden
iads_sop.Stop()
out("Dienst wurde beendet!")
Else ' Dienst läuft nicht
' --- Dienst starten
iads_sop.Start()
out("Dienst gestartet!")
End If
End Sub

```

Listing 15.28: Status eines Dienstes umkehren

15.5.5 Dateisystem-Dienst verwalten

Dateisystem- Dienst

Ein »Fileservice«-Objekt ist ein spezielles »Service«-Objekt, das einen Dienst zum Zugriff auf ein Dateisystem bereitstellt. Ein »Fileservice«-Objekt ist von dem allgemeinen »Service«-Objekt abgeleitet und erweitert dieses um einige Funktionalitäten, insbesondere um den Zugriff auf die verbundenen Benutzer (Sessions), die geöffneten Dateien (Resources) und die Freigabe (Fileshares).

Der Fileservice in Windows heißt »lanmanserver«. Der Zugriff auf den NT-Fileservice erfolgt folgendermaßen: *WinNT://computer-name/lanmanserver*. Ein »Fileservice«-Objekt ist im Gegensatz zu einem »Service«-Objekt ein Container: Ein »Fileservice« kann »FileShare«-Objekte enthalten. Außerdem enthält ein »Fileservice«-Objekt zwei Collections:

- Sessions verweist auf die offenen Benutzersitzungen.
- Resources enthält die geöffneten Ressourcen.

Das klassische ADSI bietet dazu eine passende COM-Schnittstelle *IADsFileService* mit zwei untergeordneten Collections (siehe Tabelle).

Collection-Name	Bedeutung	Schnittstelle der Unterobjekte
Sessions	Benutzersitzungen (mit einem Rechner verbundene Benutzer)	IADsSession
Resources	Auf einem Rechner geöffnete Dateien/Verzeichnisse	IADsResource

Tabelle 15.11: Unterobjekte der COM-Schnittstelle IADsFileService

Die außerdem vorhandene Schnittstelle IADsFileServiceOperations entspricht der Funktionalität der Schnittstelle IADsServiceOperations.

Freigaben verwalten

Ein »Fileshare«-Objekt repräsentiert eine Verzeichnisfreigabe. Der Zugriff auf diese Objekte erfolgt ausschließlich über den NT-Dienst »ntlanmanserver«. Eine Freigabe ist direkt über einen ADSI-Pfad der Form *WinNT://ComputerName/lanmanserver/Freigabename* erreichbar.

Verzeichnisklasse
»Fileshare«

Attribut	Erläuterung
CurrentUserCount	Anzahl der aktuell mit dieser Verzeichnisfreigabe verbundenen Benutzer
Description	Textliche Beschreibung der Freigabe
HostComputer	ADSI-Pfad des Computers, auf dem die Freigabe eingerichtet ist
Name	Name der Verzeichnisfreigabe
Path	Lokaler Pfad, auf den die Freigabe abgebildet wird
MaxUserCount	Maximale Anzahl der Benutzer (-1 = unbestimmt)

Tabelle 15.12: Attribute der Verzeichnisdienst-Klasse »FileShare«

Zum Auflisten der Freigaben ist die COM-Interoperabilität nicht notwendig, da einfach nur der Inhalt des Containers »lanmanserver« ausgegeben werden muss.

Freigaben
auflisten

```
' =====
' Beispiel aus FCL-Buch.sln
' Auflisten der Freigaben
' (C) Holger@Schwichtenberg.de
' =====

Sub WINNT_FREIGABEN()
    Dim ofs As New _
        DirectoryEntry("WinNT://mars/lanmanserver")
    Dim c As DirectoryEntries
```

```

Dim oshare As DirectoryEntry

c = ofs.Children

For Each oshare In c
    say(getAtt(oshare, "Name") & " zeigt auf " & _
        getAtt(oshare, "Path"))
Next

End Sub

```

Listing 15.29: Liste der vorhandenen Freigaben auf einem Computer

**Freigaben nur
ohne Rechte
anlegbar**

Mit ADSI können zwar die Freigaben aufgelistet sowie Freigaben erzeugt und gelöscht werden, aber die Rechte können nicht gesetzt werden. Das Erzeugen von Verzeichnisfreigaben und die Vergabe von Rechten ist möglich mit der Windows Management Instrumentation (WMI), also dem Namespace `System.Management` in der FCL.

Benutzersitzungen und geöffnete Ressourcen

**Sitzungen und
Ressourcen
auflisten**

Die in den Collections enthaltenen `Session-` bzw. `Resource-`Objekte besitzen keinen ADSI-Pfad, da sie keine persistenten Verzeichnisobjekte sind. Sie sind daher bei der Instanziierung eines `DirectoryEntry-`Objekts nicht direkt einzeln ansprechbar, sondern können nur – wie im Folgenden gezeigt – über den Durchlauf durch die Collections angesprochen werden.

```

' --- Auflisten der Sitzungen und Ressourcen
Sub WINNT_SitzungenRessourcen()
    Dim fs As DirectoryEntry
    Dim iads_fs As ActiveDs.IADsFileService
    Dim oSession As ActiveDs.IADsSession
    Dim oResource As ActiveDs.IADsResource

    ' --- Bindung an Dienstobjekt
    fs = New _
        DirectoryEntry("WinNT://sonne2000/lanmanserver")
    ' -- Bindung an Fileservice
    iads_fs = fs.NativeObject
    ' -- Sitzungen
    For Each oSession In iads_fs.Sessions
        say(oSession.User & " ist verbunden von " & _
            oSession.Computer & " seit " & _
            oSession.ConnectTime & " Sekunden")
    Next
    ' -- Ressourcen
    For Each oResource In iads_fs.Resources

```

```
Try
    out(oResource.User & " verwendet " & _
        oResource.Path)
Catch
    out("/// nicht auswertbares Resource-Objekt!")
End Try
Next
End Sub
```

Listing 15.30: Ausgabe der Benutzersitzungen und geöffneten Dateien

Bei der Resource-Collection kommt es beim Zugriff via COM-Interoperabilität zu einem Problem, weil die Collection ein Objekt mehr enthält, als es tatsächlich geöffnete Pfade gibt. Dieses letzte Objekt ist nicht auswertbar; daher die Fehlerbehandlung mit Try...End Try in obiger Routine.



Bug

16 Namespace System.Threading

Viele Anwendungen, die heute auf Windows-Systemen ablaufen, sind so genannte *Multithread-Anwendungen*. Das bedeutet, dass die Anwendung an sich nicht nur einen ablaufenden Programmpfad hat, sondern mehrere parallel ablaufende, beispielsweise um Operationen im Hintergrund ausführen zu können und dem Benutzer trotzdem zu ermöglichen, neue Aktionen anzustoßen. Das .NET-Framework bietet eine recht umfangreiche Unterstützung für solche Programme. Die Hauptklasse ist die Klasse `System.Threading.Thread`.

Multithreading

16.1 Funktionsweise von Threads

Multithread-Anwendungen werden erst ermöglicht durch die Multitasking-Fähigkeit des Betriebssystems. Multitasking unter Windows ist ein so genanntes *präemptives Multitasking*, d.h. Programme scheinen zwar gleichzeitig abzulaufen, sie tun es aber in Wirklichkeit nicht. In der Realität wird jedem aktiven Prozess – dazu gehören auch Windows-Dienste, Programme, Threads – eine gewisse Zeit zur Abarbeitung zugestanden. Ist diese so genannte Zeitscheibe abgelaufen, wird zum nächsten Prozess gesprungen und dessen Zeitscheibe wird gestartet. Diese Zeiten sind extrem kurz, wodurch der Benutzer die Illusion gleichzeitig ablaufender Funktionen erhält.

**Preemptives
Multitasking**

Innerhalb Ihres Programms können Sie nun auch festlegen, dass es mehrere Ablaufstränge geben soll, die dann ebenfalls ein Scheibchen Zeit zum Ablauf erhalten. Anders als bei Systemanwendungen können Sie aber selbst festlegen, welche Priorität diese Threads erhalten bzw. wie viel Zeit oder wie oft ihnen Zeit zugestanden wird. Sie können also auch in Ihren eigenen Programmen quasi-gleichzeitig ablaufende Stränge festlegen, z.B. für Aktionen, die im Hintergrund laufen sollen. Ein gutes Beispiel hierfür wäre

**Ablaufstränge
(Threads)**

die Überwachung des Dateisystems, ähnlich wie beim Windows Explorer. Wenn irgendein Programm eine neue Datei in ein Verzeichnis speichert, das vom Windows Explorer angezeigt wird, zeigt auch dieser die Datei an, d.h. die Anzeige wird sofort aufgefrischt. (Falls Sie mit Windows NT4 arbeiten, mag das nicht der Fall sein – der NT4-Explorer beinhaltet diese Funktionalität nicht, NT4 selbst unterstützt aber selbstverständlich wie alle heutigen Windows-Versionen preemptives Multitasking.)

16.2 Threads starten und stoppen

16.2.1 Thread und ThreadStart

ThreadStart Die Klasse `Thread` und der Delegate `ThreadStart` arbeiten zusammen. Während `Thread` den Thread selbst repräsentiert und z.B. die Einstellung der Priorität, das Beenden oder das Starten des Threads implementiert, dient `ThreadStart` lediglich der Angabe des Einsprungpunkts für den Thread. Die Klasse `Thread` besitzt eine Methode `Start()`, der ein Objekt vom Typ `ThreadStart` übergeben werden muss.

`ThreadStart` ist wie gesagt ein Delegate, der folgendermaßen implementiert ist:

```
Public Delegate Sub ThreadStart()
```

**Einsprungpunkt
für den Thread**

Demnach muss auch die Methode, die den Einsprungpunkt für den Thread darstellt, eine Methode mit dieser Signatur sein, nämlich ohne Parameter und ohne Ergebniswert. Falls Sie dennoch Werte an diese Methode übergeben müssen, können Sie durchaus noch Felder der Klasse verwenden, in der die Methode deklariert ist. Dass es keinen Ergebniswert gibt, ist hingegen logisch, denn wir definieren hier einen selbstständigen Ablaufstrang, d.h. es gibt keine Methode, an die etwas zurückgeliefert werden könnte.

Wenn Sie wissen, welche Methode den Thread starten soll, also sozusagen die Hauptmethode des Threads ist (ebenso wie die Methode `Main()` die Hauptmethode eines C#-Programms ist), können Sie einen Thread folgendermaßen starten:

```
Dim tStart As New ThreadStart(AddressOf ThreadMethod)  
myThread = New Thread(tStart)  
myThread.Start()
```

In diesem Beispiel wäre `ThreadMain()` die Hauptmethode des Threads.

Starten eines Threads

Ein einfaches Beispiel für einen Thread und gleichzeitig den Beweis, dass sowohl der Thread als auch das Hauptprogramm vollkommen unabhängig voneinander laufen, liefert der folgende Programmcode. Der Thread bzw. die Hauptmethode des Threads ändert einfach die Farbe einer `Panel`-Komponente. Zum Beweis, dass das Hauptprogramm weiterhin läuft, wurde die Berechnung der Fibonacci-Zahlen implementiert, die unabhängig von der sich ändernden Panel-Farbe gestartet werden kann.

Das Hauptformular besteht aus einer `Panel`-Komponente für den Thread sowie einer Textbox, einem Button und einer Listbox für die Berechnung der Fibonacci-Zahlen. Es werden so viele Fibonacci-Zahlen berechnet, wie in der Textbox angegeben werden. Die Ausgabe erfolgt in einer Listbox. Den Quelltext sehen Sie in Listing 16.1

```
'=====
' FiboThread.sln
' Copyright 2002 Frank Eller
' Einfaches Thread-Beispiel
'=====

Imports System.Threading

Public Class Form1
    Inherits System.Windows.Forms.Form

    #Region " Windows Form Designer generated code "

    'Der Code, der von VS.NET eingefügt wurde, wird
    'aus Übersichtsgründen nicht dargestellt.

    #End Region

    Dim myThread As Thread

    Private Sub ThreadMethod()
        'Hauptmethode des Threads
        Dim i As Integer = 0
        Dim u As Integer = 0

        While True
```

```

        i = i + 1
        If i > 2540000 Then
            i = 1
        End If
        u = i / 10000
        pnlThread.BackColor = Color.FromArgb(u, u, u)
    End While
End Sub

```

```

Private Sub btnFibo_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnFibo.Click

    ' ---Berechnung der Fibonacci-Zahlen
    ' --- Dies geschieht im Haupt-Thread

    ' --- Deklarationen und Prüfung
    Dim fiboCount As Integer
    If tbxFibo.Text = String.Empty Then
        fiboCount = 10
    Else
        fiboCount = Convert.ToInt32(tbxFibo.Text)
    End If

    Dim fibo(fiboCount) As Integer
    Dim i As Integer

    'Initialisierung
    fibo(0) = 0
    fibo(1) = 1

    ' --- Berechnen
    lblFibonacci.Items.Clear()

    For i = 2 To fiboCount
        fibo(i) = fibo(i - 1) + fibo(i - 2)
    Next

    For Each i In fibo
        lblFibonacci.Items.Add(i)
    Next

End Sub

```

```

Private Sub tbxFibo_KeyPress( _
    ByVal sender As Object, _
    ByVal e As KeyPressEventArgs) _
    Handles tbxFibo.KeyPress

```

```

' --- Nur Eingabe von Zahlen erlauben
If Not Char.IsDigit(e.KeyChar) Then
    e.Handled = True
End If

End Sub

Private Sub Form1_Load( _
    ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles MyBase.Load

    ' --- In dieser Methode wird der Thread gestartet
    Dim tStart As New ThreadStart( _
        AddressOf ThreadMethod)
    myThread = New Thread(tStart)
    myThread.Start()

End Sub

Private Sub btnExit_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnExit.Click

    ' --- Programm beenden und Thread stoppen
    myThread.Abort()
    While Not ( _
        myThread.ThreadState = ThreadState.Stopped)
        ' --- Warten ...
    End While
    Me.Close()

End Sub
End Class

```

Listing 16.1: FiboThread.exe

Die Beendigung des Threads, die hier mit `Abort()` angestoßen wird, ist notwendig. Threads, die im .NET-Framework ausgeführt werden, werden wie Windows-Prozesse gehandhabt. Das bedeutet: Wenn Sie das Programm beenden, ohne vorher die Threads zu beenden, kann es passieren, dass der Thread selbst noch als eigener Prozess weiterläuft, ohne dass Sie es merken. Wenn Sie ein solches Programm testweise in Visual Studio laufen lassen, werden Sie feststellen, dass trotz der Beendigung noch der Debugger läuft.

Abbildung 16.1 zeigt das Programm zur Laufzeit. Das Panel, dessen Farbe über den Thread geändert wird, befindet sich rechts oben.

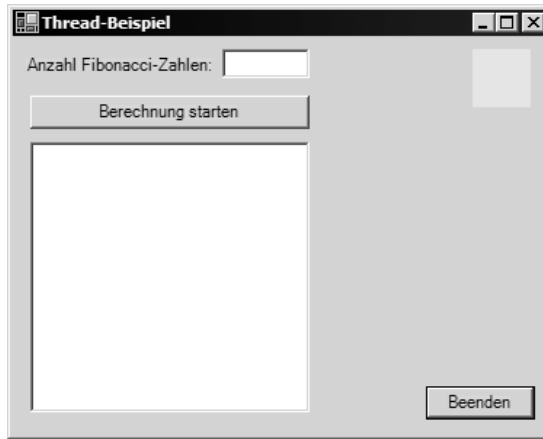


Abbildung 16.1: Ein Thread-Beispiel (SimpleThread.exe)

16.2.2 Anhalten und Fortsetzen von Threads

Suspend() und Resume()

Suspend() und Resume()

Jeder Thread kann während des Programmlaufs angehalten und weitergeführt werden. Zum Anhalten eines Threads dient die Methode `Suspend()`, die nur dann etwas bewirkt, wenn der Thread tatsächlich gerade läuft und noch nicht angehalten wurde. Das Gegenteil, also das Fortsetzen des Threads, bewirkt die Methode `Resume()`. Beide sind Instanzmethoden der Klasse `Thread`.

Abort()

Wollen Sie einen Thread komplett beenden, verwenden Sie die Methode `Abort()`. Diese Methode beendet den Thread, geht dabei allerdings einen sichereren Weg, als den Thread (bei dem es sich ja um einen Prozess handelt) einfach abzuschießen. Stattdessen wird das Beenden des Threads eingeleitet und eine Exception des Typs `ThreadAbortException` ausgelöst.

ThreadAbort-Exception

Diese Exception wird nicht gemeldet (es erscheint also keine Meldung über eine Exception zur Laufzeit) und kann auch nicht mittels `Try...Catch` abgefangen werden. Die Laufzeitschicht verhindert dies. Allerdings kann es durchaus geboten sein, einen `Finally`-Block zu deklarieren, der auf jeden Fall ausgeführt wird. Falls Sie also innerhalb des Threads Objekte benutzt haben, die noch entsorgt werden müssen (wenn Sie z.B. mit Dateien gearbeitet haben), können Sie das Aufräumen durch einen `Finally`-Block sicherstellen.

Wenn Sie `Abort()` aufgerufen haben, sollten Sie immer darauf achten, auch wirklich darauf zu warten, dass der Thread beendet wird. Sicher ist sicher. Das tun Sie einfach, indem Sie warten, bis der Status des Threads, der über die Eigenschaft `ThreadState` ermittelt werden kann, den Wert `ThreadState.Stopped` hat.

Wenn Sie die Methode `Abort()` verwenden, müssen Sie sicherstellen, dass der entsprechende Thread nicht angehalten ist. Ansonsten wird eine `ThreadStateException` ausgelöst. Im folgenden Beispiel ist dies allerdings nicht berücksichtigt, weshalb Sie vor dem Beenden alle Threads am Laufen haben müssen.



ThreadState

Mehrere Threads in Aktion

In diesem Beispiel lassen wir mehrere Threads gemeinsam laufen. Das Hauptformular wurde mit vier Panels bestückt, die für die einzelnen Threads stehen. Bei allen Panels wird die Farbe im Hintergrund mittels je eines Threads langsam geändert. Über vier Checkboxes, die ebenfalls auf dem Formular platziert sind, können Sie die einzelnen Threads zwischendurch anhalten. Der Button `btnStartThreads` startet die Threads, beendet werden sie beim Programmende.

Das Formular zur Entwurfszeit zeigt Abbildung 16.2.

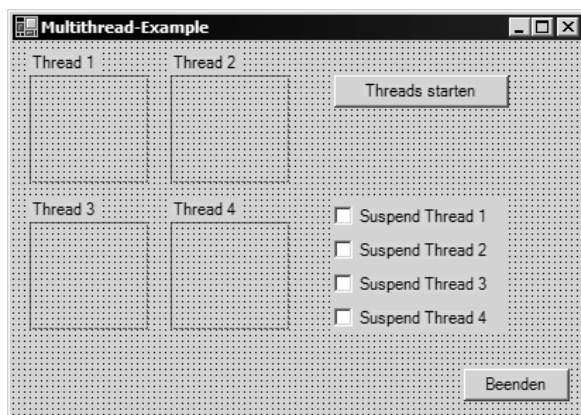


Abbildung 16.2: Das Beispielformular zur Entwurfszeit [`MultiThreadExample.sln`]

Die Funktion der Threads ist die gleiche wie beim vorherigen Beispiel, allerdings laufen nun vier Threads anstelle von einem. Über Checkboxes können die jeweiligen Threads angehalten werden.

Listing 16.2 zeigt Ihnen die nicht von Visual Studio .NET eingefügten Methoden des Programms.

```
'=====
' MultipleThreads.sln
' Copyright 2002 Frank Eller
' Mehrere Threads
'=====

Imports System.Threading

Public Class Form1
    Inherits System.Windows.Forms.Form

    #Region " Windows Form Designer generated code "

        ' Der Code, der von VS.NET eingefügt wurde,
        ' wird nicht dargestellt.

    #End Region

    Private t1 As Thread
    Private t2 As Thread
    Private t3 As Thread
    Private t4 As Thread

    Private Sub doThread1()

        ' --- Hauptmethode Thread 1
        Dim i As Integer = 0
        Dim u As Integer = 0

        While True
            i += 1
            If i > 2540000 Then
                i = 1
            End If
            u = i / 10000
            pnlThread1.BackColor = Color.FromArgb(u, u, u)
        End While

    End Sub

    Private Sub DoThread2()

        ' --- Hauptmethode Thread 2
        Dim i As Integer = 0
        Dim u As Integer = 0

        While True
            i += 1
            If i > 2540000 Then
```



```
        i = 1
    End If
    u = i / 10000
    pnlThread2.BackColor = Color.FromArgb(0, 0, u)
End While

End Sub

Private Sub DoThread3()

    ' --- Hauptmethode Thread 3
    Dim i As Integer = 0
    Dim u As Integer = 0

    While True
        i += 1
        If i > 2540000 Then
            i = 1
        End If
        u = i / 10000
        pnlThread3.BackColor = Color.FromArgb(0, u, u)
    End While

End Sub

Private Sub DoThread4()

    ' --- Hauptmethode Thread 4
    Dim i As Integer = 0
    Dim u As Integer = 0

    While True
        i += 1
        If i > 2540000 Then
            i = 1
        End If
        u = i / 10000
        pnlThread4.BackColor = Color.FromArgb(u, u, 0)
    End While

End Sub

Private Sub btnStartThreads_Click( _
    ByVal sender As Object, _
    ByVal e As System.EventArgs) _
    Handles btnStartThreads.Click

    ' --- ThreadStart-Objekte festlegen
    Dim tStart1 As _
        New ThreadStart(AddressOf doThread1)
    Dim tStart2 As _
        New ThreadStart(AddressOf DoThread2)
```

```
Dim tStart3 As _
    New ThreadStart(AddressOf DoThread3)
Dim tStart4 As _
    New ThreadStart(AddressOf DoThread4)

t1 = New Thread(tStart1)
t2 = New Thread(tStart2)
t3 = New Thread(tStart3)
t4 = New Thread(tStart4)

t1.Start()
t2.Start()
t3.Start()
t4.Start()

End Sub

Private Sub cbxSuspend1_Click( _
    ByVal sender As Object, _
    ByVal e As System.EventArgs) _
    Handles cbxSuspend1.Click

    If cbxSuspend1.Checked Then
        t1.Suspend()
    Else
        t1.Resume()
    End If

End Sub

Private Sub cbxSuspend2_Click( _
    ByVal sender As Object, _
    ByVal e As System.EventArgs) _
    Handles cbxSuspend2.Click

    If cbxSuspend2.Checked Then
        t2.Suspend()
    Else
        t2.Resume()
    End If

End Sub

Private Sub cbxSuspend3_Click( _
    ByVal sender As Object, _
    ByVal e As System.EventArgs) _
    Handles cbxSuspend3.Click

    If cbxSuspend3.Checked Then
        t3.Suspend()
```

```
Else
    t3.Resume()
End If

End Sub

Private Sub cbxSuspend4_Click( _
    ByVal sender As Object, _
    ByVal e As System.EventArgs) _
    Handles cbxSuspend4.Click

    If cbxSuspend4.Checked Then
        t4.Suspend()
    Else
        t4.Resume()
    End If

End Sub

Private Sub Form1_Closing(ByVal sender As Object, _
    ByVal e As System.ComponentModel.CancelEventArgs) _
    Handles MyBase.Closing

    t1.Abort()
    t2.Abort()
    t3.Abort()
    t4.Abort()

    Dim allAborted As Boolean

    Do
        allAborted = _
            (t1.ThreadState = ThreadState.Stopped) And _
            (t2.ThreadState = ThreadState.Stopped) And _
            (t3.ThreadState = ThreadState.Stopped) And _
            (t4.ThreadState = ThreadState.Stopped)
    Loop While Not allAborted

End Sub

Private Sub btnExit_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnExit.Click

    Me.Close()

End Sub
End Class
```

Listing 16.2: MultipleThreads.exe

Wenn Sie das Beispielprogramm ausführen, werden Sie sehen, dass die Farben der einzelnen Panels sich langsam aber stetig ändern. Da jeder Thread Rechenzeit beansprucht, gehen die Farbänderungen umso flüssiger vor sich, je mehr Threads abgeschaltet wurden. Außerdem werden Sie feststellen, dass, wenn alle Threads eingeschaltet sind und Sie in den `Thread`-Methoden die Zeile

```
i += 1
```

durch

```
i += 5
```

ersetzen, also die Farbänderung schneller vor sich gehen lassen, diese etwas abgehackt aussieht. Das hat mit der Zeitscheibe für die Threads zu tun, die in diesem Beispiel für alle Threads gleich ist.

Sleep() und Interrupt()

Sleep() Eine weitere Möglichkeit, einen Thread anzuhalten, ist die Methode `Sleep()`, eine statische Methode der Klasse `Thread`. Sie ermöglicht es, einen Thread für eine bestimmte Anzahl von Millisekunden anzuhalten oder sogar so lange, bis der Thread von einem anderen Thread aus wieder eine Freigabe bekommt, weiterzulaufen.

TimeSpan Die Methode `Sleep()` arbeitet mit zwei Arten von Parametern, nämlich entweder mit einem `Integer`-Wert, der die Zeitspanne in Millisekunden angibt, die der Thread blockiert bleiben soll, oder mit einem `TimeSpan`-Wert. `TimeSpan` ist ein `Struct`, mit dessen Hilfe Sie auch mit Sekunden- oder Minuten-Werten statt nur mit Millisekunden arbeiten können. Arbeiten Sie mit der ersten Methode, können Sie den Thread auch dazu bewegen, den Rest seiner Zeitscheibe einfach abzugeben, d.h. wenn alle notwendigen Funktionen beendet sind, das System zu veranlassen, zum nächsten Thread zu springen. Dazu übergeben Sie der Methode `Sleep()` den Wert 0.



Der längste Wert, den Sie hier über `TimeSpan` angeben können, ist ein Tag. Das hat aber in der Regel nicht viel zu sagen – zwar könnten Sie einen Thread bei entsprechenden Anwendungen auch längere Zeit anhalten müssen, aber das ließe sich auch noch mit anderen Mitteln lösen.

Die dritte Möglichkeit, nämlich einen Thread so lange »schlafen« zu lassen, bis er wieder »aufgeweckt« wird, besteht in der Übergabe des Parameters `Timeout.Infinite`. Dabei handelt es sich um einen konstanten Wert, der zur Folge hat, dass der Thread so lange einschläft, bis seine Instanzmethode `Interrupt()` aufgerufen wird.

Timeout.Infinite

`Sleep()` ist eine statische Methode, `Interrupt()` eine Instanzmethode. Der Hintergrund ist der, dass immer nur der aktuell laufende Thread angehalten werden darf – daher darf `Sleep()` keine Instanzmethode sein, weil man sie ansonsten für einen bestimmten Thread zu jedem beliebigen Zeitpunkt aufrufen könnte. Da sie aber statisch ist, wird sie innerhalb einer Thread-Methode aufgerufen und stoppt dann den gerade laufenden Thread.

`Interrupt()` selbst muss allerdings in diesem Fall zwangsläufig eine Instanzmethode sein, denn sie kommt nur dann zum Einsatz, wenn der Thread wirklich auf ein Kommando zum Weitermachen wartet. Da der Thread in diesem Moment angehalten ist, brauchen wir eine Instanzmethode, um ihn explizit ansprechen zu können.

Interrupt()

Zeitscheibe eines Threads freigeben

Im folgenden Beispiel verwenden wir nahezu den gleichen Code wie im vorherigen Beispiel. Allerdings geben wir den Rest der Zeitscheibe eines Threads jetzt frei, sobald die Arbeit getan ist. Dadurch wird erreicht, dass die Farbänderung im Gegensatz zum vorherigen Beispiel auch bei Verwendung aller Threads flüssiger abläuft, d.h. keine abrupten Farbsprünge mehr macht.

Allerdings muss dazu eine weitere Änderung vorgenommen werden. Da der Thread nach dem Aufruf von `Sleep()` neu gestartet wird, würden auch die Variablen wieder initialisiert. Das müssen wir verhindern, daher machen wir aus der lokalen Variablen `i` ein Feld und deklarieren sie für das Formular `global`. Ansonsten hat sich am Code nichts geändert, weswegen auch manche sich wiederholende Teile weggelassen wurden.

```
'=====
' MultipleThreads2.sln
' Copyright 2002 Frank Eller
' Mehrere Threads Beispiel 2
'=====
```

```
Imports System.Threading
```

```
Public Class Form1
```

```

Inherits System.Windows.Forms.Form

#Region " Windows Form Designer generated code "

    'Der Code, der von VS.NET eingefügt wurde,
    ' wird nicht dargestellt.

#End Region

Private t1 As Thread
Private t2 As Thread
Private t3 As Thread
Private t4 As Thread

Private Sub doThread1()

    ' --- Hauptmethode Thread 1
    Dim i As Integer = 0
    Dim u As Integer = 0

    While True
        i = i + 5
        If i > 2540000 Then
            i = 1
        End If
        u = i / 10000
        pnlThread1.BackColor = Color.FromArgb(u, u, u)
    End While

End Sub

' Die anderen Thread-Methoden entsprechend,
' daher hier nicht mehr abgedruckt

Private Sub btnStartThreads_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnStartThreads.Click

    ' --- ThreadStart-Objekte festlegen
    Dim tStart1 As _
        New ThreadStart(AddressOf doThread1)
    Dim tStart2 As _
        New ThreadStart(AddressOf DoThread2)
    Dim tStart3 As _
        New ThreadStart(AddressOf DoThread3)
    Dim tStart4 As _
        New ThreadStart(AddressOf DoThread4)

    t1 = New Thread(tStart1)
    t2 = New Thread(tStart2)

```

```
t3 = New Thread(tStart3)
t4 = New Thread(tStart4)

t1.Start()
t2.Start()
t3.Start()
t4.Start()

End Sub

Private Sub cbxSuspend1_Click( _
    ByVal sender As Object, _
    ByVal e As System.EventArgs)
    Handles cbxSuspend1.Click

    If cbxSuspend1.Checked Then
        t1.Suspend()
    Else
        t1.Resume()
    End If

End Sub

' Die anderen CheckBox-Methoden entsprechend,
' daher hier nicht mehr abgedruckt

Private Sub Form1_Closing(ByVal sender As Object, _
    ByVal e As System.ComponentModel.CancelEventArgs) _
    Handles MyBase.Closing

    t1.Abort()
    t2.Abort()
    t3.Abort()
    t4.Abort()

    Dim allAborted As Boolean

    Do
        allAborted = _
            (t1.ThreadState = ThreadState.Stopped) And _
            (t2.ThreadState = ThreadState.Stopped) And _
            (t3.ThreadState = ThreadState.Stopped) And _
            (t4.ThreadState = ThreadState.Stopped)
    Loop While Not allAborted

End Sub

Private Sub btnExit_Click( _
    ByVal sender As System.Object, _
```

```
ByVal e As System.EventArgs) _  
Handles btnExit.Click  
  
Me.Close()  
  
End Sub  
End Class
```

Listing 16.3: MultipleThreads2.exe

Die Geschwindigkeit ist nicht gerade atemberaubend. Wenn Sie einige der Threads abschalten, können Sie sehen, dass die Farbänderung schneller vor sich geht. Sie sehen auch, dass offensichtlich jeder Thread die gleiche Menge an Zeit zur Verfügung hat. Das hat mit Prioritäten zu tun und Sie können dieses Verhalten auch ändern. Abbildung 16.3 zeigt zunächst noch ein Bild zur Laufzeit des Programms.

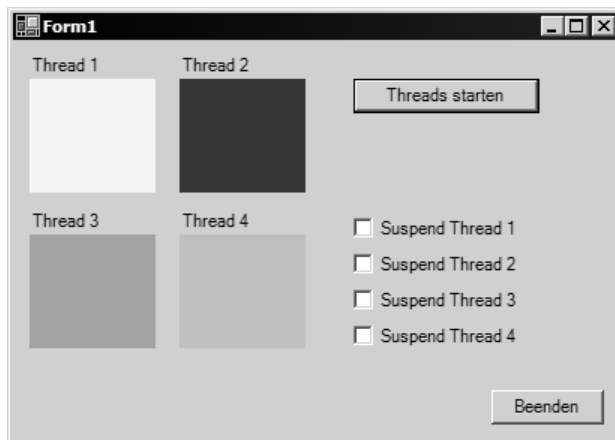


Abbildung 16.3: Das Thread-Beispiel zur Laufzeit

16.3 Thread-Prioritäten setzen

In allen vorherigen Beispielen haben die Threads der Anwendung die gleiche Priorität. Sie haben die Möglichkeit, diese Priorität zu ändern und somit einem bestimmten Thread mehr Zeit zur Ausführung zukommen zu lassen – sprichwörtlich. Denn die Zeitscheibe, die ein Thread zugewiesen bekommt, ist abhängig von der Priorität, die er hat.

Jeder Thread besitzt eine bestimmte Priorität, die Sie über die Eigenschaft `Priority` festlegen können. Es gibt fünf unterschiedliche Prioritätsstufen, die in der Aufzählung `ThreadPriority` festgelegt sind.

- ▶ `ThreadPriority.Highest` gibt diesem Thread die höchstmögliche Priorität. Diese Einstellung sollten Sie nur dann verwenden, wenn es unbedingt nötig ist, da ein solcher Thread andere Threads mit einer niedrigeren Priorität komplett blockieren kann.
- ▶ `ThreadPriority.AboveNormal` gibt dem Thread ein wenig mehr Priorität als einem »normalen« Thread.
- ▶ `ThreadPriority.Normal` ist die Standardeinstellung für einen neuen Thread.
- ▶ `ThreadPriority.BelowNormal` gibt dem Thread weniger Priorität, als es normalerweise der Fall ist.
- ▶ `ThreadPriority.Lowest` ist die niedrigste Einstellung. Wie auch für `ThreadPriority.Highest` gilt, dass Sie diese Einstellung möglichst nicht verwenden sollten, da ein solcher Thread sehr leicht von anderen aufgrund seiner niedrigen Prioritätsstufe blockiert wird.

Eigentlich gibt es wenige Gründe, die Standardeinstellungen zu verändern. Falls Sie es dennoch tun wollen, sollten Sie den höchsten und den niedrigsten Wert tunlichst vermeiden. Geben Sie einem Thread einen zu hohen Prioritätswert, kann es sein, dass sich die Hauptfunktionen Ihres Programms schwerfällig verhalten. Reagiert das Hauptprogramm gar nicht mehr, bleibt nur noch der Weg über den Taskmanager. Es empfiehlt sich also, nur die mittleren drei Einstellungen zu verwenden. Noch besser ist es, alle Threads auf der Einstellung `ThreadPriority.Normal` zu belassen und nur die Threads, die wirklich nicht bedeutend sind, eine Stufe niedriger zu setzen.

Die Priorität bestimmt zwar grundlegend, wie viel Laufzeit ein Thread zugewiesen bekommt; sie ist aber dennoch von vielen Faktoren abhängig. Die Einstellung der Eigenschaft `Priority` ist nur einer dieser Faktoren. Der reale Anteil an Laufzeit, der einem Thread zugewiesen wird, berechnet sich in .NET unter anderem aus der Priorität des Threads, der Priorität des Prozesses, in dem der Thread läuft, und aus einem dynamischen Wert. Der aus diesen Werten errechnete Wert ist die tatsächliche Priorität des Threads.



16.4 Synchronisation

16.4.1 Wozu synchronisieren?

Wenn Sie mit mehreren Threads arbeiten, müssen Sie möglicherweise auch auf die Synchronisation dieser Threads achten. Und zwar dann, wenn Sie mit mehreren Threads auf das gleiche Objekt zugreifen wollen. Der Grund ist ganz einfach.

Nehmen wir an, Sie haben ein Objekt, in dem sich ein Array befindet. Nehmen wir weiter an, dieses Objekt stellt Methoden zur Verfügung, mit deren Hilfe die Werte dieses Arrays ausgelesen oder verändert werden können, und Methoden zum Sortieren dieses Arrays.

Wenn Sie nun nur mit einem Thread arbeiten, ist das Ganze kein Problem. Sie können die Methoden aufrufen und die gewünschte Funktion wird ausgeführt. Genauer gesagt, es kann keine unerwünschte Funktion ausgeführt, z.B. der Inhalt des zu sortierenden Arrays geändert werden, während dieses noch sortiert wird. Innerhalb ein und desselben Threads werden Methoden ja nacheinander aufgerufen.

Verwenden Sie aber zwei Threads, die beide auf dasselbe Objekt zugreifen, kann Folgendes passieren: Thread 1 greift auf das Objekt zu und versucht, das enthaltene Array zu sortieren, d.h. die Sortier-Routine wird aufgerufen. Da aber die Zeitscheibe eines Threads sehr klein ist, wird dieser beendet, bevor die Routine ausgeführt wurde (lange vorher sogar, wir reden hier von Millisekunden an Zeitscheiben). Nun bekommt der nächste Thread Zeit zugewiesen.

Wenn dieser zweite Thread nun auch auf das Objekt zugreift, aber die Methode zum Ändern der Array-Werte benutzt, bekommen Sie ein Problem. In dem Fall kann es nämlich sein, dass das Array verändert wird, während der erste Thread praktisch noch beim Sortieren ist – d.h. die Werte werden verändert und die Sortierung liefert ein eigentlich falsches Ergebnis.

Viel schlimmer kann es bei komplexeren Situationen kommen. Wenn zwei Threads unsynchronisiert auf dasselbe Objekt zugreifen, ist es durchaus möglich, dass bestimmte Werte einmal ungültig sind, weil der eine Thread dem anderen dazwischengepfuscht hat. Dann haben Sie ein Problem.

Vermieden werden solche Situationen durch einen so genannten **Mutex** (*Mutual exclusion lock*). Es wird schlicht verhindert, dass ein Thread auf ein Objekt zugreift, das von einem anderen Thread gerade bearbeitet wird. Das Prinzip ist recht einfach: Ein Thread sperrt ein Objekt für den Zugriff, bevor er mit dem Aufruf einer Methode des Objekts beginnt. Er legt also einen Mutex »um« das Objekt. Wenn seine Zeitscheibe beendet ist und ein anderer Thread auf dasselbe Objekt zugreifen will, wird dieser so lange in den Wartemodus geschickt, bis der Mutex wieder freigegeben und der Zugriff möglich ist. Dann ist die Arbeit des ersten Threads auch üblicherweise beendet – die Threads sind synchronisiert.

Mutex

16.4.2 Die Klasse Monitor

Im Namespace `System.Threading` gibt es eine Klasse `Monitor`, die diese Funktionalität beinhaltet. Diese Klasse besteht lediglich aus sechs statischen Methoden, die Sie in Tabelle 16.1 finden.

Methode	Funktion
<code>Enter()</code>	Sperrt das angegebene Objekt für den aktuellen Thread. Die Sperre wird durch den Aufruf von <code>Exit()</code> wieder aufgehoben.
<code>Exit()</code>	Hebt eine Sperre wieder auf. <code>Exit()</code> wird üblicherweise am Ende einer Methode, die gemeinsam verwendet wird, aufgerufen.
<code>Pulse()</code>	<code>Pulse()</code> informiert den nächsten anstehenden Thread darüber, dass ein Objekt blockiert ist. Falls dieser auch auf das Objekt zugreifen will, wird er in eine Warteschlange gestellt. Der nächste Thread, der in der Warteschlange steht, bekommt den Zugriff, sobald die Sperre aufgelöst wurde.
<code>PulseAll()</code>	Wie <code>Pulse()</code> , nur dass jetzt alle Threads informiert und ggf. in die Warteschlange gestellt werden. Auch hier übernimmt der Thread, der in der Warteschlange an vorderster Position steht.
<code>TryEnter()</code>	Wie <code>Enter()</code> , allerdings wird hier lediglich versucht, ein Objekt zu sperren, ggf. für eine bestimmte Anzahl an Millisekunden. Dadurch kann sichergestellt werden, dass nur ein Objekt gesperrt wird, das auch freigegeben ist.
<code>Wait()</code>	Hebt die Blockierung auf und blockiert stattdessen den aktuellen Thread, bis dieser erneut versucht, das Objekt zu sperren.

Tabelle 16.1: Die Methoden der Klasse `Monitor`

Allen Methoden wird zumindest die Instanz des Objekts übergeben, das gesperrt werden soll. Am häufigsten werden vermutlich die Methoden `Enter()`, `TryEnter()` und `Exit()` benutzt.

Verwenden der Monitor-Klasse

Die Funktion der `Monitor`-Klasse soll anhand eines kleinen Beispiels mit mehreren Arrays gezeigt werden. Verwendet wird eine Klasse, die zehn Arrays mit je zehn Elementen implementiert, die ausgelesen oder gesetzt werden können. Der erste Thread soll nun die Arrays sortieren, der zweite soll immer das aktuelle Array ausgeben. Das Ganze soll unabhängig voneinander funktionieren. Obwohl es sich hier um ein Beispiel handelt, könnte es ebenso gut sein, dass der zweite Thread das Array manipulieren will.

Das Formular besteht aus zwei Listboxen, genannt `lbxResult` und `lbxError`, die zur Anzeige etwaiger Fehler bzw. des Ergebnisses dienen. Aufgeteilt wurden die Meldungen deshalb, weil dann klar ersichtlich ist, dass auch wirklich beide Threads ausgeführt werden und nacheinander Zugriff auf das gemeinsame Objekt erhalten. Dazu kommen ein Button zum Beenden des Programms und zwei Buttons zum Starten bzw. Stoppen der Threads.

Sortiert werden Arrays, die in einer Klasse zusammengefasst sind. Aus dieser Klasse wird ein Objekt erzeugt, auf das beide Threads Zugriff erhalten. Dieses Beispiel hat zugegebenermaßen keinen besonderen Realitätsanspruch, zeigt aber gut, wie die `Monitor`-Klasse verwendet werden kann. In Listing 16.4 sehen Sie die Klasse mit den Arrays.

```
'=====
' MonitorExample.sln
' Copyright 2002 Frank Eller
' Klasse MultiArrays
'=====

Public Class MultiArrays

    Private intArrays(10, 10) As Integer
    Private _currArray As Integer = 0

    Public ReadOnly Property CurrArray() As Integer
        Get
            Return _currArray
        End Get
    End Property
```

```
Private Sub CreateArrays()  
    Dim i As Integer  
    Dim u As Integer  
    Dim rnd As New Random()  
  
    ' --- Erzeugen von 10 Arrays mit je 10 Elementen  
    For i = 0 To 9  
        For u = 0 To 9  
            intarrays(i, u) = rnd.Next(1, 100)  
        Next  
    Next  
End Sub  
  
Public Function GetCurrentArray() As Integer()  
    Dim currArray(10) As Integer  
    Dim i As Integer  
  
    For i = 0 To 9  
        currArray(i) = intarrays(Me._currArray, i)  
    Next  
  
    Return currArray  
End Function  
  
Public Sub SetCurrentArray(ByVal values As Integer())  
    Dim i As Integer  
  
    For i = 0 To 9  
        intarrays(_currArray, i) = values(i)  
    Next  
End Sub  
  
Public Sub MoveToNext()  
    _currArray = _currArray + 1  
    If _currArray > 9 Then  
        _currArray = 1  
    End If  
End Sub  
  
Public Overloads Function ToString( _  
    ByVal arrNumber As Integer) As String  
  
    Dim i As Integer  
    Dim result As String  
  
    For i = 0 To 8  
        result = result + _  
            intArrays(arrNumber, i).ToString() + ", "  
    Next  
    result = result + _  
        intArrays(arrNumber, 9).ToString()
```

```

        Return result
    End Function

    Public Sub New()
        CreateArrays()
    End Sub

End Class

```

Listing 16.4: Die Klasse MultiArrays [MonitorExample.sln]

Im eigentlichen Code erzeugen wir zwei Threads. Der erste holt sich das aktuelle Array, sortiert es und liefert es an die Klasse zurück, der andere greift unabhängig davon ebenfalls auf die Klasse zu und liefert eine Ausgabe des aktuellen Arrays, wenn möglich.

Thread.Sleep() Um wirklich zu simulieren, dass der erste Thread längere Zeit auf das Objekt zugreift, wird `Thread.Sleep(0)` benutzt, bevor `Monitor.Exit()` aufgerufen wird. Dadurch wird anderen Threads die Möglichkeit gegeben, ihren Code auszuführen. Da der zweite Thread auf das gleiche Objekt wie der erste zugreift, muss zwangsläufig eine Konstellation entstehen, bei der Thread 2 keinen Zugriff erhält, da das Objekt noch von Thread 1 gesperrt ist. Listing 16.5 zeigt den gesamten Quellcode.

```

'=====
' MonitorExample.sln
' Copyright 2002 Frank Eller
' Beispielprogramm für die Monitor-Klasse
'=====

Imports System.Threading

Public Class Form1
    Inherits System.Windows.Forms.Form

    ' Von Visual Studio generierter Code wurde
    ' der Übersichtlichkeit halber weggelassen

    Private mArrays As New MultiArrays()

    Private t1 As Thread
    Private t2 As Thread

    Private Sub Swap(ByRef x As Integer, _
        ByRef y As Integer)

        Dim z As Integer

```

```

z = x
x = y
y = z
End Sub

```

```

Private Sub SortArray(ByRef arr() As Integer)

```

```

    Dim i As Integer
    Dim hasChanged As Boolean = False

```

```

    For i = 1 To 9
        If arr(i - 1) < arr(i) Then
            Swap(arr(i - 1), arr(i))
        End If
    Next
End Sub

```

```

Private Sub ThreadMethod1()

```

```

    ' --- Methode für den ersten Thread
    ' --- Sortiert nacheinander alle Arrays
    Dim currArray(10) As Integer
    Dim arrNumber As Integer
    Dim couldEnter As Boolean = False
    Dim i As Integer

```

```

While True
    If Monitor.TryEnter(mArrays) Then
        couldEnter = True
        arrNumber = mArrays.CurrArray
        currArray = mArrays.GetCurrentArray()
        SortArray(currArray)
        mArrays.SetCurrentArray(currArray)
        mArrays.MoveNext()
        Thread.Sleep(0)
        Monitor.Exit(mArrays)
    Else
        couldEnter = False
    End If

```

```

    If couldEnter Then
        lbxResult.Items.Add("Array " + _
            arrNumber.ToString() + _
            " wurde sortiert")
    Else

```

```

        lbxError.Items.Add( _
            "Thread 1: Kein Zugriff möglich")
    End If

```

```

    Thread.Sleep(TimeSpan.FromSeconds(0.5R))
End While

```

```

End Sub

Private Sub ThreadMethod2()

    ' --- Methode für den zweiten Thread
    ' --- Gibt nacheinander die Arrays aus

    Dim currArray(10) As Integer
    Dim arrNumber As Integer
    Dim couldEnter As Boolean = False

    While True
        If Monitor.TryEnter(mArrays) Then
            couldEnter = True
            arrNumber = mArrays.CurrArray
            currArray = mArrays.GetCurrentArray()
            Monitor.Exit(mArrays)
        Else
            couldEnter = False
        End If

        If couldEnter Then
            lblResult.Items.Add("Array " + _
                                arrNumber.ToString() + _
                                ": " + mArrays.ToString(arrNumber))
        Else
            lblError.Items.Add( _
                                "Thread 2: Kein Zugriff möglich")
        End If

        If couldEnter Then
            Thread.Sleep(TimeSpan.FromSeconds(0.5R))
        End If
    End While

End Sub

Private Sub btnStart_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnStart.Click

    Dim tStart1 As _
        New ThreadStart(AddressOf ThreadMethod1)
    Dim tStart2 As _
        New ThreadStart(AddressOf ThreadMethod2)
    t1 = New Thread(tStart1)
    t2 = New Thread(tStart2)
    t1.Start()
    t2.Start()

End Sub

```



```

Private Sub Form1_Closing(ByVal sender As Object, _
    ByVal e As System.ComponentModel.CancelEventArgs) _
    Handles MyBase.Closing

    If t1.IsAlive Then
        t1.Abort()
        t2.Abort()

        Do Until ( _
            t1.ThreadState = ThreadState.Stopped) And _
            (t2.ThreadState = ThreadState.Stopped)
            ' --- Warten
        Loop
    End If
End Sub

Private Sub btnExit_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnExit.Click

    Me.Close()
End Sub

Private Sub btnStop_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnStop.Click

    t1.Abort()
    t2.Abort()
End Sub
End Class

```

*Listing 16.5: Sperren eines Objekts für einen Thread bei gleichzeitigem Zugriff
[MonitorExample.sln]*

Wenn Sie das Beispiel starten, kann es gut sein, dass sich eine ganze Zeit lang nichts tut. Irgendwann aber wird sich die zweite Listbox füllen und Ihnen anzeigen, dass die Synchronisation tatsächlich funktioniert. Abbildung 16.4 zeigt das Programm zur Laufzeit.

16.4.3 Die Anweisung SyncLock

`SyncLock` tut prinzipiell das Gleiche wie die Anweisungen `Enter()` und `Exit()` der Klasse `Monitor`, d.h. es handelt sich im Prinzip um eine verkürzte Schreibweise.

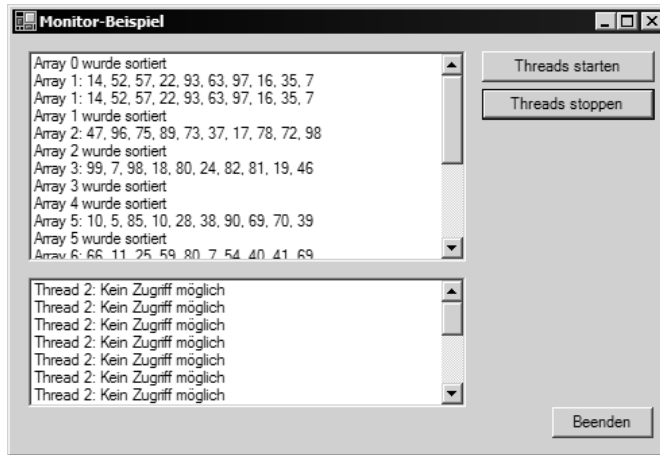


Abbildung 16.4: MonitorExample.exe zur Laufzeit

Bei dieser Anweisung besteht Verwechslungsgefahr, wenn man C# und Visual Basic .NET vergleicht, denn in C# heißt diese Anweisung `lock`. Eine `lock`-Anweisung gibt es auch in Visual Basic .NET, sie bezieht sich dort jedoch auf Dateien.

Die Syntax der `SyncLock`-Anweisung sieht folgendermaßen aus:

```
SyncLock Objekt
    Anweisungen
End SyncLock
```

Da sich die Funktionalität nicht von der im letzten Beispiel gezeigten unterscheidet, wird hier kein weiteres Beispiel zur Synchronisation vorgestellt. Stattdessen soll noch ein weiteres interessantes Thema bezüglich Threads angesprochen werden, nämlich Thread-Pools.

16.5 Threadpools

Threadpool Ein Threadpool ist eine effektive Möglichkeit, mehrere Threads zu verwalten. Angenommen, es sollte eine größere Anzahl gleichzeitig (oder nahezu gleichzeitig) ablaufender Handlungen gestartet werden, z.B. das Kopieren mehrerer Dateien; Sie wollen aber dem Benutzer ermöglichen, währenddessen weiterzuarbeiten.

Würden Sie nun für jeden dieser Ablaufstränge einen eigenen Thread programmieren, würde das das Programm ab einer

bestimmten Anzahl von Threads verlangsamen, denn durch das preemptive Multitasking wird die Rechenzeit unter allen Threads verteilt (zu denen auch der Haupt-Thread der Anwendung gehört).

Wenn Sie nun die Priorität des Haupt-Threads höher setzen als die der anderen Threads, werden diese langsamer, was ebenfalls nicht erwünscht ist. Sie müssen also dafür sorgen, dass nur eine bestimmte Anzahl Threads gleichzeitig läuft. Das können Sie durch einen Threadpool ermöglichen.

In diesem Pool befindet sich eine bestimmte Anzahl an Threads, die gleichzeitig laufen dürfen. Jede neue Operation, die im Hintergrund ablaufen soll, wird nun in eine Warteschlange gestellt. Ist ein Thread verfügbar, nimmt dieser sich der Operation an und führt sie aus. Dabei können Sie aber sicher sein, dass niemals mehr als die verfügbare Anzahl Threads gleichzeitig laufen.

Die Klasse `ThreadPool` stellt eine solche Funktionalität zur Verfügung. Standardmäßig enthält ein Threadpool maximal 25 Threads, die benutzt werden können. Sie haben die Möglichkeit, Threads zu benutzen, indem Sie Objekte bzw. eine Arbeitsfunktion in eine Warteschlange stellen. Wenn im Threadpool ein Thread verfügbar ist, ruft dieser die angegebene Methode auf und arbeitet sie ab.

ThreadPool

Die Methode `QueueUserWorkItem()` stellt Objekte in die Warteschlange. Die Übergabeparameter sind einmal die Adresse der Arbeitsmethode und weiterhin (optional) ein Objekt, mit dem gearbeitet werden kann. Die Arbeitsmethode muss dabei der Signatur des Delegates `WaitCallback` entsprechen, der für den Aufruf der Methode verwendet wird.

QueueUserWorkItem()

Der Delegate `WaitCallback` ist folgendermaßen deklariert:

WaitCallback

```
<Serializable>  
Public Delegate Sub WaitCallback( ByVal state As Object )
```

Wenn Sie über `QueueUserWorkItem()` sowohl die Methode als auch ein Arbeitsobjekt übergeben, wird dieses Arbeitsobjekt der Methode anstelle des Parameters `state` übergeben, so dass Sie dieses Objekt benutzen können. Sie müssen es aber nicht benutzen, sondern können auch für unterschiedliche Objekte Threads anfordern, die dann nacheinander abgearbeitet werden.

Threadpool mit mehreren Objekten

In diesem Beispiel wird ein Threadpool benutzt, wobei von mehreren Objekten, die bereits eine Funktionalität beinhalten, jeweils die Arbeitsmethode übergeben wird. Diese wird ausgeführt und der Thread danach beendet.

Das `state`-Objekt wird in diesem Beispiel nicht benötigt, da kein Objekt übergeben wird, mit dem gearbeitet werden soll. Die gesamte Funktionalität ist bereits in der aufgerufenen Methode enthalten.

Zunächst die Deklaration der Klasse, in der sich die Arbeitsmethode befindet und von der später mehrere Instanzen erzeugt werden. Es handelt sich um eine Klasse, die die Fibonacci-Zahlen rekursiv berechnet. Rekursiv deshalb, weil es dann zwischen den einzelnen Threads garantiert Geschwindigkeitsunterschiede gibt, d.h. dass die Threads mit Sicherheit nicht nacheinander abgearbeitet werden. So wird sichtbar, dass es sich hierbei wirklich um einen Multithread-Vorgang handelt.

```
'=====
' ThreadPool_1.sln
' Copyright 2002 Frank Eller
' Klasse FiboRecursive
'=====

Public Class FiboRecursive

    ' --- Berechnet Fibonacci-Zahlen rekursiv

    ' --- Felder
    Private fiboCount As Integer
    Private objectNo As Integer

    ' --- Konstruktor
    Public Sub New(ByVal count As Integer, _
        ByVal objNo As Integer)

        Me.fiboCount = count
        Me.objectNo = objNo
    End Sub

    ' --- Die eigentliche Fibonacci-Funktion
    Private Function FiboRek(ByVal count As Integer) _
        As Integer

        Dim result As Integer = 0
```

```

    If count < 2 Then
        result = 1
    Else
        result = FiboRek(count - 1) + _
            FiboRek(count - 1)
    End If

    Return result
End Function

' --- Diese Methode wird aus dem ThreadPool
' --- heraus aufgerufen. Es handelt sich um
' --- die Methode, die der Signatur des
' --- WaitCallback-Delegates entspricht

Public Sub ThreadedFiboRecursive( _
    ByVal state As Object)

    Dim fiboValues(Me.fiboCount) As Integer
    Dim i As Integer
    Dim resultStr As String = String.Empty

    For i = 0 To Me.fiboCount
        fiboValues(i) = Me.FiboRek(i)
    Next

    For Each i In fiboValues
        resultStr += i.ToString() + " "
    Next

    ' --- Kontrolle an nächsten Thread übergeben
    Thread.Sleep(0)

    ' --- Ergebnis In das Hauptformular schreiben
    projectMain.MainForm.lbxResult.Items.Add( _
        objectNo.ToString + ": " + resultStr)

End Sub

End Class

```

Listing 16.6: Die Klasse *FiboRecursive* [ThreadPool_1.sln]

Das Hauptformular besteht aus zwei Buttons – einem zum Starten der Bearbeitung unserer Objekte und einem zum Beenden des Programms – und einer Listbox, in der das Ergebnis dargestellt wird. Um aus der Methode `ThreadedFiboRecursive()` auf die Elemente des Hauptformulars zugreifen zu können, muss ein wenig getrickst werden. Der Einsprungpunkt des Programms – die Methode `Main()` – wird in eine eigene Klasse außerhalb des Haupt-

formulars verlegt. Dadurch kann eine Instanz des Hauptformulars vor der `Run()`-Methode von `Application` erzeugt werden und es wird möglich, auf die Elemente des Formulars zuzugreifen. Tun Sie dies nicht, haben Sie keinen Zugriff auf die `Listbox` aus der Methode `ThreadedFiboRecursive()` heraus.

Zusätzlich muss die Eigenschaft `Modifiers` der `Listbox` noch auf `public` gesetzt werden, damit sie auch verfügbar ist. Hier also der komplette Code des Hauptformulars.

```
'=====
' ThreadPool_1.sln
' Copyright 2002 Frank Eller
' Anwendung eines Threadpools
'=====

Public Class projectMain

    Public Shared MainForm As Form1 = New Form1()

    <STAThread()> _
    Public Shared Sub Main()
        Application.Run(MainForm)
    End Sub

End Class

Public Class Form1
    Inherits System.Windows.Forms.Form

    #Region " Windows Form Designer generated code "

        ' Der Code, der von VS.NET eingefügt wurde, wird
        ' aus Übersichtsgründen nicht dargestellt.

    #End Region

    ' --- Diese Methode fügt die Objekte dem Threadpool
    ' --- hinzu
    Private Sub AddToThreadPool()

        Dim i As Integer
        Dim ThreadItems(5) As FiboRecursive

        For i = 0 To 5
            ThreadItems(i) = _
                New FiboRecursive((25 - (i * 3)), i)
            ThreadPool.QueueUserWorkItem( _
                New WaitCallback( AddressOf _
                    ThreadItems(i).ThreadedFiboRecursive))
        Next i
    End Sub
End Class
```

```
Next

End Sub

' --- EventHandler für den Start-Button
Private Sub btnStart_Click(_
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnStart.Click

    AddToThreadPool()

End Sub

' --- Beenden des Programms
Private Sub btnExit_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnExit.Click

    Me.Close()

End Sub

End Class
```

Listing 16.7: Der Code des Hauptformulars von *ThreadPool_1.exe*

In der Methode `AddToThreadPool` werden sechs Objekte erzeugt, die nacheinander dem Threadpool zugewiesen werden, d.h. sie werden in die Warteschlange gestellt. Wenn ein Thread verfügbar ist, nimmt er sich der jeweiligen Methode an. Die Methode selbst (die Methode `ThreadedFiboRecursive()` der jeweiligen Klasse) wird ausgeführt und schreibt irgendwann das Resultat der Fibonacci-Berechnung in die Listbox.

Das Ergebnis des Programms sehen Sie in Abbildung 16.5.

Aus der Nummerierung am Anfang der Zeilen wird ersichtlich, dass die Threads wirklich in einer beliebigen Reihenfolge abgearbeitet wurden.

Threadpool mit Arbeitsobjekten

Eine weitere Möglichkeit ist, ein Objekt zu erzeugen, das eine grundlegende Methode beinhaltet, die die Arbeit erledigt, und dann weitere Arbeitsobjekte zu benutzen, die dieser Methode übergeben werden.

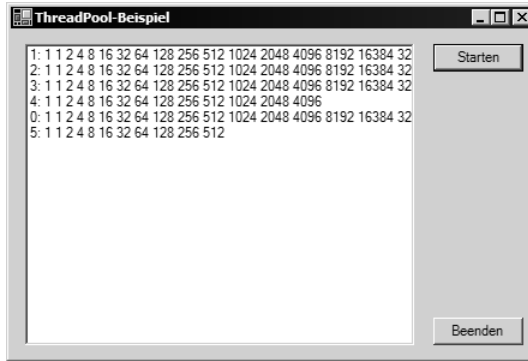


Abbildung 16.5: `ThreadPool_1` im Einsatz

Die angesprochene Signatur des `WaitCallback-Delegates` macht dies möglich, indem das gewünschte Arbeitsobjekt über den Parameter `state` übergeben wird. Das Hauptformular des Beispiels besteht wieder aus zwei Buttons und einer Listbox, die das Ergebnis anzeigt. Die Bezeichnungen sind die gleichen wie beim vorherigen Beispiel.

ManualResetEvent

Um zu erfahren, wann der Threadpool abgearbeitet ist, also wann alle unsere Elemente bearbeitet wurden, wird ein Feld `currCount` eingeführt, das die abgearbeiteten Elemente zählt. Wenn alle Elemente abgearbeitet sind, wird ein Event gesendet, das dem Programm mitteilt, dass die Threads fertig sind. Hierzu wird die Klasse `ManualResetEvent` aus `System.Threading` verwendet, mit der ein Thread angehalten werden kann, bis ein bestimmtes Signal übergeben wird.

Es werden wieder die Fibonacci-Zahlen berechnet. Die eigentliche Funktionalität wird in einer Klasse namens `WorkerClass` implementiert, die zu bearbeitende Klasse heißt `AFibo` und beinhaltet ein Array, in das die Fibonacci-Zahlen eingefügt werden sowie die Anzahl der Fibonacci-Zahlen, die bei Verwendung der jeweiligen Instanz der Klasse berechnet werden soll. Die Deklaration der Klasse `AFibo` sehen Sie in Listing 16.8.

```
'=====
' ThreadPool_2.sln
' Copyright 2002 Frank Eller
' Klasse AFibo
'=====
```

```
Public Class AFibo
```



```

Public FiboCount As Integer
Public FiboValues() As Integer
Public stateNumber As Integer

Public Sub New(ByVal count As Integer, _
    ByVal StateNo As Integer)

    Me.FiboCount = count
    ReDim Me.FiboValues(count)
    stateNumber = StateNo

End Sub

End Class

```

Listing 16.8: Die Klasse AFibo

Die Klasse `WorkerClass` besitzt eine Methode `CalcFibo()`, die die Arbeit erledigt und natürlich der Deklaration des Delegates `WaitCallback` entspricht. Diesmal wird allerdings im Gegensatz zum vorherigen Beispiel der Parameter `state` wirklich genutzt, er enthält das jeweilige `AFibo`-Objekt, mit dem gearbeitet werden soll.

Da in dieser Klasse gezählt werden muss, wie viele Elemente bereits abgearbeitet sind, um ein Feedback geben zu können, wann alle Klassen abgearbeitet wurden, wird eine Möglichkeit benötigt, von mehreren Threads aus auf den Zähler zugreifen zu können. Die Zählvariable heißt `currCount`. Der Zugriff über mehrere Threads hinweg wird durch die Klasse `Interlocked` ermöglicht, deren Methode `Increment()` verwendet wird, um den Zähler zu erhöhen.

Interlocked

Auch auf das Array, das die Resultate enthält, muss zugegriffen werden. Damit sich die Threads nicht in die Quere kommen, wird `SyncLock` verwendet. Die gesamte Deklaration der Klasse `WorkerClass` zeigt Listing 16.9.

```

'=====
' ThreadPool_2.sln
' Copyright 2002 Frank Eller
' Klasse WorkerClass
'=====

Public Class WorkerClass

    Public tpEvent As ManualResetEvent
    Public ResultArray() As String

    Private maxCount As Integer = 0

```

```

Private currCount As Integer = 0

Public Sub New(ByVal max As Integer)
    Me.maxCount = max
    ReDim ResultArray(max - 1)

End Sub

'Die eigentliche Fibonacci-Funktion
Private Function FiboRek(ByVal count As Integer) _
    As Integer

    Dim result As Integer = 0

    If count < 2 Then
        result = 1
    Else
        result = FiboRek(count - 1) + _
            FiboRek(count - 1)
    End If

    Return result
End Function

Public Sub CalcFibo(ByVal state As Object)

    Dim currFibo As AFibo = CType(state, AFibo)
    Dim i As Integer

    For i = 0 To currFibo.FiboCount
        currFibo.FiboValues(i) = FiboRek(i)
    Next

    SyncLock ResultArray
        ResultArray(currFibo.stateNumber) = _
            currFibo.stateNumber.ToString() + ": "
        For Each i In currFibo.FiboValues
            ResultArray(currFibo.stateNumber) += _
                i.ToString() + " "
        Next
    End SyncLock

    Interlocked.Increment(currCount)

    If currCount = maxCount Then
        tpEvent.Set() 'fertig
    End If
End Sub

End Class

```

Listing 16.9: Die Klasse WorkerClass

Das Objekt `tpEvent`, das in dieser Klasse nur deklariert, aber nicht instanziiert wird, enthält ein `ManualResetEvent`-Objekt, das im Hauptformular der Anwendung deklariert und instanziiert wird. Es wird dann einfach zugewiesen.

`tpEvent` wird im Hauptformular auf *unsigaled* gesetzt, der Haupt-Thread kann dann mit Hilfe der Methode `WaitOne()` des `ManualResetEvent`-Objekts auf die Beendigung der Threads warten. Es wird so lange gewartet, bis `tpEvent` auf *Signaled* gesetzt wird, was durch die Methode `Set()` geschieht. Diese rufen wir, wie aus Listing 16.9 ersichtlich, dann auf, wenn alle `AFibo`-Objekte abgearbeitet sind.

Den Quellcode des Hauptformulars der Anwendung zeigt Listing 16.10.

```
'=====
' ThreadPool_2.sln
' Copyright 2002 Frank Eller
' ThreadPool mit Arbeitsobjekten
'=====

Public Class ProjectMain

    Public Shared MainForm As New Form1()

    <STAThread(> _
    Public Shared Sub Main()
        Application.Run(MainForm)
    End Sub

End Class

Public Class Form1
    Inherits System.Windows.Forms.Form

    #Region " Windows Form Designer generated code "

        'Der Code, der von VS.NET eingefügt wurde, wird
        'aus Übersichtsgründen nicht dargestellt.

    #End Region

    Private Sub btnExit_Click( _
        ByVal sender As System.Object, _
        ByVal e As System.EventArgs) _
        Handles btnExit.Click

        Me.Close()
    End Sub
```

```

Private Sub btnStart_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnStart.Click

    ' --- Deklarationen
    Dim i As Integer

    lbxResult.Items.Add("Starte Threads ...")
    lbxResult.Items.Add(" ")

    ' --- Threadpool füllen
    ' --- Event erzeugen und auf unsignaled setzen
    Dim tpEvent As New ManualResetEvent(False)

    lbxResult.Items.Add("Erzeuge Objekte ...")
    Dim worker As New WorkerClass(5)
    worker.tpEvent = tpEvent

    ' --- Diese Methoden gehen nicht auf Win9x
    For i = 4 To 0 Step -1
        ThreadPool.QueueUserWorkItem( _
            New WaitCallback(AddressOf worker.CalcFibo), _
            New AFibo((i + 1) * 3, i))
    Next

    lbxResult.Items.Add( _
        "Warte auf Abfertigung des ThreadPools")
    lbxResult.Items.Add("")

    tpEvent.WaitOne(Timeout.Infinite, True)

    lbxResult.Items.Add( _
        "ThreadPool abgefertigt, Ergebnisse:")
    lbxResult.Items.Add("")

    lbxResult.Items.AddRange(worker.ResultArray)

End Sub
End Class

```

Listing 16.10: Threadpool mit Arbeitsobjekten [ThreadPool_2.sln]

Zunächst wird in diesem Beispiel eine neue Instanz der Klasse `WorkerClass` erzeugt, der die Anzahl der zu bearbeitenden Objekte übergeben wird. Die Methode `QueueUserWorkItem()` wird danach mehrmals aufgerufen, wobei immer wieder neue Objekte mit anderen Werten übergeben werden. In diesem Beispiel ist es so, dass das Objekt, das am längsten zur Bearbeitung braucht, als erstes übergeben wird.

Über `tpEvent.WaitOne()` wird gewartet, bis das Signal kommt, dass alle Objekte abgearbeitet wurden. Danach erfolgt die Ausgabe. Abbildung 16.6 zeigt das Programm, aus dem auch ersichtlich wird, dass der Thread, der am wenigsten Zeit benötigt hat, als erster abgearbeitet wurde – obwohl er als letzter angestoßen wurde.

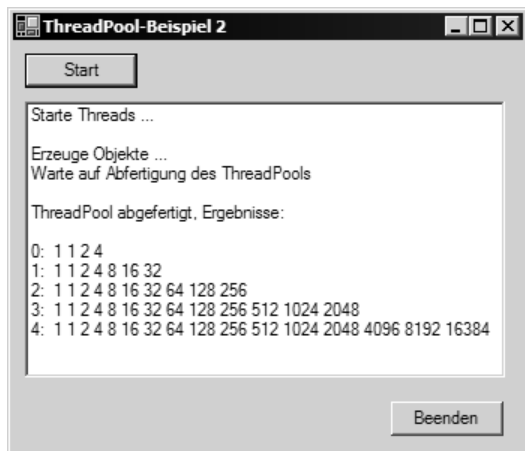


Abbildung 16.6: *ThreadPool_2.exe* im Einsatz

17 Namespace System.Windows.Forms

Der Namespace `System.Windows.Forms` enthält Klassen und Komponenten, die speziell dafür entworfen wurden, grafische Benutzerschnittstellen für Windows-Applikationen zu erstellen. Dazu gehören sowohl visuelle Steuerelemente, wie z.B. eine Textbox oder ein Label, als auch Dialogboxen wie ein Druckerdialog oder ein Dateidialog.

17.1 Grundlagen

17.1.1 Die Quelldatei des Programms

Eine grafische Benutzerschnittstelle erstellen Sie, indem Sie Steuerelemente auf einem Formular platzieren. Wenn Sie das tun, setzt der Designer Ihre Aktion in Quellcode um und fügt diesen automatisch ein. Auch alle Änderungen an den Eigenschaften eines Steuerelements werden in Quellcode umgesetzt. Die Methode, in der diese Basiseinstellungen zugewiesen werden, heißt `InitializeComponent()`. Das könnte z. B. so aussehen wie in Abbildung 17.1.

grafische Benutzerschnittstelle

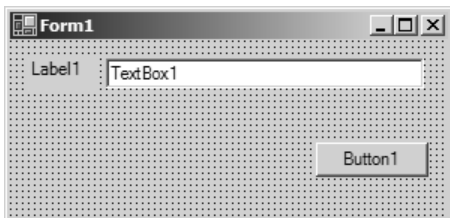


Abbildung 17.1: Drei Komponenten auf einem Formular mit angepasster Größe und Position

Dort wurden ein Label, eine Textbox und ein Button in das Hauptformular eingefügt und deren Position und Größe angepasst. In der Methode `InitializeComponent()` sind alle diese Änderungen, die Sie auf der visuellen Seite vorgenommen haben, in Quellcode umgesetzt. Diesen sehen Sie in Listing 17.1.

Das passende Listing dazu:

```
Friend WithEvents Label1 As _
    System.Windows.Forms.Label
Friend WithEvents Button1 As _
    System.Windows.Forms.Button
Friend WithEvents TextBox1 As _
    System.Windows.Forms.TextBox
<System.Diagnostics.DebuggerStepThrough(> _
    Private Sub InitializeComponent()

        Me.Label1 = New System.Windows.Forms.Label()
        Me.Button1 = New System.Windows.Forms.Button()
        Me.TextBox1 = New System.Windows.Forms.TextBox()
        Me.SuspendLayout()
        '
        'Label1
        '
        Me.Label1.Location = _
            New System.Drawing.Point(12, 12)
        Me.Label1.Name = "Label1"
        Me.Label1.Size = New System.Drawing.Size(48, 23)
        Me.Label1.TabIndex = 0
        Me.Label1.Text = "Label1"
        '
        'Button1
        '
        Me.Button1.Location = _
            New System.Drawing.Point(204, 68)
        Me.Button1.Name = "Button1"
        Me.Button1.TabIndex = 1
        Me.Button1.Text = "Button1"
        '
        'TextBox1
        '
        Me.TextBox1.Location = _
            New System.Drawing.Point(64, 12)
        Me.TextBox1.Name = "TextBox1"
        Me.TextBox1.Size = _
            New System.Drawing.Size(212, 20)
        Me.TextBox1.TabIndex = 2
        Me.TextBox1.Text = "TextBox1"
        '
        'Form1
```



```

    ,
    Me.AutoScaleBaseSize = _
        New System.Drawing.Size(5, 13)
    Me.ClientSize = New System.Drawing.Size(292, 117)
    Me.Controls.AddRange( _
        New System.Windows.Forms.Control() _
        {Me.TextBox1, Me.Button1, Me.Label1})
    Me.Name = "Form1"
    Me.Text = "Form1"
    Me.ResumeLayout(False)

```

End Sub

Listing 17.1: Die Methode InitializeComponent() nach dem Einfügen einiger Komponenten

Normalerweise müssen Sie an diesem Code nichts ändern. Visual Studio gibt Ihnen auch den Hinweis, hier nichts zu verändern – in der Tat ist es aber so, dass Sie auch hier Änderungen vornehmen können, wenn Sie genau wissen, was Sie tun. Wenn Sie die Änderungen korrekt vornehmen, werden diese auch im Design-Modus widergespiegelt. Quelltext und Design sind also eng miteinander verknüpft.

**Erzeugten Code
ändern**

Normalerweise verwenden Sie für jedes Formular bzw. für jede Klasse eine eigene Quelldatei. Manchmal werden auch mehrere Klassen in einer einzigen Quelldatei untergebracht. Bei Formularen ist das grundsätzlich zwar auch möglich, bringt aber eigentlich nichts außer Verwirrung. Zwar haben Sie dann den Quelltext zweier Formulare in einer Datei, der Designer zeigt aber nur das erste Formular an – das bedeutet auch, dass Sie nur an diesem Formular im Design-Modus arbeiten können. Außerdem weigerte sich das Visual Studio .NET nach diesem Versuch standhaft, ein Kontextmenü anzuzeigen.

**Mehrere Formulare
in einer Datei**

17.1.2 Start des Programms

Der Start eines Programms ist ebenfalls festgelegt. In C# sehen Sie im Quelltext eine Methode `Main()`, die wie folgt deklariert ist:

```

[STAThread]
Static void Main()
{
    Application.Run(New Form1());
}

```

Programmstart `Form1` bezeichnet dabei das Hauptformular der Anwendung. Über die Anweisung `Application.Run()` wird die Nachrichtenschleife (Message Queue) von Windows gestartet, d.h. ab diesem Moment werden die Ereignisse des übergebenen Formulars abgearbeitet. Wird dieses Formular beendet, ist das auch das Ende des Programms.

Main() In Visual Basic .NET werden Sie die Methode `Main()` vergeblich suchen, hier wird das Hauptformular lediglich in den Projektoptionen angegeben und das Programm startet dann mit einer neuen Instanz dieses Formulars. Das bedeutet allerdings nicht, dass es nicht auch mit einer Methode `Main()` funktionieren würde – dazu müssen sie lediglich eine solche Methode nach obigem Muster erstellen und in den Projektoptionen die `Main()`-Methode als Startmethode einstellen. Grundsätzlich tut Visual Basic im Hintergrund aber auch nichts anderes.

Die Verwendung der Methode `Main()` wird lediglich dann interessant, wenn Sie versuchen, von einem zweiten Formular auf die Komponenten des Hauptformulars zuzugreifen. Das funktioniert nämlich nicht. Mit Hilfe einer weiteren Klasse und einer `Main()`-Methode funktioniert es aber doch. Ein entsprechendes Beispiel finden Sie im nächsten Abschnitt.

17.1.3 Anordnen von Steuerelementen

Das Anordnen von Steuerelementen bei einer evtl. Größenänderung eines Formulars war immer eine lästige Sache (auch wenn manche Entwickler das nicht so tragisch sehen). Es mussten stets die jeweiligen Positionen im `Paint`-Ereignis (oder im `Resize`-Ereignis) angepasst werden, wenn das Formular vergrößert oder verkleinert wurde. Auch hier bietet das .NET Framework Hilfe an.

Anchor Alle visuellen Komponenten besitzen eine Eigenschaft `Anchor`, mit deren Hilfe sie an den Kanten eines Formulars verankert werden können. Wenn Sie z.B. einen Button verwenden möchten, der immer rechts unten zu sehen sein soll, egal wie groß das Formular ist, setzen Sie seine `Anchor`-Eigenschaft einfach auf `Bottom` und `Right`. Soll eine Komponente stets in der Mitte des Formulars verbleiben und die Größe proportional zum Formular ändern, setzen Sie `Anchor` einfach auf `Left`, `Top`, `Bottom` und `Right` – schon bleibt das Element in der Mitte und ändert seine Größe. Gerade bei Formularen, die recht viele Komponenten besitzen, ist diese Eigenschaft

ein großer Schritt nach vorne. Abbildung 17.2 zeigt das Anchor-Feld im Eigenschaftsfenster.

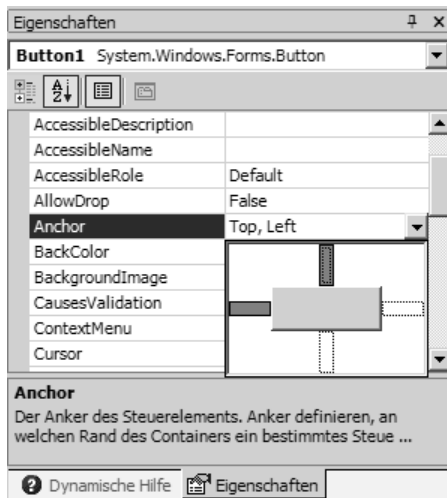


Abbildung 17.2: Das Anchor-Feld

17.2 Arten von Steuerelementen

Das .NET Framework umfasst einige Steuerelemente, die bereits einen großen Teil der in einem Windows-Programm benötigten Funktionalität abdecken. Natürlich haben Sie die Möglichkeit, eigene Steuerelemente hinzuzufügen. Es gibt drei Kategorien, nämlich Controls (visuelle Steuerelemente), nicht-visuelle Steuerelemente und Dialoge.

17.2.1 Controls

Als Controls oder visuelle Steuerelemente werden alle enthaltenen Komponenten bezeichnet, die etwas mit dem Design der Programmoberfläche zu tun haben. Sie sind grundsätzlich von der Klasse `Control` abgeleitet. Diese Klasse liefert die Basisfunktionalität für visuelle Komponenten.

17.2.2 Nicht-visuelle Steuerelemente

Ebenfalls als Steuerelemente, nicht aber als visuelle, werden die Steuerelemente bezeichnet, die zusätzliche Funktionalität bieten. Ein Beispiel dafür wäre z.B. die Komponente `ErrorProvider` oder auch die datensensitiven Komponenten, z.B. `DataSet`. Sie sind nicht von der Klasse `Control` abgeleitet (`DataSet` ist z.B. direkt von `MarshalByRefObject` abgeleitet, das wiederum ein direkter Nachfahre der Klasse `Object` ist). Diese Komponenten werden zur Entwurfszeit nicht direkt auf dem Formular angezeigt, sondern in einem gesonderten Bereich darunter.

17.2.3 Dialoge

Eine weitere Komponentenart sind die Dialoge. Standardmäßig bietet das .NET Framework Dialoge zum Öffnen und Speichern von Dateien, zum Einstellen des Zeichensatzes, zur Einstellung von Farben, zum Drucken, zur Seiteneinrichtung beim Drucker und für die Druckvorschau. Leider wird einer der Dialoge vermisst, nämlich der zur Auswahl eines Verzeichnisses.

17.2.4 Übersicht

Eine Übersicht über die Standard-Steuerelemente liefert Tabelle 17.1.

Komponente	Funktion
<code>TextBox</code>	Bietet die Möglichkeit der Texteingabe. Eine <code>TextBox</code> kann sowohl einzeilig als auch mehrzeilig sein. Im zweiten Fall wird der Text in einem Array aus Strings gespeichert, auf das über die Eigenschaft <code>Lines</code> zugegriffen werden kann.
<code>RichTextBox</code>	Auch eine <code>RichTextBox</code> ist zur Texteingabe vorgesehen. Im Unterschied zur <code>TextBox</code> kann eine <code>RichTextBox</code> das RTF-Format verwenden, d.h. der enthaltene Text kann formatiert werden.
<code>Label</code>	Ein <code>Label</code> dient zur Anzeige eines statischen Textes auf dem Formular.
<code>LinkLabel</code>	Ein <code>LinkLabel</code> ermöglicht es, einen Link ins Internet anzuzeigen und bei Klick darauf auch dorthin zu springen.
<code>StatusBar</code>	Die <code>StatusBar</code> entspricht einer Statuszeile, wie Sie sie aus allen Windows-Programmen kennen.

Tabelle 17.1: Übersicht über die Standardkomponenten in Visual Studio

Komponente	Funktion
CheckedListBox	Die <code>CheckedListBox</code> ist eine Liste von Eintragungen mit je einer <code>Checkbox</code> davor, wodurch die Elemente markiert werden können. Das entspricht nicht der Verhaltensweise einer normalen <code>Listbox</code> , da hier die Elemente unabhängig voneinander markiert werden können und es auch bleiben.
ComboBox	Eine <code>Combobox</code> ist ebenfalls aus allen Windows-Programmen bekannt.
DomainUpDown	Das Steuerelement <code>DomainUpDown</code> dient dazu, sich über zwei Buttons (Hoch/Herunter) durch eine Liste von Textelementen zu bewegen.
Listbox	Auch die <code>Listbox</code> ist aus vielen Windows-Programmen bekannt.
ListView	Eine <code>ListView</code> liefert die Funktionalität, die Sie aus der rechten Explorer-Seite kennen. Allerdings nur, was die Anzeige betrifft ... es ist hier kein Windows Explorer eingebaut.
NumericUpDown	<code>NumericUpDown</code> ermöglicht es, sich durch eine Liste numerischer Werte zu bewegen.
TreeView	Eine <code>Treeview</code> entspricht dem Aussehen nach der linken Seite des Windows Explorers.
PictureBox	Die <code>PictureBox</code> dient der Aufnahme und Bearbeitung einer Grafik.
ImageList	Das Steuerelement <code>ImageList</code> ermöglicht es, eine Liste von Grafiken aufzunehmen, auf die dann über den Index zugegriffen werden kann.
CheckBox	Auch die <code>Checkbox</code> ist aus diversen Windows-Programmen wohlbekannt.
RadioButton	Der <code>RadioButton</code> ist ebenfalls eine bekannte Komponente aus vielen Windows-Applikationen.
TrackBar	Die <code>TrackBar</code> ist eine Komponente, die es ermöglicht, ähnlich eines Lautstärkereglers einen Wert einzustellen.
DateTimePicker	Mit Hilfe von <code>DateTimePicker</code> können Sie aus einem grafischen Kalender ein Datum auswählen.
MonthCalendar	<code>MonthCalendar</code> ist ebenfalls ein Datums-Steuerelement, das es ermöglicht, auch einen ganzen Bereich auszuwählen.
ColorDialog	Liefert den Standard-Farbdialog von Windows
FontDialog	Liefert den Standard-Zeichensatzdialog von Windows
OpenFileDialog	Liefert einen Dialog zum Öffnen einer Datei
SaveFileDialog	Liefert einen Dialog zum Speichern einer Datei

Tabelle 17.1: Übersicht über die Standardkomponenten in Visual Studio (Fortsetzung)

Komponente	Funktion
PrintDialog	Liefert einen Dialog zum Drucken
PrintPreviewDialog	Liefert eine Druckvorschau
MainMenu	Dieses Steuerelement repräsentiert das Hauptmenü einer Anwendung.
ContextMenu	Dieses Steuerelement repräsentiert ein Kontextmenü.
Button	Repräsentiert eine herkömmliche Windows-Schaltfläche
NotifyIcon	NotifyIcon repräsentiert ein Icon für eine Applikation, das aber nicht in der Startleiste gezeigt wird, sondern im System-Tray (also in dem Bereich, in dem sich auch die Uhrzeit und das Datum befinden).
ToolBar	Repräsentiert eine herkömmliche Windows-Toolbar
Panel	Mit Hilfe eines Panels können Sie Fensterbereiche voneinander abgrenzen.
GroupBox	Auch die Groupbox dient zur Abgrenzung von Bereichen, bietet allerdings noch die Möglichkeit, eine Überschrift anzugeben.
TabControl	Ein TabControl ist ein mehrseitiges Steuerelement. Auf die einzelnen Seiten kann mittels Tabs zugegriffen werden.

Tabelle 17.1: Übersicht über die Standardkomponenten in Visual Studio (Fortsetzung)

Leider kann in diesem Buch aus Platzgründen nicht auf die einzelnen Komponenten eingegangen werden. Sie werden allerdings bereits nach kurzer Zeit feststellen, dass ihre Handhabung sehr einfach ist.

Hierarchie Zur Übersicht zeigt Abbildung 17.3 noch einen Ausschnitt aus der Vererbungshierarchie der Steuerelemente des .NET Frameworks. Würden an dieser Stelle alle Steuerelemente dargestellt, wäre das Diagramm nicht mehr lesbar, daher also nur ein Auszug, der Ihnen aber die Zusammenhänge aufzeigen sollte.

17.3 Datenbindung

Datenbindung in Windows Forms ermöglicht es dem Programmierer, jede beliebige Art von Daten an ein Steuerelement zu binden. Dazu gehören nicht nur die Daten, die aus einer Datenbank stammen, sondern beispielsweise auch die Inhalte einer `ArrayList`, die Inhalte eines Arrays oder aber die Inhalte anderer Steuerelemente.

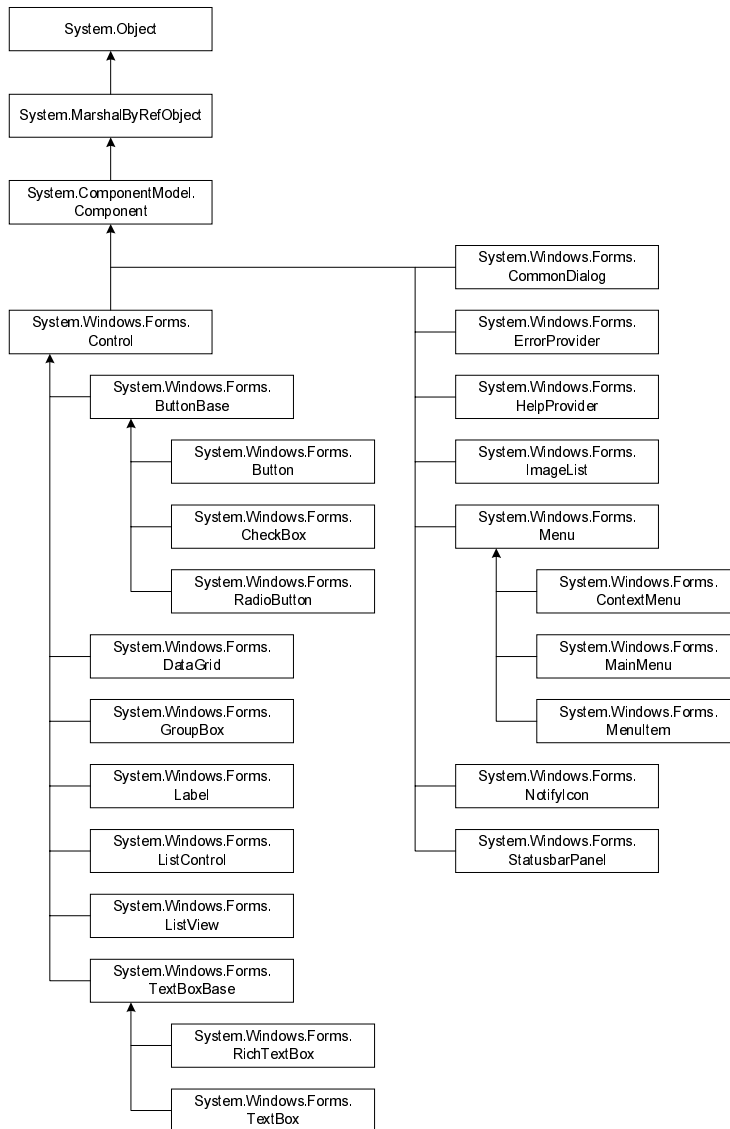


Abbildung 17.3: Vererbungsbaum von Windows Forms (Ausschnitt)

Es gibt zwei Arten der Datenbindung, die einfache und die komplexe Datenbindung. Bei der einfachen Datenbindung wird ein Steuerelement an ein einziges Datenelement gebunden. Ein Beispiel hierfür wäre die Anzeige eines Datensatzes in einem Dateneingabeformular. Jede Textbox zeigt dabei genau ein Element des Datensatzes an, ist also an dieses eine Element gebunden.

einfache Datenbindung

komplexe Datenbindung

Die komplexe Datenbindung ermöglicht es, ein Steuerelement an mehrere Datenobjekte zu binden. Ein gutes Beispiel hierfür ist das Steuerelement `DataGrid`, das den gesamten Inhalt einer Datentabelle anzeigen kann.

Binding

Kern der Datenbindung ist die Klasse `Binding`, die eine einfache Bindung zwischen einem Datenelement und einem Steuerelement herstellt. Jedes Steuerelement, das von `Control` abgeleitet ist, stellt eine Eigenschaft `DataBindings` vom Typ `ControlBindingCollection` zur Verfügung, die `Binding`-Objekte enthält. `Binding` selbst ermöglicht den Zugriff auf das gebundene Objekt über die Eigenschaft `DataSource`. Das angebundene Objekt kann jedes beliebige Objekt sein, das die Schnittstelle `IList` implementiert. Abbildung 17.4 zeigt das Objektmodell der `Binding`-Klasse.

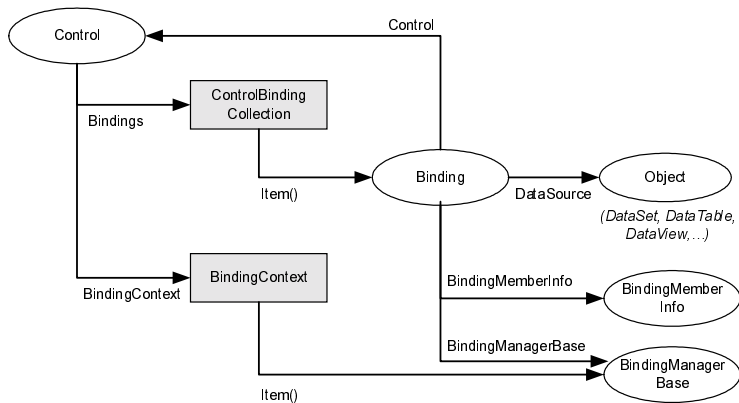


Abbildung 17.4: Das Objektmodell der `Binding`-Klasse

Datenbindung mit Steuerelementen

Binding-Objekt erzeugen

Ein einfaches Beispiel für Datenbindung ist das Binden zweier Steuerelemente aneinander. Das ist mit wenigen Zeilen erledigt. Zunächst wird eine Instanz der Klasse `Binding` benötigt, in der festgelegt wird, welche Eigenschaft der Datenquelle an welche Eigenschaft des anzeigenden Elements gebunden wird und – natürlich – um welche Datenquelle es sich handelt.

DataBindings

Danach muss das erzeugte `Binding`-Objekt nur noch der `DataBindings`-Aufzählung des Steuerelements hinzugefügt werden, das zur Anzeige dient. Das Ganze wird im `Load`-Ereignis des Formulars durchgeführt.

Im Beispiel selbst wird der Wert eines `DateTimePicker`-Steuerelements in einem Label dargestellt. Immer wenn sich dieser Wert ändert, ändert sich auch die Anzeige im Label. Den Quelltext des Load-Ereignisses sehen Sie in Listing 17.2.

```
Private Sub Form1_Load( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles MyBase.Load

    Dim aBinding As Binding
    aBinding = New Binding("Text", dtpDate, "Value")

    ' --- Jetzt anbinden
    lblDatum.DataBindings.Add(aBinding)

End Sub
```

Listing 17.2: Das Load-Ereignis mit der Anbindung des Steuerelements
[DataBindExample.sln]

Einen Screenshot des Programms zeigt Abbildung 17.5.



Abbildung 17.5: Datenbindung mit .NET [DataBindExample.sln]

17.4 Die Klasse Form

Formulare sind die Basis von Windows-Programmen. Alle Formulare sind von der Klasse `Form` abgeleitet, die über verschiedene weitere Klassen auch von `System.Windows.Forms.Control` abstammt. Die Hierarchie sehen Sie in Abbildung 17.6.

Formulare

Die Klasse `Form` repräsentiert ein Fenster in einer Anwendung. Alle Fenster, dazu gehören auch Dialoge, Toolboxen, MDI-Child-Fenster, verankerte und unverankerte Fenster, werden durch die Klasse `Form` repräsentiert.

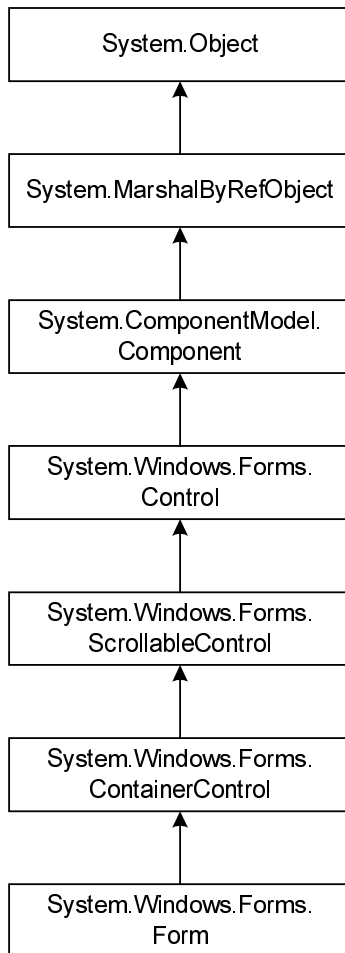


Abbildung 17.6: Hierarchie von *System.Windows.Forms.Form*

17.4.1 Eigenschaften

Die Eigenschaften der Klasse `Form` ermöglichen es Ihnen, das Erscheinungsbild Ihrer Formulare anzupassen. Sie können z.B. festlegen, welche der `System-Buttons` in der Kopfzeile erscheinen sollen, wo das Formular auf dem Desktop angezeigt wird, welche Farbe oder Größe es hat und noch vieles mehr. Tabelle 17.2 zeigt einige der interessantesten Eigenschaften der Klasse `Form`.

Eigenschaft	Bedeutung
AcceptButton	Legt im Falle eines Dialogfensters fest, welcher Button der OK-Button ist. Beim Betätigen dieses Buttons wird <code>DialogResult</code> automatisch auf den entsprechenden Wert des Buttons eingestellt.
ActiveControl	Liefert das Steuerelement, das gerade den Fokus besitzt
ActiveForm	Statisch (<code>shared</code>); liefert das aktuelle aktive Fenster der Anwendung
ActiveMdiChild	Liefert das aktive MDI-Child-Fenster zurück
AllowDrop	Gibt an, ob das Formular Ziel einer Drag&Drop-Operation sein kann
AutoScale	Gibt an, ob die Steuerelemente des Formulars skaliert werden sollen
AutoScaleBaseSize	Ruft die Grundgröße für die automatische Skalierung eines Formulars ab
AutoScroll	Gibt an, ob der automatische Bildlauf ermöglicht wird oder nicht
AutoScrollMargin	Legt die Größe des Rands für den automatischen Bildlauf fest
AutoScrollMinSize	Gibt die minimale Größe des automatischen Bildlaufs an
AutoScrollPosition	Liefert die Position des automatischen Bildlaufs
BackColor	Gibt die Hintergrundfarbe für das Formular an
BackgroundImage	Gibt das Hintergrundbild für das Formular an
CancelButton	Gibt bei Dialogen an, welcher Button zum Abbruch vorgesehen ist. Bei einem Klick auf den angegebenen Button wird <code>DialogResult</code> automatisch auf den entsprechenden Wert des Buttons eingestellt.
CanFocus	Gibt an, ob das Formular den Fokus erhalten kann
Capture	Gibt an, ob die Maus das Formular erfasst hat
CausesValidation	Gibt an, ob alle untergeordneten Steuerelemente, die eine Validierung erfordern, validiert werden, wenn das Formular den Fokus erhält
ClientSize	Liefert die Größe des Client-Bereichs des Formulars
ContextMenu	Gibt das Kontextmenü für dieses Fenster an
ControlBox	Gibt an, ob das Fenster ein Systemmenü enthalten soll
Controls	Enthält eine Liste der Steuerelemente des Formulars
Cursor	Gibt an, wie der Mauszeiger über dem Formular angezeigt werden soll

Tabelle 17.2: Eigenschaften der Klasse Form

Eigenschaft	Bedeutung
DialogResult	Gibt an, welcher Ergebniswert (im Falle dafür, dass das Formular als Dialog angezeigt wird) zurückgeliefert wird
DisplayRectangle	Liefert ein Rechteck, das den sichtbaren Bereich des Formulars beinhaltet
Dock	Gibt an, an welchen Rändern des Formulars Steuerelemente angedockt werden können
DockPadding	Ruft ab, welchen Leerraum die Ränder des Formulars für das Andocken haben
Focused	Gibt an, ob das Formular den Fokus hat
Font	Legt den verwendeten Zeichensatz fest
ForeColor	Legt die Vordergrundfarbe des Formulars fest (die Schriftfarbe)
FormBorderStyle	Legt die Art der Umrandung für das Formular fest. Hier kann eingestellt werden, ob es sich um ein größenveränderliches Formular, einen Dialog (nicht größenveränderlich) oder eine Toolbox (mit kleiner Titelleiste) handeln soll.
Handle	Liefert einen Handle auf das Formular
HasChildren	Gibt an, ob das Formular untergeordnete Steuerelemente enthält
Height	Gibt die Höhe des Formulars an
HelpButton	Legt fest, ob ein Hilfe-Button in der Titelleiste des Formulars angezeigt werden soll
Icon	Legt das Icon für das Formular fest
KeyPreview	Legt fest, ob alle Tastendrucke zuerst an das Formular gesendet werden, bevor sie an das Steuerelement gehen. Damit können beispielsweise Tastenkürzel ausgewertet werden.
Left	Gibt die x-Koordinate der linken oberen Ecke des Formulars an
MaximizeBox	Legt fest, ob ein Button zum Maximieren des Formulars in dessen Titelleiste angezeigt werden soll
MaximumSize	Legt die maximale Größe des Formulars fest
MdiChildren	Ruft ein Array aus Formularen ab, die die untergeordneten MDI-Formulare darstellen
MdiParent	Gibt an, ob es sich bei dem Formular um ein MDI-Containerformular handelt

Tabelle 17.2: Eigenschaften der Klasse Form (Fortsetzung)

Eigenschaft	Bedeutung
Menu	Gibt das Hauptmenü des Formulars an
MinimizeBox	Legt fest, ob ein Button zum Minimieren des Formulars in dessen Titelleiste angezeigt wird
MinimumSize	Legt die minimale Größe des Formulars fest
Modal	Gibt an, ob das Formular modal oder nicht-modal angezeigt wird
Name	Der Name des Formulars
Opacity	Legt fest, wie transparent das Formular angezeigt werden soll
ShowInTaskbar	Legt fest, ob das Formular in der Taskbar angezeigt wird
Size	Liefert die Größe des Formulars
SizeGripStyle	Legt fest, ob der Größenveränderungspunkt in der rechten unteren Ecke des Formulars dargestellt wird. Der Wert ist vom Typ <code>SizeGripStyle</code> .
StartPosition	Dient zum Festlegen der Startposition des Formulars
Text	Dient dem Festlegen der <i>Caption</i> (also des Textes in der Titelleiste) des Formulars
Top	Gibt die y-Position des oberen Rands des Formulars an
TopMost	Legt fest, ob das Formular innerhalb der Applikation immer im Vordergrund angezeigt werden soll
TransparencyKey	Dient zum Festlegen der Farbe, die die transparenten Bereiche des Formulars festlegt
Width	Legt die Breite des Formulars fest
WindowState	Zum Festlegen des Status dieses Formulars. Das Formular kann maximiert, minimiert oder normal angezeigt werden. Der zugewiesene Wert ist vom Typ <code>FormWindowState</code> .

Tabelle 17.2: Eigenschaften der Klasse Form (Fortsetzung)

17.4.2 Formulare dynamisch erzeugen

Innerhalb eines Programms ist es sehr oft notwendig, ein Formular (z.B. einen Dialog) anzuzeigen und wieder zu schließen, möglicherweise auch mit einer Auswertung der darin getätigten Eingaben. Zum Erzeugen eines Formulars rufen Sie einfach dessen Konstruktor auf, zum Anzeigen dienen die Methoden `Show()` und `ShowDialog()`. Wie der Name schon sagt, zeigt `Show()` ein nicht-modales Dialogfeld, d.h. die Anwendung im Hintergrund läuft weiter, `ShowDialog()` ein modales Fenster, d.h. dieses Fenster muss

Show() und ShowDialog()

zuerst geschlossen werden, bevor weitere Aktionen innerhalb der Anwendung durchgeführt werden können.

Das folgende kleine Beispiel zeigt, wie ein About-Dialog erzeugt und angezeigt werden kann. Auf dem angezeigten Fenster befindet sich außer einer Anzeige lediglich ein Button zum Beenden. Die Eigenschaft `AcceptButton` dieses Fensters wird auf diesen Button eingestellt, so dass kein Code benötigt wird, um das Fenster zu schließen. Das Ganze besteht lediglich aus zwei Code-Zeilen:

```
Private Sub btnAbout_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnAbout.Click

    Dim aFrmAbout As New FrmAbout()
    aFrmAbout.ShowDialog()
End Sub
```

Sie haben auch die Möglichkeit, nach der Erzeugung des Formulars auf dessen Eigenschaften zuzugreifen, beispielsweise um den Text in der Titelleiste zu ändern. Eine weitere Zeile genügt.

```
Private Sub btnAbout_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnAbout.Click

    Dim aFrmAbout As New FrmAbout()
    aFrmAbout.Text = "Über dieses Programm"
    aFrmAbout.ShowDialog()
End Sub
```

Das Resultat zeigt Abbildung 17.7.

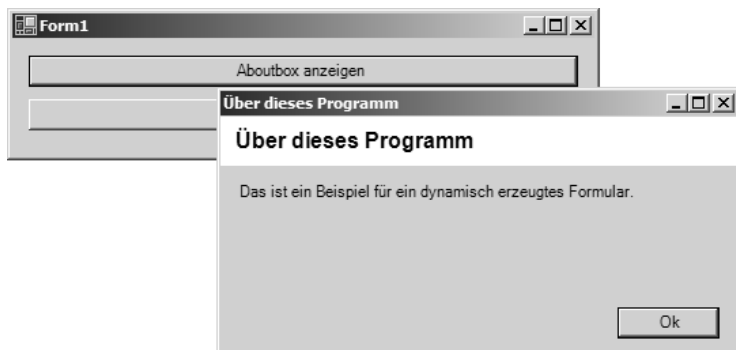


Abbildung 17.7: Eine Aboutbox dynamisch generiert [AboutBox.sln]

17.4.3 Zugriff auf Komponenten eines Formulars

Die Eigenschaft `Controls` enthält alle auf einem Formular angeordneten Steuerelemente in Form einer `Collection`. Diese können Sie mit Hilfe der `For Each`-Anweisung durchlaufen und so auf alle Steuerelemente zugreifen. Damit ist es beispielsweise möglich, die angezeigten Werte einer oder mehrerer Komponenten zu ändern.

Controls

Nützlich ist hierbei die Eigenschaft `Tag`, die jedes Steuerelement besitzt. Bei gleichen Steuerelementen kann diese zur Unterscheidung benutzt werden. Das folgende Beispiel benutzt `Tag`, um einigen `Label`-Komponenten eine andere Hintergrundfarbe zuzuweisen. Auf dem Formular befinden sich noch weitere Komponenten; durch einen Typenvergleich wird sichergestellt, dass nur auf die Komponenten zugegriffen wird, die vom Typ `Label` sind.

Tag

```
'=====
' AccessControls.sln
' Copyright 2002 Frank Eller
' Greift auf Komponenten des Formulars zu
'=====

Public Class Form1
    Inherits System.Windows.Forms.Form

    #Region " Windows Form Designer generated code "

        ' Der Code, der von VS.NET eingefügt wurde, wird
        ' aus Übersichtsgründen nicht dargestellt.

    #End Region

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

        Dim o As Object
        Dim aTag As Integer

        For Each o In Me.Controls
            If o.GetType() Is Label1.GetType() Then
                aTag = (CType(o, Label).Tag)

                Select Case aTag
                    Case 1
                        CType(o, Label).BackColor = Color.Blue
                    Case 2
```

```

        CType(o, Label).BackColor = Color.Red
    Case 3
        CType(o, Label).BackColor = Color.Cyan
    Case 4
        CType(o, Label).BackColor = Color.Pink
    Case 5
        CType(o, Label).BackColor = Color.Black
    Case 6
        CType(o, Label).BackColor = Color.Yellow
    End Select
End If
Next
End Sub

Private Sub btnExit_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnExit.Click

    Me.Close()
End Sub
End Class

```

Listing 17.3: Zugriff auf die Steuerelemente eines Formulars [AccessControls.sln]

Einen Screenshot sehen Sie in Abbildung 17.8.

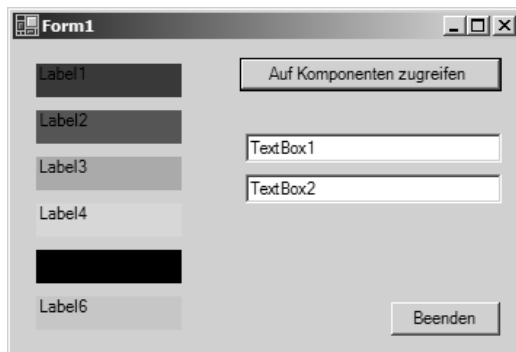


Abbildung 17.8: Zugriff auf die Steuerelemente eines Formulars [AccessControls.sln]

17.4.4 Zugriff auf Komponenten des Hauptformulars

Der Zugriff auf die Komponenten des Hauptformulars aus einem anderen Formular heraus ist nicht ohne weiteres möglich. Hier muss ein wenig nachgeholfen werden. Wenn man allerdings verstanden hat, wie der Start eines Programms vor sich geht, ist das kein Problem mehr.

Es gibt zwei Möglichkeiten, das Hauptformular eines Programms festzulegen: Entweder wird das Hauptformular in Visual Basic in den Projektoptionen festgelegt oder es wird eine Methode `Main()` benutzt, ebenso wie in C#, in der eine Instanz des Hauptformulars an die Methode `Application.Run()` übergeben wird. In C# gibt es nur diese eine Möglichkeit. Nun ist aber bekannt, dass die Klasse, in der `Main()` deklariert ist, beliebig sein kann. Visual Studio baut sie immer in die Klasse des Hauptformulars mit ein (in C#, in Visual Basic wird sie nicht benutzt). Das verhindert allerdings einen Zugriff auf die Komponenten dieses Hauptformulars von außerhalb.

Festlegen des Hauptformulars

Wollen Sie dennoch darauf zugreifen, müssen Sie die Methode `Main()` auslagern. Dazu erstellen Sie einfach eine neue Klasse beliebigen Namens, z.B. `ProjectMain`, und deklarieren darin die neue `Main()`-Methode. In dieser können Sie nun ebenfalls die Methode `Application.Run()` verwenden und dieser eine Instanz des Hauptformulars übergeben. In C# müssen Sie darauf achten, die bereits bestehende `Main()`-Methode zu löschen, in Visual Basic müssen Sie in den Projektoptionen den Programmstart auf die `Main()`-Methode umstellen. Listing 17.4 zeigt diesen ersten Schritt.

Main() auslagern

```
Public Class ProjectMain

    Public Shared Sub Main()
        Application.Run(New Form1())
    End Sub

End Class

Public Class Form1
    Inherits System.Windows.Forms.Form

    #Region " Windows Form Designer generated code "

    'Der Code, der von VS.NET eingefügt wurde, wird
    'aus Übersichtsgründen nicht dargestellt.

    #End Region

End Class
```

Listing 17.4: Eine neue Main-Methode für das Programm

Der zweite Schritt ist nun, eine neue Instanz des Hauptformulars zu erzeugen, die dann an die Methode `Application.Run()` übergeben wird. Dazu erstellen Sie in der neuen Klasse ein statisches

Instanz des Hauptformulars erzeugen

(shared) Feld, erzeugen eine Instanz des Hauptformulars und benutzen diese dann als Hauptformular. Den kompletten Quellcode zeigt Listing 17.5.

```
Public Class ProjectMain

    Public Shared mainForm As New Form1()

    Public Shared Sub Main()
        Application.Run(mainForm)
    End Sub

End Class

Public Class Form1
    Inherits System.Windows.Forms.Form

    #Region " Windows Form Designer generated code "

        'Der Code, der von VS.NET eingefügt wurde, wird
        'aus Übersichtsgründen nicht dargestellt.

    #End Region

    Private Sub btnShowForm_Click( _
        ByVal sender As System.Object, _
        ByVal e As System.EventArgs) _
        Handles btnShowForm.Click

        Dim aForm As New Form2()
        aForm.Show()

    End Sub

    Private Sub btnExit_Click( _
        ByVal sender As System.Object, _
        ByVal e As System.EventArgs) _
        Handles btnExit.Click

        Me.Close()
    End Sub
End Class
```

Listing 17.5: Der Quellcode des Hauptformulars [AccessMainform.sln]

Jetzt ist es möglich, über ein zweites Formular – und zwar über das statische Feld der neuen Klasse `ProjectMain` – auf die Elemente des Hauptformulars zuzugreifen. Wie Sie sehen können, wird im Quellcode des Hauptformulars bereits ein weiteres Formular erzeugt, von dem aus nun auf das Hauptformular zugegriffen

wird. Der Einfachheit halber wird lediglich die Eigenschaft `Text` des Hauptformulars geändert.

```
Public Class Form2
    Inherits System.Windows.Forms.Form

    #Region " Windows Form Designer generated code "

    'Der Code, der von VS.NET eingefügt wurde, wird
    'aus Übersichtsgründen nicht dargestellt.

    #End Region

    Private Sub btnClose_Click( _
        ByVal sender As System.Object, _
        ByVal e As System.EventArgs) _
        Handles btnClose.Click

        Me.Close()
    End Sub

    Private Sub btnChangeText_Click( _
        ByVal sender As System.Object, _
        ByVal e As System.EventArgs) _
        Handles btnChangeText.Click

        ProjectMain.mainForm.Text = txtChangeText.Text
    End Sub
End Class
```

Listing 17.6: Der Quellcode des zweiten Formulars [AccessMainform.sln]

Das Hauptformular erreichen Sie nun über `ProjectMain.mainForm`. Einen Screenshot der beiden Fenster zur Laufzeit sehen Sie in Abbildung 17.9.



Abbildung 17.9: Ändern des Titels eines Hauptformulars aus einem anderen Formular heraus [AccessMainform.sln]

Dass das erste Formular immer noch das Hauptformular ist, sehen Sie daran, dass das Programm nach wie vor beendet wird, wenn Sie das erste Formular schließen.

17.4.5 MDI-Applikationen

MDI steht für *Multiple Document Interface*. Dabei bildet das Hauptformular einen Container für mehrere, untergeordnete Formulare. Auch hierfür bietet das .NET Framework eine Unterstützung, die sehr leicht zu implementieren ist.

Schlüssel zu einer MDI-Applikation ist die Eigenschaft `IsMdiContainer` des Hauptformulars. Wenn Sie diese Eigenschaft auf `True` setzen, ändert sich das Formular und zeigt den Arbeitsbereich nun dunkel an.

MDI-Fenster
hinzufügen

Von diesem Moment an können Sie MDI-Child-Fenster hinzufügen. Das funktioniert, indem ein neues Fenster erzeugt und dessen Eigenschaft `MdiParent` auf das Hauptfenster gesetzt wird. Damit ist es nun ein Child des Hauptfensters und über die speziellen MDI-Methoden dieses Fensters erreichbar.

MDI-Fenster
anordnen

Eine dieser Methoden ist `LayoutMdi()`. Diese Methode ändert die Anordnung der MDI-Fenster. Übliche Anordnungen sind horizontal geteilt, vertikal geteilt oder überlappend. Diese Darstellungen können mit diesem Befehl erreicht werden. Übergeben wird ein Parameter vom Typ `MdiLayout`, der festlegt, wie die Child-Fenster angeordnet werden sollen.

Weitere Eigenschaften im Zusammenhang mit MDI liefert Tabelle 17.3.

Eigenschaft	Bedeutung
<code>IsMdiContainer</code>	Legt fest, ob das aktuelle Dokument als Hauptfenster einer MDI-Applikation dienen soll
<code>IsMdiChild</code>	Über diese Eigenschaft kann ermittelt werden, ob es sich bei dem Fenster um ein MDI-Child-Fenster handelt.
<code>MdiParent</code>	Legt das Hauptfenster eines MDI-Childs fest
<code>ActiveMdiChild</code>	Liefert das gerade aktive MDI-Child-Fenster zurück
<code>MdiChildren</code>	Liefert eine Auflistung alle MDI-Child-Fenster zurück

Tabelle 17.3: Eigenschaften eines Formulars bzgl. MDI-Applikationen

Bei der Erstellung von MDI-Applikationen gelten grundsätzlich zwei Verhaltensweisen, unabhängig von irgendwelchen Einstellungen:

- ▶ Wird das Hauptformular geschlossen, werden auch die Child-Fenster geschlossen. Deren `Closed`- bzw. `Closing`-Ereignisse werden nacheinander abgearbeitet.
- ▶ Ein MDI-Child-Fenster besitzt immer Buttons zum Maximieren, Minimieren und Schließen, unabhängig von den Einstellungen der entsprechenden Eigenschaften.

Ein Mini-Editor im MDI-Stil

Als Beispiel soll eine kleine Editor-Applikation dienen, mit der Texte bearbeitet werden können. Das Ganze ist allerdings nur ein Gerüst für einen evtl. richtigen Editor, der dann auch Dateiformate erkennt oder Syntax-Highlighting beinhaltet.

Der Editor soll allerdings bereits Basisfunktionalität mitbringen. Dazu gehört, dass das Clipboard mit einbezogen wird. Dafür steht die Klasse `Clipboard` zur Verfügung, die diese Zwischenablage repräsentiert. Im Beispiel wird sie nur dazu verwendet, das Vorhandensein von Daten zu kontrollieren.

Clipboard

Weiterhin sollen eine Undo- und eine Redo-Funktion eingebaut werden und natürlich soll das Programm Texte laden und speichern können. Die Undo- und Redo-Funktionen sollen auch noch anzeigen, welches denn die Aktion ist, die rückgängig gemacht bzw. wiederholt werden soll.

Das Erste ist das Erstellen eines Basisformulars für die MDI-Fenster. Dieses soll natürlich nur einen Editor beinhalten, in diesem Fall eine Komponente vom Typ `RichTextBox`. Diese bringt schon alles an Funktionalität mit, was für die Applikation benötigt wird. Sie kann Daten in die Zwischenablage kopieren, Daten aus der Zwischenablage einfügen, Dateien laden und speichern und hat auch eine Undo- und eine Redo-Funktion. Über die Eigenschaften `UndoActionName` und `RedoActionName` kann auch die zuletzt erfolgte Aktion ermittelt werden. Damit wäre alles beisammen, was für die Applikation benötigt wird.

Basisformular

Die eingefügte Komponente wird über die Eigenschaft `Dock` so eingestellt, dass sie immer die gesamte Größe des Formulars einnimmt. Weiterhin, da die Texte ja für jedes Child-Fenster getrennt

Dock

geladen bzw. gespeichert werden können, werden zwei öffentliche Methoden implementiert, je eine zum Laden und zum Speichern.

Damit diese Methoden auch richtig funktionieren und nicht innerhalb des Hauptformulars jedes Mal der Dateiname gespeichert werden muss, übernimmt das auch das Child-Formular. Die Eigenschaft `CompleteFileName` soll dabei immer den aktuellen Dateinamen enthalten. Bei einer Zuweisung an diese Eigenschaft wird dann auch schon gleich der Text im Kopf des Fensters korrekt auf den Dateinamen (allerdings ohne den Pfad) eingestellt. Listing 17.7 zeigt den Quelltext des Formulars.

```
'=====
'MDITest.sln
'Copyright 2002 Frank Eller
'MdiBase - Basisformular für MDI-Children
'=====

Imports System.IO

Public Class MdiBase
    Inherits System.Windows.Forms.Form

    #Region " Windows Form Designer generated code "

        'Der Code, der von VS.NET eingefügt wurde, wird
        'aus Übersichtsgründen nicht dargestellt.

    #End Region

    Private _completeFileName As String = String.Empty

    Public Property CompleteFileName() As String
        Get
            Return _completeFileName
        End Get
        Set(ByVal Value As String)
            _completeFileName = Value
            Me.Text = Path.GetFileName(_completeFileName)
        End Set
    End Property

    Public Sub LoadDocument()

        Me.Editor.LoadFile(Me._completeFileName, _
            RichTextBoxStreamType.PlainText)

    End Sub
```

```

Public Sub SaveDocument()

    If Me.Text = "Neu" Then
        If dlgSaveFile.ShowDialog() = DialogResult.OK Then
            Me.CompleteFileName = dlgSaveFile.FileName
        End If
    End If
    If Me.Text <> "Neu" Then
        Me.Editor.SaveFile(_completeFileName, _
            RichTextBoxStreamType.PlainText)
    End If

End Sub

Private Sub MdiBase_Closing( _
    ByVal sender As Object, _
    ByVal e As System.ComponentModel.CancelEventArgs) _
    Handles MyBase.Closing

    Me.SaveDocument()

End Sub
End Class

```

Listing 17.7: Die Basis für ein MDI-Child-Fenster [MDITest.sln]

Am Quelltext ist zu erkennen, dass die enthaltene Datei automatisch gespeichert wird, wenn das Fenster geschlossen wird. Da dieses Ereignis, nämlich `Closing`, auch dann aufgerufen wird, wenn die gesamte Applikation beendet wird, verlieren wir also keine Daten.

Besitzt das Dokument noch keinen Namen (dann steht in der Titelleiste das Wort »Neu«, das bei der Erzeugung des Fensters eingestellt wird), wird zum Speichern ein Dialog aufgerufen, in dem dann ein Dateiname eingegeben werden kann. Ist das Speichern nicht erwünscht, klickt man dort einfach auf **ABBRECHEN**.

Das Hauptfenster liefert natürlich auch das Hauptmenü für die Applikation. Dieses ist folgendermaßen aufgebaut:

Hauptmenü

- ▶ Datei: Neu, Speichern, Laden, Beenden
- ▶ Bearbeiten: Rückgängig, Wiederholen, Ausschneiden, Kopieren, Einfügen
- ▶ Fenster: Übereinander, Überlappend, Nebeneinander

Die Namen der einzelnen Menüpunkte wurden so gewählt, dass sie eindeutig sind.

Dokumente erzeugen, laden und speichern

Ein neues Dokument wird einfach generiert, indem ein neues Formular vom Typ `MdiBase` erzeugt und seine Eigenschaft `MdiParent` auf das Hauptfenster der Applikation eingestellt wird. Der Text des neuen Formulars wird auf »Neu« eingestellt. Damit kann später erkannt werden, ob das Dokument schon gespeichert wurde. Hier gibt es sicherlich bessere Methoden, aber für diese einfache Applikation genügt es.

```
Private Sub mnuFileNew_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles mnuFileNew.Click

    ' --- Erstellen eines neuen Fensters
    Dim doc As New MdiBase()
    doc.Text = "Neu"
    doc.MdiParent = Me
    doc.Editor.Text = String.Empty
    doc.Show()

End Sub
```

Listing 17.8: Ein neues MDI-Fenster [MDIText.sln]

Die Ereignisbehandlungsroutine zum Laden sieht auch nicht viel anders aus, nur dass diesmal die Methode `LoadDocument()` des Child-Fensters aufgerufen wird.

```
Private Sub mnuFileLoad_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles mnuFileLoad.Click

    ' --- Deklarationen
    Dim doc As MdiBase

    ' --- Erstellen eines neuen Fensters
    ' --- und Laden der Datei
    If dlgOpenFile.ShowDialog() = _
        DialogResult.OK Then

        ' --- Jetzt erstellen
        doc = New MdiBase()
        doc.MdiParent = Me
```



```

doc.CompleteFileName = dlgOpenFile.FileName
doc.LoadDocument()
doc.Show()

```

```

End If
End Sub

```

Listing 17.9: Laden einer Datei [MDITest.sln]

Da das MDI-Fenster bereits alle Daten beinhaltet und auch beim Speichern automatisch der Dialog aufgerufen wird, wenn noch kein Dateiname festgelegt wurde, ist die Methode zum Speichern wohl die einfachste Methode, da sie (außer der notwendigen Umwandlung) nur aus dem Aufruf von `SaveDocument()` besteht.

```

Private Sub mnuFileSave_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles mnuFileSave.Click

    ' --- Speichern des Textes im aktiven Fenster
    Dim doc As MdiBase
    doc = CType(Me.ActiveMdiChild, MdiBase)
    doc.SaveDocument()

End Sub

```

Listing 17.10: Speichern einer Datei [MDITest.sln]

Das Menü Bearbeiten

Die Funktionen für das BEARBEITEN-Menü sind ein wenig umfangreicher. Zunächst muss, damit die Menüeinträge funktionieren, auch wirklich ein Child-Fenster existieren. Daher wird dies kontrolliert, sobald das Menü geöffnet wird. Damit wird sichergestellt, dass die entsprechenden Funktionen nicht aufgerufen werden können, wenn kein Child-Fenster existiert. Wenn denn ein Child-Fenster aktiv ist, wird gleichzeitig ermittelt, ob es Undo-/Redo-Funktionen gibt, die angezeigt werden können.

```

Private Sub mnuEdit_Popup( _
    ByVal sender As Object, _
    ByVal e As System.EventArgs) _
    Handles mnuEdit.Popup

    ' --- Undo und Redo checken
    ' --- Deklarationen
    Dim undoStr As String = "&Rückgängig "
    Dim redoStr As String = "&Wiederholen "
    Dim doc As MdiBase

```

```

' --- Testen, ob mdiChildren da sind
If Me.ActiveMdiChild Is Nothing Then
' --- Deaktivieren der Menüpunkte
mnuRedo.Enabled = False
mnuUndo.Enabled = False
mnuEditCut.Enabled = False
mnuEditCopy.Enabled = False
mnuEditPaste.Enabled = False
Exit Sub
Else
mnuRedo.Enabled = True
mnuUndo.Enabled = True
mnuEditCut.Enabled = True
mnuEditCopy.Enabled = True
mnuEditPaste.Enabled = True
End If

doc = CType(Me.ActiveMdiChild, MdiBase)

If doc.Editor.CanUndo Then
undoStr += doc.Editor.UndoActionName
End If

If doc.Editor.CanRedo Then
redoStr += doc.Editor.RedoActionName
End If

mnuUndo.Text = undoStr
mnuRedo.Text = redoStr

End Sub

```

Listing 17.11: Kontrolle beim Öffnen des Bearbeiten-Menüs [MDITest.sln]

Die übrigen Funktionen dieses Menüs sind nicht weiter schwierig. Listing 17.12 zeigt die Methoden im Zusammenhang.

```

Private Sub mnuEditCut_Click( _
ByVal sender As System.Object, _
ByVal e As System.EventArgs) _
Handles mnuEditCut.Click

' --- Ausschneiden des markierten Textes
' --- im aktuellen Fenster
Dim doc As MdiBase
doc = CType(Me.ActiveMdiChild, MdiBase)
doc.Editor.Cut()

End Sub

Private Sub mnuEditCopy_Click( _
ByVal sender As System.Object, _

```

```
ByVal e As System.EventArgs) _
Handles mnuEditCopy.Click

' --- Kopieren des markierten Textes im
' --- aktuellen Fenster
Dim doc As MdiBase
doc = CType(Me.ActiveMdiChild, MdiBase)
doc.Editor.Copy()

End Sub

Private Sub mnuEditPaste_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
Handles mnuEditPaste.Click

' --- Einfügen aus der Zwischenablage
Dim clpData As DataObject = _
    Clipboard.GetDataObject
Dim doc As MdiBase

doc = CType(Me.ActiveMdiChild, MdiBase)
If clpData.GetDataPresent(DataFormats.Text) Then
    doc.Editor.Paste()
End If

End Sub

Private Sub mnuUndo_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
Handles mnuUndo.Click

Dim doc As MdiBase

doc = CType(Me.ActiveMdiChild, MdiBase)
If doc.Editor.CanUndo Then
    doc.Editor.Undo()
End If

End Sub

Private Sub mnuRedo_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
Handles mnuRedo.Click

Dim doc As MdiBase

doc = CType(Me.ActiveMdiChild, MdiBase)
If doc.Editor.CanRedo Then
```

```
        doc.Editor.Redo()
    End If
```

```
End Sub
```

Listing 17.12: Die Methoden des Bearbeiten-Menüs [MDITest.sln]

Das Fenster-Menü

Die Methoden des Fenster-Menüs sind ebenfalls nicht weiter kompliziert, da sie ja eigentlich nur aus einem Aufruf der Methode `LayoutMdi()` bestehen. Listing 17.13 zeigt die Methoden.

```
Private Sub mnuWindowTile_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles mnuWindowTile.Click
```

```
    Me.LayoutMdi(MdiLayout.TileHorizontal)
```

```
End Sub
```

```
Private Sub mnuWindowTileVert_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles mnuWindowTileVert.Click
```

```
    Me.LayoutMdi(MdiLayout.TileVertical)
```

```
End Sub
```

```
Private Sub mnuWindowCascade_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles mnuWindowCascade.Click
```

```
    Me.LayoutMdi(MdiLayout.Cascade)
```

```
End Sub
```

Listing 17.13: Die Methoden des Menüs Fenster [MDITest.sln]

Mit diesen wenigen Anweisungen haben Sie gerade einen kompletten Text-Editor programmiert. Zur Laufzeit sieht dieser dann so aus, wie in Abbildung 17.10 dargestellt. Natürlich fehlen hier noch Funktionen, beispielsweise zum Einstellen der Schriftart, diese können Sie aber sicherlich sehr schnell selbst implementieren.

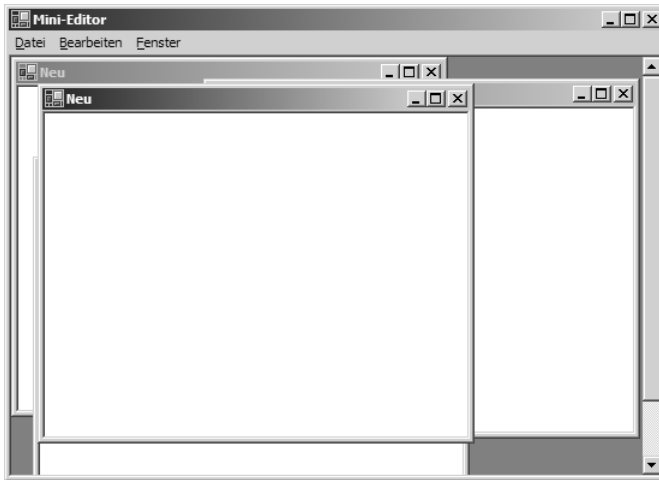


Abbildung 17.10: Das Programm zur Laufzeit [MDITest.sln]

17.5 Arbeiten mit den Steuerelementen

Die Anzahl der standardmäßig verfügbaren Steuerelementen ist recht hoch, daher können an dieser Stelle nicht alle besprochen werden. Der Fokus liegt deswegen auf einigen Anwendungsbeispielen, die in Ihren eigenen Anwendungen öfter vorkommen könnten.

17.5.1 Dynamische Erzeugung von Steuerelementen

Da Steuerelemente ja auch nur Klassen sind, können sie auch dynamisch erzeugt werden. Wenn Sie ein Steuerelement über seinen Konstruktor erzeugen, müssen Sie ebenso wie bei der Erzeugung im Design-Modus von Visual Studio die benötigten Eigenschaften festlegen. Selbstverständlich können Sie die erzeugte Komponente auch wieder entfernen.

Im Beispiel soll eine gewisse Anzahl an Textboxen erzeugt werden, wenn ein Button geklickt wird. Die Anzahl wird mit Hilfe einer weiteren Textbox vorgegeben, sollte zehn aber nicht überschreiten. Zusätzlich sollen überflüssige Textboxen ebenfalls entfernt werden, d.h. wenn bereits sechs Textboxen erzeugt wurden, aber nur drei neu erzeugt werden sollen, bleiben die ersten drei stehen und die letzten drei werden gelöscht. Außerdem soll es natürlich auch möglich sein, alle Textboxen zu löschen.

Den Quellcode zu diesem kleinen Programm sehen Sie in Listing 17.14.

```
Public Class Form1
    Inherits System.Windows.Forms.Form

    #Region " Windows Form Designer generated code "

        'Der Code, der von VS.NET eingefügt wurde, wird
        'aus Übersichtsgründen nicht dargestellt.

    #End Region

    Private TB(9) As TextBox

    Private Sub RemoveItem(ByVal number As Integer)

        ' --- Entfernen einer Textbox
        If Not TB(number) Is Nothing Then
            Me.Controls.Remove(TB(number))
            TB(number) = Nothing
        End If
    End Sub

    Private Sub btnCreateComps_Click( _
        ByVal sender As System.Object, _
        ByVal e As System.EventArgs) _
        Handles btnCreateComps.Click

        ' --- Deklarationen
        Dim aCount As Integer = Convert.ToInt32(txtCount.Text)
        Dim i As Integer

        ' --- Nicht mehr als 10 Items
        If aCount > 10 Then aCount = 10

        ' --- Textboxen erzeugen und Position festlegen
        For i = 0 To aCount - 1
            If TB(i) Is Nothing Then
                TB(i) = New TextBox()
                TB(i).Left = 8 + ((i * 50) + 4)
                TB(i).Top = 68
                TB(i).Width = 40
                TB(i).Height = 13
                Me.Controls.Add(TB(i))
            End If
        Next

        ' --- Überflüssige Textboxen entfernen
        If aCount < 10 Then
            For i = aCount To 9
```

```

        RemoveItem(i)
    Next
End If

End Sub

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

    ' --- Alle Textboxen entfernen
    Dim i As Integer
    For i = 0 To 9
        RemoveItem(i)
    Next
End Sub

Private Sub btnExit_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnExit.Click

    Me.Close()
End Sub
End Class

```

Listing 17.14: Dynamisches Erzeugen und Löschen von Textboxen
[CreateComponents.sln]

Einen Screenshot der Anwendung zeigt Abbildung 17.11.

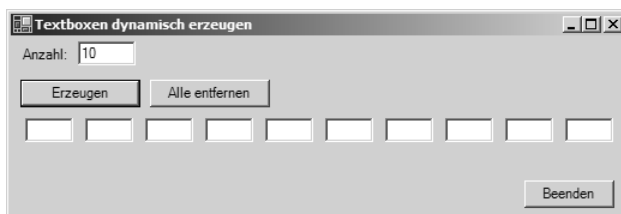


Abbildung 17.11: Dynamische Erzeugung von Textboxen [CreateComponents.sln]

17.5.2 Eingaben kontrollieren

Eingaben einschränken

Über die Ereignisbehandlungsroutine `KeyPress` ist es auf einfache Art möglich, Eingaben des Benutzers zu überwachen oder auch zu ändern. Der Parameter `e` enthält eine Eigenschaft `KeyChar`, die die

KeyPress

gerade gedrückte Taste als Zeichen beinhaltet. Diese kann kontrolliert und dann entsprechend weiterbehandelt oder nicht mehr weiter behandelt werden. Dafür ist der zweite Parameter, `Handled`, zuständig. Damit ist es sehr einfach, beispielsweise die Eingabe in einer Textbox auf reine Zahlen zu beschränken. Wird die Eigenschaft `Handled` auf `True` gesetzt, wird das Zeichen nicht weitergegeben, d.h. es erscheint nicht in der Textbox. Andernfalls erscheint es. Eine entsprechende Methode sieht etwa so aus:

```
Private Sub txtIsDigit_KeyPress( _
    ByVal sender As Object, ByVal e As _
    System.Windows.Forms.KeyPressEventArgs) _
    Handles txtIsDigit.KeyPress

    If (Not Char.IsDigit(e.KeyChar)) And _
        (e.KeyChar <> vbBack) Then
        e.Handled = True
    End If
End Sub
```

Listing 17.15: Abweisen eines Zeichens in einer Textbox [CheckKeys.sln]

Die Kontrolle auf den Wert `vbBack`, der die Backspace-Taste repräsentiert, ist notwendig, da der Anwender ansonsten seine Eingaben nicht mehr löschen könnte.

Eingaben ändern

Eine weitere Möglichkeit, die manchmal in Betracht kommt, ist, das Zeichen zu ändern. Beispielsweise gibt es Programme, bei denen statt eines Kommas ein Punkt erscheinen soll (oder umgekehrt). In diesem Fall müsste der Tastendruck abgefangen und entsprechend abgeändert werden.

SelectedText Da der Parameter `e.KeyChar` nur lesbar ist, wird er wie in der vorherigen Methode zunächst einmal abgewiesen, wenn er einem bestimmten Zeichen entspricht. Danach wird das neue Zeichen einfach der Eigenschaft `SelectedText` der entsprechenden Textbox zugewiesen. `SelectedText` repräsentiert immer den aktuell markierten Text oder, falls keine Markierung vorhanden ist, die aktuelle Position des Cursors innerhalb eines Textfelds. Die Methode zum Ändern eines Zeichens sieht damit folgendermaßen aus:

```
Private Sub txtComma_KeyPress( _
    ByVal sender As Object, _
    ByVal e As _
    System.Windows.Forms.KeyPressEventArgs) _
```



```
Handles txtComma.KeyPress

If e.KeyChar = "." Then
    e.Handled = True
    txtComma.SelectedText = ","
End If

End Sub
```

Listing 17.16: Ändern eines Zeichens in einer Textbox [CheckKeys.sln]

Einen Screenshot des laufenden Programms sehen Sie in Abbildung 17.12.

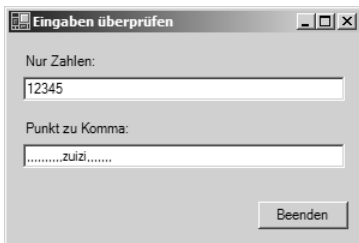


Abbildung 17.12: Das Programm CheckKeys.exe zur Laufzeit

17.5.3 Drag&Drop

Drag&Drop ist eine allgemein als benutzerfreundlich anerkannte Methode, um Daten von einem Element in ein anderes zu kopieren oder zu verschieben. Auch im .NET Framework ist Drag&Drop-Unterstützung implementiert. Die entscheidenden Methoden bzw. Ereignisse dabei sind:

- ▶ `DoDragDrop()` zum Starten der Drag&Drop-Operation
- ▶ `DragEnter` zum Festlegen, ob die Drag&Drop-Operation durchgeführt werden darf
- ▶ `DragLeave`, das Ereignis, welches auftritt, wenn etwas aus einem Bereich gezogen wird
- ▶ `DragEnter`, das Ereignis, das auftritt, wenn etwas in einen Bereich gezogen wird
- ▶ `DragDrop`, das Ereignis, das auftritt, wenn der Vorgang beendet ist (also die Maustaste losgelassen wurde)

DragEventArgs Alle Ereignisse erhalten einen Parameter vom Typ `DragEventArgs`, in dem bestimmte, für den Drag&Drop-Vorgang wichtige Informationen enthalten sind bzw. gespeichert werden. Tabelle 17.4 liefert eine Übersicht.

Eigenschaft	Bedeutung
AllowedEffect	Legt fest, welche Arten von Drag&Drop-Operationen von Seiten des Quellelements erlaubt sind
Data	Enthält die Daten, die verschoben oder kopiert werden sollen
Effect	Legt fest, welche Arten von Drag&Drop-Operationen von Seiten des Zielobjekts erlaubt sind
KeyState	Enthält den Status der Sondertasten, wie <code>Shift</code> , <code>Strg</code> usw.
X	Enthält die x-Koordinate des Mausursors
Y	Enthält die y-Koordinate des Mausursors

Tabelle 17.4: Eigenschaften des Parameters `DragEventArgs`

AllowDrop Um ein Steuerelement Drag&Drop-fähig zu machen, müssen Sie die Eigenschaft `AllowDrop` der Komponente auf `True` setzen. Damit ist diese nun in der Lage, per Maus gezogene Daten zu empfangen.

Drag&Drop mit Daten

Anhand eines kleinen Beispiels soll gezeigt werden, wie Daten von einem Element in ein anderes verschoben werden können. Die Oberfläche des Programms besteht aus fünf Listboxen. Die erste Listbox ist bereits mit Werten gefüllt. Diese Werte sollen nun auf die anderen Listboxen verteilt werden können. Den Aufbau des Programms zeigt Abbildung 17.13.

Eine Drag&Drop-Operation wird üblicherweise im Ereignis `OnMouseDown` gestartet. Über den Parameter `e` vom Typ `MouseEventArgs` kann überprüft werden, welche Taste der Maus gedrückt wird, und somit können Sie selbst festlegen, dass beispielsweise nur mit der rechten Maustaste gezogen werden können soll. Für dieses Beispiel soll es egal sein, welche Maustaste verwendet wird.

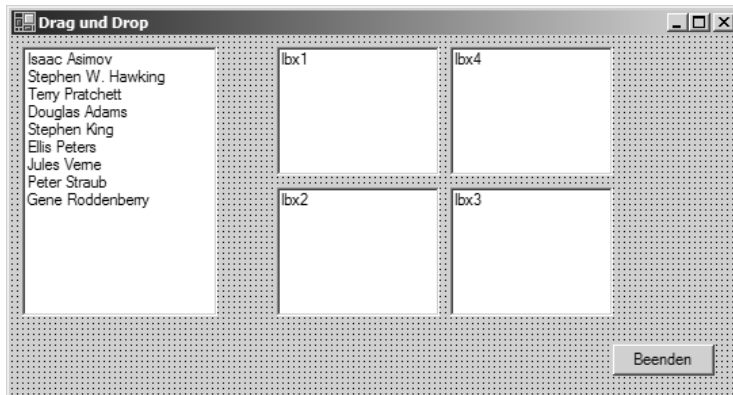


Abbildung 17.13: Der Aufbau des Drag&Drop-Beispiels [DragDropExample.sln]

Da die Elemente zwischen allen Listboxen hin- und hergeschoben werden sollen, müsste normalerweise für jede Listbox auch das Ereignis `MouseDown` belegt werden. Einfacher ist es jedoch, lediglich ein Ereignis zu belegen und die Listbox, von der aus gezogen wird, über den Parameter `Sender` zu ermitteln.

Sender

Weiterhin soll das Element ja aus der Listbox gelöscht werden, aus der es herausgezogen wird. Deshalb werden sowohl das Element als auch die Referenz auf die Listbox in einem Feld des Formulars, in diesem Fall in den Variablen `source` und `sourceIndex`, gespeichert.

Das Element, das verschoben werden soll, muss ebenfalls noch ermittelt werden. Offensichtlich wird nämlich die Eigenschaft `SelectedIndex` nicht upgedated, wenn eine Drag&Drop-Operation gestartet wird. Daher muss der korrekte Index über die Methode `IndexFromPoint()` der Listbox ermittelt werden. Hier die komplette Routine:

IndexFromPoint()

```
Private Sub lbxSource_MouseDown( _
    ByVal sender As Object, _
    ByVal e As System.Windows.Forms.MouseEventArgs) _
    Handles lbxSource.MouseDown, lbx1.MouseDown, _
            lbx2.MouseDown, lbx3.MouseDown, _
            lbx4.MouseDown

    ' --- Starten von Drag&Drop
    Dim aPoint As Point
    Dim lbx As ListBox
    Dim aIndex As Integer
```

```

lbox = CType(sender, ListBox)

' --- Zu ziehendes Item ermitteln
aPoint = New Point(e.X, e.Y)
aIndex = lbox.IndexFromPoint(aPoint)

If aIndex >= 0 Then
    source = lbox
    sourceIndex = aIndex
    lbox.DoDragDrop(lbox.Items(aIndex), _
        DragDropEffects.All)
End If

End Sub

```

Listing 17.17: Das Ereignis MouseDown [DragDropExample.sln]

Auf die gleiche Art und Weise werden auch die Ereignisse `DragEnter` und `DragDrop` implementiert, nämlich für alle Listboxen ein gemeinsames Ereignis. In `DragEnter` wird festgelegt, dass die gezogenen Daten dann akzeptiert werden sollen, wenn es sich um Text handelt. Festgelegt wird das über die Eigenschaft `Effects` des Parameters `DragEventArgs`.

```

Private Sub lbox1_DragEnter(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.DragEventArgs) _
    Handles lbox1.DragEnter, lbox2.DragEnter, _
        lbox3.DragEnter, lbox4.DragEnter

    If (e.Data.GetDataPresent(DataFormats.Text)) Then
        e.Effect = DragDropEffects.All
    Else
        e.Effect = DragDropEffects.None
    End If

End Sub

```

Listing 17.18: Das Ereignis DragEnter [DragDropExample.sln]

Wird das Element fallen gelassen, kann das ursprüngliche Element gelöscht und die Daten können der Listbox hinzugefügt werden.

```

Private Sub lbox1_DragDrop(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.DragEventArgs) _
    Handles lbox1.DragDrop, lbox2.DragDrop, _
        lbox3.DragDrop, lbox4.DragDrop

    Dim lbox As ListBox
    lbox = CType(sender, ListBox)

```

```

' --- Zuerst Quelle löschen
If Not source Is Nothing Then
    source.Items.RemoveAt(sourceIndex)
End If
ltx.Items.Add(e.Data.GetData(DataFormats.Text))

End Sub

```

Listing 17.19: Das Ereignis DragDrop [DragDropExample.sln]

Damit wäre das Programm fertig gestellt. Sie können die verschiedenen Namen aus den Listboxen nun wild verschieben. Das Ganze sieht zur Laufzeit dann aus wie in Abbildung 17.14.

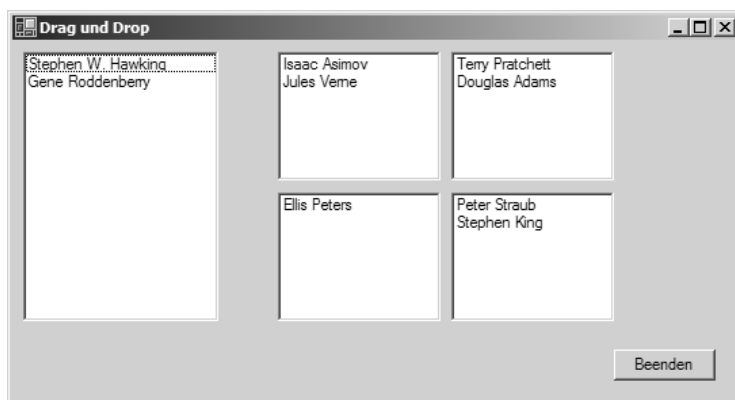


Abbildung 17.14: Das Programm zur Laufzeit [DragDropExample.sln]

Drag&Drop mit Steuerelementen

Das Verschieben von Daten ist eine Sache. Sicherlich haben Sie aber auch schon festgestellt, dass Sie z. B. in der Entwicklungsumgebung Komponenten beispielsweise von einem Panel in ein anderes verschieben können. Auch diese Art von Drag&Drop kann leicht nachgestellt werden.

Für dieses Beispiel sind auf dem Hauptformular drei Panels angeordnet, von denen eines einen Button beherbergt. Dieser soll nun wahlweise auf ein anderes Panel geschoben werden können.

Die Ereignisse sind die gleichen, nur dass diesmal mit Panels gearbeitet wird. Verschoben wird ein Button. Da dieser möglicherweise noch gedrückt werden können soll, wird festgelegt, dass nur mit der rechten Maustaste verschoben werden kann. Zum Starten des Drag&Drop-Vorgangs dient natürlich das MouseDown-

**Steuerelemente
verschieben**

Ereignis des Buttons. Das Ereignis `DragEnter` der Panels muss nicht weiter beachtet werden; darin wird lediglich festgelegt, dass jede Art von Daten akzeptiert werden soll.

```
Private Sub btnDrag_MouseDown( _
    ByVal sender As Object, _
    ByVal e As System.Windows.Forms.MouseEventArgs) _
    Handles btnDrag.MouseDown

    ' --- Ziehen nur mit rechter Taste
    If e.Button = MouseButton.Right Then
        btnDrag.DoDragDrop(btnDrag, DragDropEffects.All)
    End If
End Sub

Private Sub pnl1_DragEnter(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.DragEventArgs) _
    Handles pnl1.DragEnter, pnl2.DragEnter, _
        pnl3.DragEnter

    ' --- Soll immer valid sein
    e.Effect = DragDropEffects.All

End Sub
```

Listing 17.20: Die Ereignisse `MouseDown` und `DragEnter` [`DragDropButton.sln`]

Ablegen des Steuerelements

Beim Ablegen soll nun die aktuelle Mausposition ermittelt und die Position des Buttons entsprechend festgelegt werden. Um den Button zu verschieben, genügt es, ihn der Aufzählung `Controls` des Ziel-Panels hinzuzufügen. Die Mausposition jedoch muss umgerechnet werden. Dazu dient die Methode `PointToClient()` des Panels.

```
Private Sub pnl1_DragDrop(ByVal sender As Object, _
    ByVal e As System.Windows.Forms.DragEventArgs) _
    Handles pnl1.DragDrop, pnl2.DragDrop, _
        pnl3.DragDrop

    ' --- Deklarationen
    Dim pnl As Panel

    pnl = CType(sender, Panel)
    pnl.Controls.Add(btnDrag)
    btnDrag.Location = _
        pnl.PointToClient(pnl.MousePosition)

End Sub
```

Listing 17.21: Das Ereignis `DragDrop` [`DragDropButton.sln`]

Das Ergebnis zeigt Abbildung 17.15.



Abbildung 17.15: Das fertige Programm mit dem bereits gezogenen Button [DragDropButton.sln]

18 Der Namespace System.Drawing

Mit der Einführung des .NET Frameworks wurde auch ein neues Modell zum Erstellen von Grafiken implementiert, das das bisher genutzte GDI (*Graphics Device Interface*) ablöst. Diese verbesserte Version mit Namen GDI+ bietet einiges an Vorteilen, nicht zuletzt eine größere Performance und einfaches Handling. Auch die Fähigkeiten von GDI+ übertreffen die von GDI bei weitem.

**Graphics Device
Interface**

GDI+ unterstützt 2D-Zeichnungen, Grafiken (Bitmap und Metafile) und Text. Einige der implementierten Features sind:

Features von GDI+

- ▶ Unterstützung von Alpha-Blending
- ▶ Texturen und Farbverläufe
- ▶ Regions
- ▶ Gleitkomma-Koordinaten
- ▶ Unterstützung für Bitmaps, PNG-Grafiken, JPEG-Grafiken, TIF-Grafiken und viele mehr
- ▶ Bildbearbeitungsfunktionen
- ▶ Standardfunktionen wie Drehen, Beschneiden usw.
- ▶ ClearType-Unterstützung
- ▶ Unicode-Unterstützung
- ▶ Text mit Farbverläufen oder Texturen

Vom Leistungsumfang her ist GDI+ dem alten GDI also deutlich überlegen. Hinzu kommt, dass alle diese Funktionen sehr einfach zu benutzen sind. Leider ist es aber auch so, dass all diese Möglichkeiten so umfangreich sind, dass Sie in diesem Kapitel lediglich einen verhältnismäßig kleinen Einblick in das bekommen können, was GDI+ eigentlich ermöglicht. Es wird allerdings

genug sein, um das Arbeiten mit grafischen Funktionen zu beginnen und Sie werden sehen, dass auch die weiteren Features sehr leicht zu handhaben sind.



Der Namespace System.Drawing wird bei einem neuen Projekt standardmäßig eingebunden, in C# mittels using-Anweisung, in Visual Basic .NET projektweit über die Projekteinstellungen. Sie müssen diesen Namespace daher nicht mehr gesondert mit einbinden.

18.1 Die Klasse Graphics

- Zeichenoberfläche

Zum Zeichnen bzw. zum Erstellen einer Grafik benötigen Sie eine Oberfläche, auf der Sie die Grafik erstellen können. Das ist vergleichbar mit der Leinwand eines Malers. Im .NET Framework wird diese Zeichenfläche durch die Klasse Graphics zur Verfügung gestellt. Wenn Sie also eine Grafik erstellen, tun Sie dies immer auf einem Graphics-Objekt.
- CreateGraphics()

Die weiteren, für Grafiken relevanten Klassen des .NET Frameworks und auch einige der entsprechenden Steuerelemente beinhalten alle eine Methode CreateGraphics(), die ein solches Graphics-Objekt zurückliefert, auf dem Sie dann die Bitmap erstellen können. Graphics selbst besitzt keine Konstruktoren, kann daher also nicht wie andere Klassen instanziiert werden.
- shared-Methoden

Eine weitere Möglichkeit, ein Graphics-Objekt zu erhalten, sind die statischen (shared) Methoden dieser Klasse, die ebenfalls eine Instanz dieser Klasse zurückliefern. Tabelle 18.1 listet diese Methoden und ihre Bedeutung auf.

Methode	Funktion
FromHDC()	Erzeugt ein neues Graphics-Objekt aus einem Handle auf einen Gerätekontext
FromHwnd()	Erzeugt ein neues Graphics-Objekt von einem Handle auf ein anderes Fenster
FromImage()	Erzeugt ein neues Graphics-Objekt aus einem Image- oder einem Bitmap-Objekt

Tabelle 18.1: Statische Methoden der Klasse Graphics

Zum eigentlichen Erstellen einer Grafik dienen die zahlreichen Methoden von Graphics. Was und wie gezeichnet wird, hängt von weiteren Klassen ab, die z.B. einen Zeichenstift, verschiedene

Arten von Pinseln oder Füllmustern beinhalten. Tabelle 18.2 listet einige der Zeichenmethoden auf.

Methode	Funktion
DrawArc()	Zeichnet einen Kreisausschnitt. Die Methode ist mehrfach überladen, erster Parameter ist jedoch immer ein Pen-Objekt, mit dem gezeichnet wird.
Clear()	Löscht die Zeichenoberfläche
DrawBezier()	Zeichnet eine Bezier-Kurve mit dem vorgegebenen Zeichenstift
DrawEllipse()	Zeichnet eine Ellipse bzw. einen Kreis mit dem vorgegebenen Zeichenstift
DrawIcon()	Zeichnet ein Icon-Objekt in die Zeichenfläche. Wenn Sie verhindern wollen, dass das gezeichnete Icon evtl. skaliert wird, verwenden Sie DrawIconUnstretched().
DrawImage()	Zeichnet eine Image-Grafik. Wenn Sie verhindern wollen, dass das gezeichnete Bild skaliert wird, verwenden Sie DrawImageUnscaled().
DrawLine()	Zeichnet eine Linie mit dem vorgegebenen Zeichenstift
DrawLines()	Zeichnet mehrere durch Point-Objekte definierte Linien
DrawPath()	Zeichnet ein GraphicsPath-Objekt auf die Zeichenfläche. GraphicsPath ist eine Klasse, die eine Folge zusammenhängender Linien verkörpert. Mit dieser Klasse können Sie z.B. gerade Linien mit Kurven verbinden und so alle Formen von Grafikobjekten erstellen.
DrawPie()	Zeichnet ein Tortenstück
DrawPolygon()	Zeichnet ein Vieleck mit dem vorgegebenen Zeichenstift
DrawRectangle()	Zeichnet ein Rechteck mit dem vorgegebenen Zeichenstift
DrawRectangles()	Zeichnet mehrere Rechtecke, die durch Rectangle-Objekte angegeben sind
DrawString()	Zeichnet einen String, d.h. ermöglicht das Schreiben auf der Zeichenoberfläche. Verwendet werden ein Brush-Objekt oder ein Font-Objekt.
FillEllipse()	Zeichnet eine gefüllte Ellipse (oder einen gefüllten Kreis) mit dem vorgegebenen Brush-Objekt
FillPie()	Zeichnet ein gefülltes Tortenstück unter Verwendung des vorgegebenen Brush-Objekts

Tabelle 18.2: Instanzmethoden der Klasse Graphics

Methoden	Funktion
<code>FillPolygon()</code>	Zeichnet ein gefülltes Vieleck unter Verwendung des vorgegebenen <code>Brush</code> -Objekts
<code>FillRectangle()</code>	Zeichnet ein gefülltes Rechteck unter Verwendung des vorgegebenen <code>Brush</code> -Objekts
<code>Flush()</code>	Führt alle noch anstehenden Zeichenoperationen aus

Tabelle 18.2: Instanzmethoden der Klasse `Graphics` (Fortsetzung)

ClipBounds() Weitere Methoden, z.B. die Methode `ClipBounds()`, die ein `RectangleF`-Objekt zurückliefert, das den aktuellen Zeichenbereich repräsentiert, werden im Verlauf des Kapitels noch erklärt.



Alle Methoden der Klasse `Graphics` erwarten entweder ein Objekt vom Typ `Pen` oder ein Objekt vom Typ `Brush` zum Zeichnen. Das erste Argument aller Zeichenmethoden, die mit `Draw` beginnen, ist ein `Pen`-Objekt, das erste Argument aller Methoden, die mit `Fill` beginnen, ist ein `Brush`-Objekt. Die `Draw`-Methoden zeichnen Umrisse mit einem Zeichenstift, die `Fill`-Methoden sind für Füllungen bzw. Effekte zuständig.

18.2 Hilfsklassen

Der Namespace `System.Drawing` beinhaltet einige Hilfsklassen und Aufzählungen, die sich bei der Arbeit als sehr nützlich erweisen. Diese Klassen und Aufzählungen sollen hier kurz vorgestellt werden. Die Verwendung gestaltet sich in der Regel recht einfach.

- ▶ `ColorTranslator` ermöglicht die Umsetzung von Farbwerten von und zu GDI+-Farben. Die Klasse beinhaltet lediglich statische Methoden zur Umwandlung in Windows-, OLE- und HTML-Farben.
- ▶ `Brushes` enthält einige statische Methoden, die jeweils ein `SolidBrush`-Objekt zurückliefern. Ein `SolidBrush` wird benutzt, um eine einfarbige Füllung zu erzeugen.
- ▶ `Pens` enthält statische Methoden, die jeweils ein `Pen`-Objekt mit einer bestimmten Farbe und der Breite 1 zurückliefern.
- ▶ `SystemBrushes` enthält ebenfalls Methoden, die ein `SolidBrush`-Objekt erzeugen; enthalten sind hier aber nur die Systemfarben.

- ▶ `SystemColors` enthält ebenfalls statische Methoden, die jeweils ein `Color`-Objekt mit einer bestimmten Farbe zurückliefern.
- ▶ `SystemIcons` enthält statische Methoden, die `Icon`-Objekte zurückliefern. Enthalten sind einige Icons, die vom System verwendet werden.
- ▶ `SystemPens` enthält statische Methoden, die ein `Pen`-Objekt zurückliefern, das einer bestimmten Systemfarbe entspricht. Jedes dieser `Pen`-Objekte hat eine Breite von einem Pixel.
- ▶ `KnownColor` ist eine Aufzählung aller bekannten Farben des Systems, die z.B. dazu benutzt werden kann, eine Combobox mit Farben zu füllen.

Weiterhin sind noch einige Converter-Klassen enthalten, z.B. `FontConverter` oder `ColorConverter`, die dazu dienen, Farben, Zeichensätze usw. in andere Datentypen umzuwandeln.

Converter-Klassen

Die Verwendung dieser Hilfsklassen kann teilweise ein wenig Arbeit sparen. Wenn Sie z.B. ein Formular mit einer Farbe füllen wollen, können Sie so vorgehen:

```
Dim g As Graphics = Me.CreateGraphics()  
g.FillRectangle()(New SolidBrush(Color.Blue), g.ClipBounds())
```

oder Sie können sich einige Schreibarbeit sparen. Das folgende Fragment tut das Gleiche:

```
Dim g As Graphics = Me.CreateGraphics()  
g.FillRectangle()(brushes.Blue, g.ClipBounds())
```

Ebenso leicht können Sie beispielsweise eine Combobox oder eine Listbox mit den Farbbezeichnungen füllen, die das System kennt. Dazu wird die Aufzählung `KnownColors` verwendet:

```
Dim colors As New KnownColor()  
cbxColor1.Items.AddRange( _  
    System.Enum.GetNames(colors.GetType()))  
cbxColor2.Items.AddRange( _  
    System.Enum.GetNames(colors.GetType()))
```

Sicherlich werden Sie von Zeit zu Zeit noch weitere Verwendungsmöglichkeiten für diese Klassen finden.

18.3 Füllwerkzeuge

Brush Brush ist die Basisklasse für alle Füllobjekte. Brush selbst ist abstrakt, Sie müssen also ein Objekt verwenden, das eine reale Implementation dieser Klasse darstellt. Zu diesen gehören z.B. die Klassen SolidBrush, LinearGradientBrush oder PathGradientBrush. Abbildung 18.1 zeigt die Vererbungshierarchie der Füllklassen.

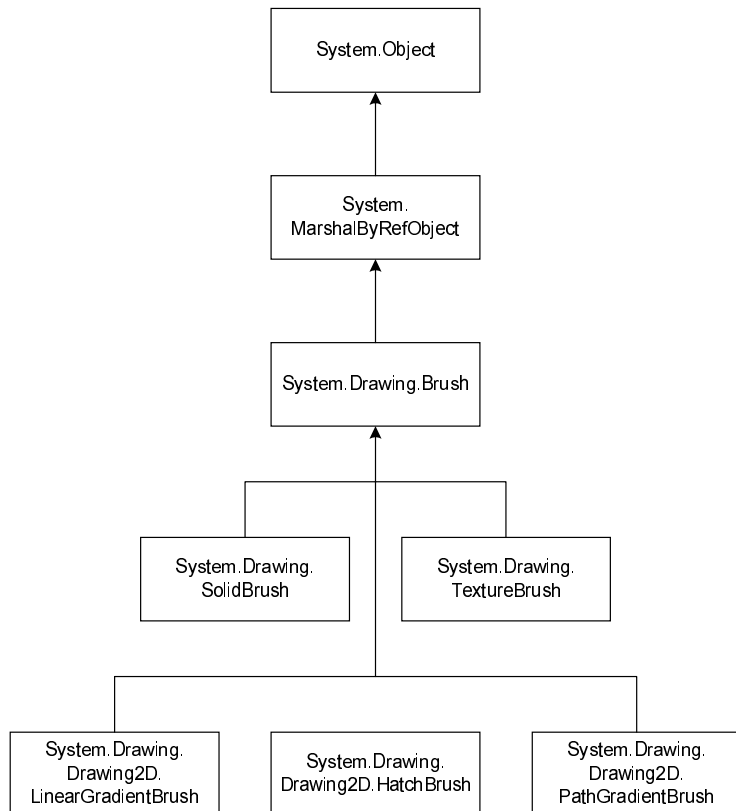


Abbildung 18.1: Vererbungshierarchie der Brush-Klassen

18.3.1 Die Klasse SolidBrush

SolidBrush SolidBrush ist ein Füllobjekt, mit dem Sie eine einfarbige Füllung erzeugen können. Es handelt sich dabei um das einfachste Brush-Objekt, enthalten ist lediglich eine Eigenschaft Color, mit der Sie die Farbe des Pinsels bestimmen können. Der Konstruktor der Klasse erwartet ebenfalls eine Farbe in Form eines Color-Objekts

als Parameter. Danach können Sie einfarbige Füllungen erzeugen. Gezeigt wurde im vorherigen Abschnitt bereits das Füllen des Hintergrunds eines Formulars mit Hilfe eines `SolidBrush`-Objekts. `SolidBrush` ist im Namespace `System.Drawing` deklariert.

18.3.2 Die Klasse `LinearGradientBrush`

Ein Gradient ist ein Farbverlauf. Die Klasse `LinearGradientBrush`, deklariert im Namespace `System.Drawing.Drawing2D`, ermöglicht das Zeichnen von Farbverläufen mit zwei oder mehr Farben, wobei auch einige Effekte mit eingebaut sind. Der Konstruktor erwartet zumindest die Angabe von zwei Farben und entweder die Angabe eines `Rectangle`-Objekts oder zwei Angaben in Form von `Point`-Objekten. Weitere überladene Versionen des Konstruktors ermöglichen auch noch die Angabe eines Winkels bzw. einer Orientierung des Farbverlaufs.

Die Orientierung kann aber auch festgelegt werden, indem Sie zwei entsprechende `Point`- (oder `PointF`)-Objekte angeben. Liegt der Startpunkt der Füllung beispielsweise in der linken oberen Ecke und der Endpunkt in der rechten unteren Ecke, haben Sie automatisch einen Winkel von 45° (bzw., je nachdem wie Sie es sehen, einen Winkel von 135°).

Farbverläufe

**Richtung des
Farbverlaufs**

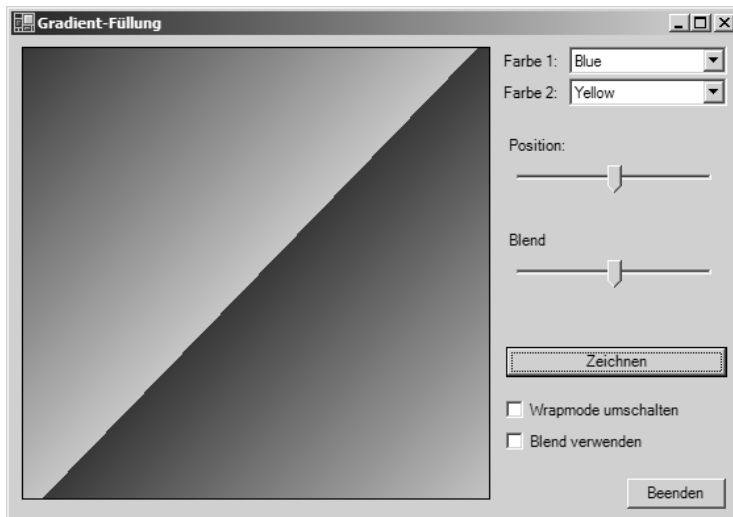


Abbildung 18.2: Eine einfache Farbverlaufsfüllung [`FillGradient.sln`]

Die Positionen geben den Anfangs- und Endpunkt des Pinsels an, müssen aber nicht dem Anfangs- und Endpunkt des Zeichenbereichs entsprechen. Falls der Zeichenbereich größer ist, wird die Füllung einfach wiederholt, wie Sie an Abbildung 18.2 sehen können.

In diesem Beispiel (der Quelltext folgt noch) wird über den POSITION-Regler angegeben, wo sich das Ende des Farbverlaufs bezüglich des Graphics-Objekts befinden soll. Gezeichnet wird auf einem Panel. Auf den Regler BLEND kommen wir gleich noch zu sprechen, auch hier ist eine interessante Möglichkeit eingebaut, die Sie auch aus diversen Zeichenprogrammen kennen.

WrapMode

LinearGradientBrush besitzt eine Eigenschaft `WrapMode`, mit der Sie festlegen können, ob die Farbverlaufsfüllung gespiegelt werden soll und in welche Richtung das geschehen soll. Damit kann ein interessanter Effekt erzielt werden, nämlich eine Art Neon-Linie. Abbildung 18.3 zeigt eine entsprechende Abbildung des Programms, in der dieser Effekt sehr stark sichtbar ist.

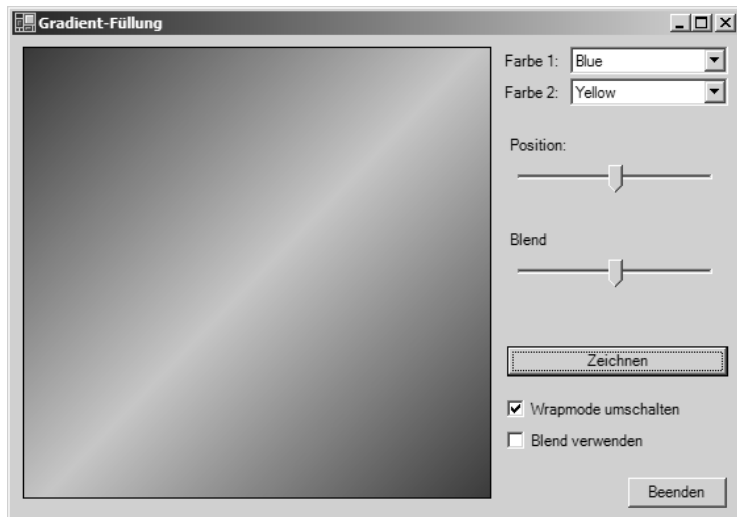


Abbildung 18.3: Farbverlauf gespiegelt [FillGradient.sln]

Die möglichen Zuweisungen an `WrapMode` sind `WrapMode.Tile`, `WrapMode.TileFlipX`, `WrapMode.TileFlipY` und `WrapMode.TileFlipXY`. Die ebenfalls mögliche Einstellung `WrapMode.TileFlipClamp` kann bei `LinearGradientBrush` nicht verwendet werden. Im Beispiel wurde `WrapMode.TileFlipXY` verwendet.

Eine weitere Möglichkeit ist das Verschieben des Mittelpunkts eines Farbverlaufs. Das wird über ein Objekt vom Typ `Blend` erreicht, das der gleichnamigen Eigenschaft von `LinearGradientBrush` zugewiesen wird.

`Blend` besitzt zwei Eigenschaften, die in diesem Fall von Interesse sind. `Positions` beinhaltet ein Array aus `Single`-Werten, mit dem angegeben wird, welche Entfernungen vom Rand für die jeweiligen Farben verwendet werden sollen. D.h., Sie setzen damit praktisch eine Markierung. `Factors` enthält ebenfalls eine prozentuale Einstellung, die angibt, wie viel Prozent einer Farbe an der entsprechenden Markierung verwendet werden soll.

Wenn Sie beispielsweise angeben, dass an der Position 50% (also genau in der Mitte des Farbverlaufs) ein Faktor von 20% verwendet werden soll, werden von Farbe 1 20% an dieser Stelle dargestellt, von Farbe 2 80%. Das Resultat ist eine Verschiebung des Mittelpunkts. Abbildung 18.4 und Abbildung 18.5 zeigen zwei verschiedene Einstellungen.

Blend

Positions und Factors

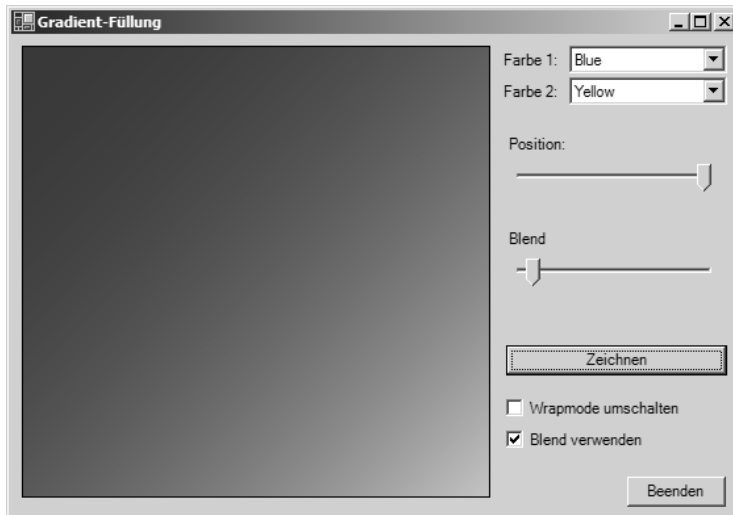


Abbildung 18.4: Die erste Einstellung des Blend-Werts [FillGradient.sln]

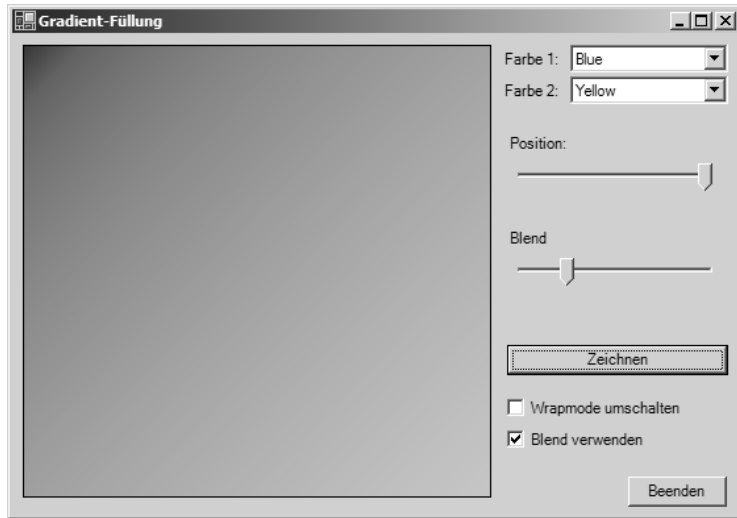


Abbildung 18.5: Die zweite Einstellung des Blend-Werts [FillGradient.sln]

Interpolation Colors

`LinearGradientBrush` ist nicht auf zwei Farben beschränkt. Im Beispiel wurden zwar nur zwei Farben verwendet, die in diesem Fall direkt im Konstruktor angegeben wurden, über die Eigenschaft `InterpolationColors` (vom Typ `ColorBlend`) können Sie aber weitere Farben und die entsprechenden Positionen bzw. Faktoren für diese Farben festlegen. Auch hier gilt, dass das Programmieren die größten Lernerfolge bringt, experimentieren Sie also ein wenig damit herum.

LinearGradientBrush verwenden

Den Quelltext, der für das obige Beispiel verwendet wurde, sehen Sie in Listing 18.1. Wie angesprochen handelt es sich um ein recht einfaches Beispiel, bei dem lediglich zwei Farben verwendet werden und ein wenig mit `Blend` und `WrapMode` experimentiert wird.

```
'=====
' FillGradient.sln
' Copyright 2002 Frank Eller
' Zeichnen eines Farbverlaufs
'=====

Imports System.Drawing
Imports System.Drawing.Drawing2D

Public Class Form1
    Inherits System.Windows.Forms.Form
```

```
#Region " Windows Form Designer generated code "
```

*'Der Code, der von VS.NET eingefügt wurde, wird
'aus Übersichtsgründen nicht dargestellt.*

```
#End Region
```

```
Private Sub Form1_Load( _  
    ByVal sender As System.Object, _  
    ByVal e As System.EventArgs) Handles MyBase.Load
```

```
    ' --- Farben festlegen  
    Dim colors As New KnownColor()  
    cbxColor1.Items.AddRange( _  
        System.Enum.GetNames(colors.GetType()))  
    cbxColor2.Items.AddRange( _  
        System.Enum.GetNames(colors.GetType()))
```

```
End Sub
```

```
Private Sub btnDraw_Click( _  
    ByVal sender As System.Object, _  
    ByVal e As System.EventArgs) _  
    Handles btnDraw.Click
```

```
    ' --- Deklarationen  
    Dim aWidth As Integer  
    Dim aHeight As Integer  
    Dim aCanvas As Graphics = _  
        drawPanel.CreateGraphics()  
    Dim aPos1 As Integer  
    Dim aPos2 As Integer  
    Dim aBrush As LinearGradientBrush  
    Dim aBlend As Blend
```

```
    ' --- Blend festlegen  
    aBlend = New Blend(1)  
    aBlend.Positions(0) = 0.5  
    aBlend.Factors(0) = _  
        10.0 / Convert.ToSingle(trbBlend.Value)
```

```
    ' --- Weitere Einstellungen festlegen  
    aWidth = drawPanel.Width  
    aHeight = drawPanel.Height  
    aPos1 = (aWidth / 100) * trbPosition.Value  
    aPos2 = (aHeight / 100) * trbPosition.Value  
    If aPos1 > aWidth Then aPos1 = aWidth  
    If aPos2 > aHeight Then aPos2 = aHeight
```

```
    ' --- Den Pinsel erzeugen  
    aBrush = New LinearGradientBrush(New Point(0, 0), _  
        New Point(aPos1, aPos2), _
```

```

        Color.FromName(cbxColor1.SelectedItem), _
        Color.FromName(cbxColor2.SelectedItem))

' --- Wrapmode kontrollieren (Checkbox)
If cbxFlip.Checked Then
    aBrush.WrapMode = WrapMode.TileFlipX
Else
    aBrush.WrapMode = WrapMode.Tile
End If

' --- Blend kontrollieren (Checkbox)
If cbxUseBlend.Checked Then aBrush.Blend = aBlend

' --- Zeichnen
aCanvas.FillRectangle()( _
    aBrush, 0, 0, drawPanel.Width, _
    drawPanel.Height)

End Sub

Private Sub btnExit_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnExit.Click

    Me.Close()
End Sub
End Class

```

Listing 18.1: Der Quelltext zum Beispiel [FillGradient.sln]

18.3.3 Die Klasse PathGradientBrush

PathGradient Brush

`PathGradientBrush` ermöglicht es, ein Objekt einer beliebigen Form mit einem Farbverlauf zu füllen. Die Form wird durch ein Objekt vom Typ `GraphicsPath` oder durch ein Array aus `Point`-Objekten dargestellt. `GraphicsPath` selbst kann mehrere Formen beinhalten, die das Aussehen des Gesamtobjekts bestimmen. Damit gibt es mehrere Möglichkeiten, einen `PathGradientBrush` zu verwenden.

CenterColor und SurroundColors

Der Farbverlauf wird immer von der Mitte aus gezeichnet. Die Startfarbe wird durch die Eigenschaft `CenterColor` festgelegt, die weiteren Farben über die Eigenschaft `SurroundColors`. Diese Farben korrespondieren allerdings mit den einzelnen hinzugefügten Formen, d.h. sie haben nicht die gleiche Funktion wie die Eigenschaft `InterpolationColors` bei `LinearGradientBrush`. Diese Eigenschaft gibt es bei `PathGradientBrush` ebenfalls.

Da der zentrale Punkt nicht zwangsläufig in der Mitte des Objekts liegen muss, kann er über die Eigenschaft `CenterPoint` festgelegt werden. Es handelt sich dabei lediglich um den Punkt, von dem die Farbverlaufsfüllung ausgeht und dessen Farbe durch `CenterColor` festgelegt ist.

PathGradientBrush verwenden

Zum Zeichnen eines Rechtecks, das mit einer Farbverlaufsfüllung belegt ist, werden die Eckpunkte dieses Rechtecks einfach dem Konstruktor des `PathGradientBrush`-Objekts übergeben. Der Aufbau des Beispielprojekts entspricht in etwa dem des `LinearGradientBrush`-Beispiels, auch hier wird die Grafik wieder auf einem Panel-Steurelement angezeigt. Das Zeichnen eines Rechtecks mit Farbverlauf zeigt Listing 18.2.

```
Private Sub btnDrawRect_Click( _  
    ByVal sender As System.Object, _  
    ByVal e As System.EventArgs) _  
    Handles btnDrawRect.Click  
  
    ' --- Zeichnet ein Rechteck mit PathGradientBrush  
    ' --- Deklarationen  
    Dim aCanvas As Graphics = _  
        pnlDraw.CreateGraphics()  
    Dim pointA As New Point(0, 0)  
    Dim pointB As New Point(pnlDraw.Width, 0)  
    Dim pointC As New Point(pnlDraw.Width, _  
        pnlDraw.Height)  
    Dim pointD As New Point(0, pnlDraw.Height)  
    Dim surroundColor As Color = _  
        Color.FromName(cbxColor2.SelectedItem)  
    Dim middleColor As Color = _  
        Color.FromName(cbxColor1.SelectedItem)  
    Dim aBrush As PathGradientBrush  
    Dim sColors() As Color = {surroundColor}  
  
    ' --- Array mit Koordinaten festlegen  
    Dim brushCoords() As Point = _  
        {pointA, pointB, pointC, pointD}  
  
    ' --- PathGradientBrush erzeugen  
    aBrush = New PathGradientBrush(brushCoords)  
  
    ' --- Farben festlegen  
    aBrush.CenterColor = middleColor  
    aBrush.SurroundColors = sColors
```

```

' --- Zeichnen
aCanvas.FillRectangle()(aBrush, 0, 0, _
    pnlDraw.Width, pnlDraw.Height)
End Sub

```

Listing 18.2: Zeichnen eines Rechtecks mit Farbverlaufs­füllung von innen heraus [PathGradient.sln]

Die Position des Mittelpunkts muss in diesem Fall nicht festgelegt werden, sie liegt standardmäßig in der Mitte des PathGradient Brush-Objekts. Auf dem Bildschirm sieht die Grafik dann aus wie in Abbildung 18.6 gezeigt.

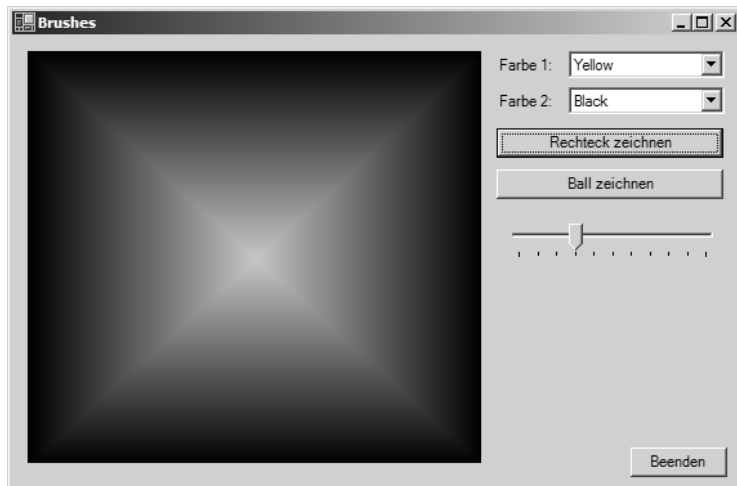


Abbildung 18.6: Ein Rechteck mit Farbverlauf [PathGradient.sln]

PathGradient Brush mit GraphicsPath

Ein weiteres interessantes Beispiel ist ein Beleuchtungseffekt, der ebenfalls über einen PathGradientBrush erzeugt werden kann. Beispielsweise sieht eine Kugel (oder ein Ball) realistischer aus, wenn man sieht, auf welchen Punkt das Licht scheint. Dieser Punkt muss dann der hellste sein, die Helligkeit nach außen hin abnehmen. Auch das lässt sich leicht über einen PathGradientBrush lösen, wobei in diesem Fall ein GraphicsPath-Objekt verwendet wird, das einen Kreis enthält. Listing 18.3 zeigt den Quelltext der Funktion.

```

Private Sub btnDrawBall_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnDrawBall.Click

' --- Deklarationen

```

```

Dim aPath As GraphicsPath
Dim aCanvas As Graphics = _
    pnlDraw.CreateGraphics()
Dim surroundColor As Color = _
    Color.FromName(cbxColor2.SelectedItem)
Dim middleColor As Color = _
    Color.FromName(cbxColor1.SelectedItem)
Dim aBrush As PathGradientBrush
Dim sColors() As Color = {surroundColor}
Dim aRect As Rectangle
Dim posValue As Integer = trbPosition.Value

' --- Das GraphicsPath-Objekt festlegen
aPath = New GraphicsPath()
aRect = New Rectangle(30, 30, _
    pnlDraw.Width - 60, pnlDraw.Height - 60)
aPath.AddEllipse(aRect)

' --- Den PathGradientBrush festlegen
aBrush = New PathGradientBrush(aPath)
aBrush.CenterPoint = _
    New PointF((aRect.Left + aRect.Right) / _
    posValue, (aRect.Top + aRect.Bottom) / _
    posValue)

' --- Die Farben festlegen
aBrush.CenterColor = middleColor
aBrush.SurroundColors = sColors

' --- Zeichnen
aCanvas.FillRectangle()(Brushes.Black, 0, 0, _
    pnlDraw.Width, pnlDraw.Height)
aCanvas.FillRectangle()(aBrush, aRect)
End Sub

```

Listing 18.3: Zeichnen eines Balls bzw. einer Kugel mit einer Farbverlaufs­füllung [PathGradient.sln]

Für dieses Beispiel wird die Position des Mittelpunkts über eine Trackbar-Komponente festgelegt. Die beste Einstellung ist die im Beispiel voreingestellte dritte Einstellung. Einen Screenshot des Programms sehen Sie in Abbildung 18.7.

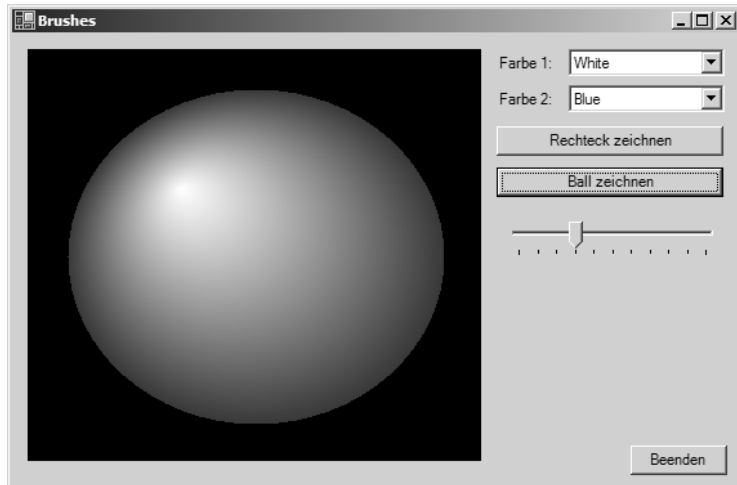


Abbildung 18.7: Ein (halbwegs realistischer) Ball [PathGradient.sln]

18.3.4 Die Klasse TextureBrush

`TextureBrush` ist ein Pinselobjekt, das dazu dient, sich wiederholende Grafiken neben- und übereinander zu zeichnen. Solche Grafiken, auch Texturen genannt, sind aus dem Internet bekannt. Idealerweise passen sie an den Kanten so aneinander, dass es im Endergebnis wie eine einzige Fläche aussieht. `TextureBrush` ist im Namespace `System.Drawing` deklariert.

`TextureBrush` besitzt einige Möglichkeiten. Um jedoch eine einfache Textur zu erzeugen, genügt es, das gewünschte Bild im Konstruktor anzugeben und die gewünschte Zeichenfläche mit `FillRectangle()` unter Verwendung des `Brushes` zu füllen. Ein Beispiel wäre das Füllen eines Formulars mit einer Textur. Der eigentliche Code besteht aus drei Zeilen:

```
Private Sub FillForm(ByVal aImage As Image)
    Dim tBrush As New TextureBrush(aImage)
    Dim aCanvas As Graphics = Me.CreateGraphics()

    aCanvas.FillRectangle()(tBrush, _
        aCanvas.ClipBounds())
End Sub
```

`Image` als abstrakte Klasse kann alle unterstützten Bildformate aufnehmen. Damit können Sie also eine Bitmap, eine *jpeg*-Datei oder sogar ein `Icon` als Hintergrund benutzen. Abbildung 18.8 zeigt das Programm zur Laufzeit.

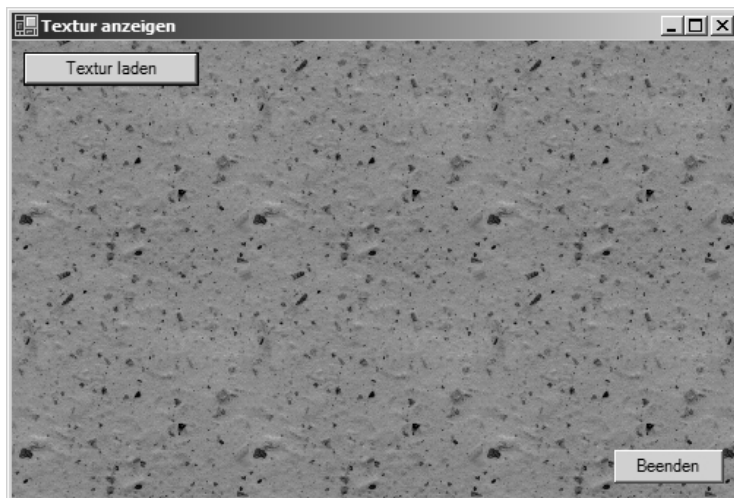


Abbildung 18.8: Ein Formular mit Hintergrundtextur [Textures.sln]

18.4 Bitmaps

Bisher wurden nur Füllungen besprochen, jetzt geht es darum, Linien, Rechtecke oder Kreise zu zeichnen. Wie bereits angemerkt dienen alle Methoden der Klasse `Graphics`, die mit `Draw` beginnen, dem Zeichnen von Linien oder Kreisen. Der Zeichenstift wird durch die Klasse `Pen` festgelegt.

18.4.1 Die Klasse `Pen`

Ein Objekt des Typs `Pen` repräsentiert einen Zeichenstift mit all seinen Eigenschaften, also mit einem Linientyp, einer Farbe, einer Breite und sogar einer eventuellen Endmarkierung, also einem Pfeil oder einem Punkt. Anders als bei den `Brush`-Objekten genügt aber ein `Pen`-Objekt. Wollen Sie einem solchen Zeichenstift beispielsweise einen Farbverlauf zuweisen, übergeben Sie im Konstruktor einfach einen `LinearGradientBrush` und können dann damit zeichnen.

Über die Eigenschaft `DashStyle` können Sie das Aussehen des `Pen`-Objekts festlegen. Damit können Sie Linien zeichnen, die gestrichelt, gepunktet oder (wie in der Standardeinstellung festgelegt) einfach nur durchgezogen sind. Die einstellbaren Werte sind in

Pen

DashStyle

der Aufzählung `DashStyle` angegeben, die im Namensraum `System.Drawing.Drawing2D` deklariert ist. Die möglichen Werte zeigt Tabelle 18.3.

Wert	Bedeutung
Custom	Legt ein benutzerdefiniertes Muster auf den Zeichenstift
Dash	Ermöglicht das Zeichnen einer gestrichelten Linie
DashDot	Ermöglicht das Zeichnen einer Linie, die abwechselnd aus einem Strich und einem Punkt besteht
DashDotDot	Ermöglicht das Zeichnen einer Linie, die aus je einem Strich und zwei Punkten besteht
Dot	Ermöglicht das Zeichnen einer gepunkteten Linie
Solid	Ermöglicht das Zeichnen einer durchgezogenen Linie. Diese Einstellung ist der Standard.

Tabelle 18.3: Werte der Aufzählung `DashStyle`

Pen erzeugen Erzeugt wird ein Zeichenstift entweder, indem dem Konstruktor eine Farbe übergeben wird, oder aber aus einem `Brush`-Objekt, womit der Zeichenstift dann die Eigenschaften dieses `Brush`-Objekts annimmt. Wahlweise kann im Konstruktor auch gleich eine Breite angegeben werden.

18.4.2 Einfache Pens verwenden

In einem kleinen Beispiel soll gezeigt werden, wie man mit Hilfe eines `Pen`-Objekts Linien bzw. auch Kreise zeichnen kann. Die Farbe des Zeichenstifts kann angegeben werden, ebenso Anfangs- und Endpunkt bzw. die Art der Linie, die verwendet werden soll. Um das Ganze besser sichtbar zu machen, wurde als Breite grundsätzlich der Wert 5 verwendet.

Color.FromName() Um die Combobox für die Farben nicht allzu umfangreich werden zu lassen, wurden diesmal nur ein paar Farben eingestellt. Es können Linien und Kreise erstellt werden, wobei für beide jeweils ein beliebiger `DashStyle`-Wert angegeben werden kann. Die Auswahl der Farben gestaltet sich noch recht einfach. In den Comboboxen für die Farben wurden Namen verwendet, die dem System bekannt sind, wodurch die Farbe über `Color.FromName()` ermittelt werden kann.

Der Quelltext ist relativ leicht verständlich, so dass er hier komplett abgedruckt werden kann. Sie sehen ihn in Listing 18.4.

```
'=====
' SimpleDrawingExample.sln
' Copyright 2002 Frank Eller
' Kreise und Linien zeichnen
'=====

Imports System.Drawing.Drawing2D

Public Class Form1
    Inherits System.Windows.Forms.Form

    #Region " Windows Form Designer generated code "

        'Der Code, der von VS.NET eingefügt wurde, wird
        'aus Übersichtsgründen nicht dargestellt.

    #End Region

    Private lineColor As Color = Color.Black
    Private circleColor As Color = Color.Black
    Private lineStyle As Integer = 0
    Private circleStyle As Integer = 0

    Private Sub LineStyleIndexChanged( _
        ByVal sender As Object, _
        ByVal e As System.EventArgs) _
        Handles cbxLineStyle.SelectedIndexChanged

        Select Case cbxLineStyle.SelectedIndex
            Case 0
                lineStyle = DashStyle.Dash
            Case 1
                lineStyle = DashStyle.Dot
            Case 2
                lineStyle = DashStyle.DashDot
            Case 3
                lineStyle = DashStyle.DashDotDot
            Case 4
                lineStyle = DashStyle.Solid
        End Select
    End Sub

    Private Sub CircleStyleChanged( _
        ByVal sender As Object, _
        ByVal e As System.EventArgs) _
        Handles cbxLineStyle2.SelectedIndexChanged

        Select Case cbxLineStyle2.SelectedIndex
            Case 0
                circleStyle = DashStyle.Dash
            Case 1
```

```

        circleStyle = DashStyle.Dot
    Case 2
        circleStyle = DashStyle.DashDot
    Case 3
        circleStyle = DashStyle.DashDotDot
    Case 4
        circleStyle = DashStyle.Solid
    End Select
End Sub

Private Sub LineColorChanged( _
    ByVal sender As Object, _
    ByVal e As System.EventArgs) _
    Handles cbxColor.SelectedIndexChanged

    lineColor = Color.FromName(cbxColor.SelectedItem)
End Sub

Private Sub CircleColorChanged( _
    ByVal sender As Object, _
    ByVal e As System.EventArgs) _
    Handles cbxColor2.SelectedIndexChanged

    circleColor = _
        Color.FromName(cbxColor2.SelectedItem)
End Sub

Private Sub btnExit_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnExit.Click

    Me.Close()
End Sub

Private Sub btnClear_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnClear.Click

    picDraw.CreateGraphics().Clear(Color.White)
End Sub

Private Sub btnDrawLine_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnDrawLine.Click

    ' --- Deklarationen
    Dim x1 = Convert.ToInt32(tbxX1.Text)
    Dim x2 = Convert.ToInt32(tbxX2.Text)

```

```

Dim y1 = Convert.ToInt32(tbxY1.Text)
Dim y2 = Convert.ToInt32(tbxY2.Text)
Dim aCanvas As Graphics = _
    picDraw.CreateGraphics()
Dim aPen As New Pen(lineColor, 5)

' --- Linienart festlegen
aPen.DashStyle = lineStyle
aPen.Color = lineColor

' --- Zeichnen
aCanvas.DrawLine(aPen, x1, y1, x2, y2)
End Sub

Private Sub btnDrawCircle_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnDrawCircle.Click

' --- Deklarationen
Dim x = Convert.ToInt32(tbxX3.Text)
Dim y = Convert.ToInt32(tbxY3.Text)
Dim r = Convert.ToInt32(tbxRadius.Text)
Dim aPen As New Pen(circleColor, 5)
Dim aCanvas As Graphics = _
    picDraw.CreateGraphics()

' --- Berechnen des Rechtecks zum Zeichnen
Dim rx As Integer = x - r
Dim ry As Integer = y - r
Dim width As Integer = r * 2
Dim height As Integer = r * 2

' --- Linienart festlegen
aPen.DashStyle = circleStyle
aPen.Color = circleColor

' --- Zeichnen
aCanvas.DrawEllipse(aPen, rx, ry, width, height)
End Sub

```

Listing 18.4: Einfache Zeichenoperationen [SimpleDrawingExample.sln]

Das Ergebnis dieses Programms sehen Sie in Abbildung 18.9.

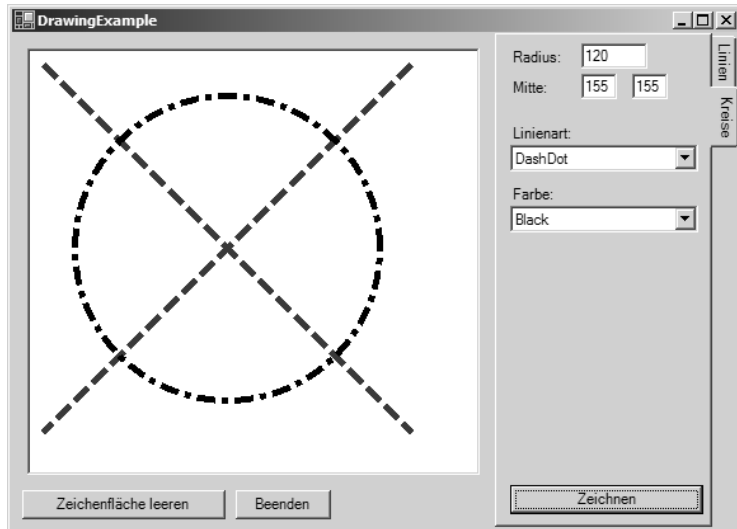


Abbildung 18.9: Kreise und Linien [SimpleDrawingExample.sln]

18.4.3 Zeichenstift mit Farbverlauf

Die Klasse `Pen` besitzt auch Konstruktoren, die ein `Brush`-Objekt als Parameter erwarten. Damit können Zeichenstifte erzeugt werden, bei denen der Zeichenstift einem `Brush`-Objekt entspricht. Auch mit dieser Möglichkeit können interessante Effekte erzielt werden.

Im Beispiel wird ein `LinearGradientBrush`-Objekt verwendet, um das Aussehen des Zeichenstifts festzulegen. Der Farbverlauf geht über die gesamte Zeichenfläche, wobei hier wieder ein Panel verwendet wird.

Wenn nun mit einem solchen `Pen`-Objekt gezeichnet wird, wird immer die Farbe gezeichnet, die sich an der gezeichneten Stelle auch am `Brush`-Objekt befinden würde. D.h. wenn eine Linie gezeichnet wird, sieht man den Farbverlauf. Wird ein Rechteck gezeichnet, sieht man in den Rändern ebenfalls den Farbverlauf. Es ist also nicht so, dass dieser Farbverlauf immer nur für die gerade gezeichnete Linie gilt, sondern vielmehr für den gesamten Hintergrund. Das `Pen`-Objekt legt praktisch die Farbe nur frei, wenn es an dieser Stelle zeichnet. Deutlich wird das anhand zweier Screenshots. Abbildung 18.10 zeigt das Beispielprogramm mit einer gezeichneten Linie, Abbildung 18.11 mit einem Rechteck.

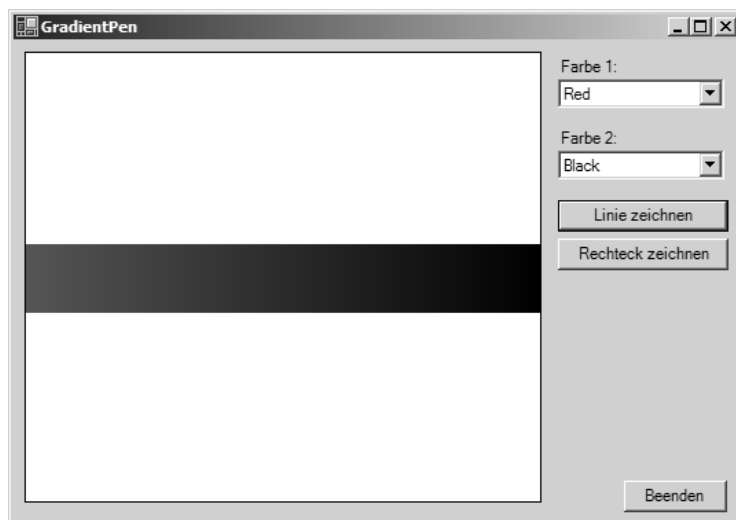


Abbildung 18.10: Eine Linie mit einem Gradient-Pen-Objekt [GradientPen.sln]

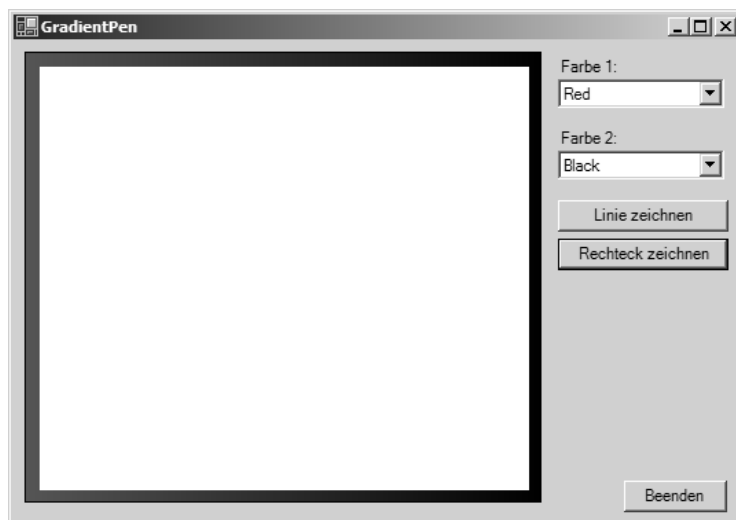


Abbildung 18.11: Ein Rechteck mit dem gleichen Gradient-Pen-Objekt [GradientPen.sln]

Der Quellcode des Beispielprogramms ist wiederum nicht besonders schwierig zu verstehen. Sie sehen ihn in Listing 18.5.

```

'=====
' GradientPen.sln
' Copyright 2002 Frank Eller
' Linien mit Farbverlauf zeichnen
'=====

Imports System.Drawing.Drawing2D

Public Class Form1
    Inherits System.Windows.Forms.Form

    #Region " Windows Form Designer generated code "

        'Der Code, der von VS.NET eingefügt wurde, wird
        'aus Übersichtsgründen nicht dargestellt.

    #End Region

    Private Sub Draw(ByVal lineOrRect As Boolean)

        ' --- Deklarationen
        Dim color1 As Color = Color.Black
        Dim color2 As Color = Color.White
        Dim aWidth As Integer = pnlDraw.Width
        Dim aheight As Integer = pnlDraw.Height
        Dim aCanvas As Graphics = _
            pnlDraw.CreateGraphics()

        ' --- Farben festlegen
        If cbxColor1.SelectedIndex > -1 Then
            color1 = Color.FromName(cbxColor1.SelectedItem)
        End If
        If cbxColor2.SelectedIndex > -1 Then
            color2 = Color.FromName(cbxColor2.SelectedItem)
        End If

        ' --- GradientBrush festlegen
        Dim aBrush = New LinearGradientBrush( _
            New Point(0, 0), _
            New Point(aWidth, 0), color1, color2)

        ' --- Pen festlegen
        Dim gPen As New Pen(CType(aBrush, Brush))

        ' --- Zeichnen
        aCanvas.Clear(Color.White)

        If lineOrRect Then
            gPen.Width = 50
            aCanvas.DrawLine(gPen, 0, CInt(aheight / 2), _
                aWidth, CInt(aheight / 2))
        End If
    End Sub
End Class

```



```
Else
    gPen.Width = 10
    aCanvas.DrawLine(gPen, 0, 5, aWidth, 5)
    aCanvas.DrawLine(gPen, 0, aheight - 5, _
        aWidth, aheight - 5)
    aCanvas.DrawLine(gPen, 5, 5, 5, aheight - 5)
    aCanvas.DrawLine(gPen, aWidth - 5, 5, _
        aWidth - 5, aheight - 5)
End If

End Sub

Private Sub btnDraw_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnDraw.Click

    Draw(True)
End Sub

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

    Draw(False)
End Sub

Private Sub btnExit_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnExit.Click

    Me.Close()
End Sub

End Class
```

Listing 18.5: Der Quellcode des Beispielprogramms [GradientPen.sln]

Natürlich können auch andere `Brush`-Objekte verwendet werden. Dies soll lediglich als Beispiel dienen.

18.5 Arbeiten mit Grafiken

18.5.1 Die Klasse Image

**Image, Bitmap
und MetaFile**

Basisklasse für das Arbeiten mit Bitmaps oder Metadateien ist die Klasse `Image`, bei der es sich um eine abstrakte Klasse handelt. Reale Implementationen dieser Klasse sind `Bitmap` und `MetaFile`. Eine reale Implementation kann aber auch über die statischen Methoden `FromFile()`, `FromHBitmap()` oder `FromStream()` von `Image` erzeugt werden.

`Image` selbst enthält einige Eigenschaften, die wichtige Informationen über ein enthaltenes Bild zurückliefern. Diese Eigenschaften sehen Sie in Tabelle 18.4.

Eigenschaft	Bedeutung
Height	Liefert die Höhe der enthaltenen Grafik zurück
HorizontalResolution	Liefert die horizontale Auflösung (Pixel pro Zoll, dpi) der enthaltenen Grafik zurück
Palette	Liefert die Paletteninformationen der enthaltenen Grafik zurück. Sie können auch eine neue Palette zuweisen.
PhysicalDimension	Liefert die realen Ausmaße der Datei zurück (Höhe und Breite)
PixelFormat	Liefert das <code>PixelFormat</code> zurück. Das <code>PixelFormat</code> gibt an, wie viele Farben die Grafik enthält.
Size	Liefert die Höhe und Breite der Grafik in Pixel zurück
VerticalResolution	Liefert die vertikale Auflösung der enthaltenen Grafik zurück
Width	Liefert die Breite der enthaltenen Grafik zurück

Tabelle 18.4: Die Eigenschaften der Klasse `Image`

Bildinformationen ermitteln

Ein kleines Beispiel benutzt die verschiedenen Eigenschaften, um detailliertere Informationen über eine Grafik zurückzuliefern. Die Grafik wird in ein `Image`-Objekt geladen, dann skaliert und auf ein Panel gezeichnet. Die Eigenschaften des `Image`-Objekts werden ausgelesen und angezeigt. Listing 18.6 zeigt den Quellcode des Programms.

PixelFormat

Obwohl `Image` in `System.Drawing` deklariert ist, sind einige der Klassen, die die Detailinformationen enthalten, in einem anderen Namespace, nämlich `System.Drawing.Imaging`, deklariert. Dieser

muss also eingebunden werden. Im Beispiel handelt es sich dabei lediglich um die Klasse `PixelFormat`. `PixelFormat` liefert Informationen über die Anzahl der Farben, also darüber, wie viele Bits pro Pixel verwendet wurden.

```
'=====
' GradientPen.sln
' Copyright 2002 Frank Eller
' Bitmap anzeigen
'=====

Imports System.Drawing.Imaging

Public Class Form1
    Inherits System.Windows.Forms.Form

    #Region " Windows Form Designer generated code "

    'Der Code, der von VS.NET eingefügt wurde, wird
    'aus Übersichtsgründen nicht dargestellt.

    #End Region

    Private Sub btnLoad_Click( _
        ByVal sender As System.Object, _
        ByVal e As System.EventArgs) _
        Handles btnLoad.Click

        Dim aImage As Image
        Dim aCanvas As Graphics

        If dlgOpenBitmap.ShowDialog() Then

            ' --- Hier Bitmap laden
            aImage = _
                Image.FromFile()(dlgOpenBitmap.FileName)

            ' --- Bild skalieren
            Dim drawWidth As Integer = pnlDraw.Width
            Dim drawHeight As Integer = pnlDraw.Height
            Dim scaleFactor As Single

            If aImage.Width > aImage.Height Then
                scaleFactor = (drawWidth * 100.0F) / _
                    aImage.Width
                drawHeight = Convert.ToInt32( _
                    ((aImage.Height * scaleFactor) / 100))
            Else
                scaleFactor = (drawHeight * 100.0F) / _
                    aImage.Height
```

```

        drawWidth = Convert.ToInt32( _
            ((aImage.Width * ScaleFactor) / 100))
    End If

    ' --- Auf Panel zeichnen
    aCanvas = pnlDraw.CreateGraphics()
    aCanvas.DrawImage()(aImage, 0, 0, _
        drawWidth, drawHeight)

    ' --- Jetzt die Infos
    Dim pf As PixelFormat

    With lblInfo.Items
        .Clear()
        .Add("Breite: " + aImage.Width.ToString())
        .Add("Höhe: " + aImage.Height.ToString())
        .Add("HRes: " + _
            aImage.HorizontalResolution.ToString() + _
            " dpi")
        .Add("VRes: " + _
            aImage.VerticalResolution.ToString() + _
            " dpi")
        .Add("PixelFormat: " + System.Enum.GetName( _
            pf.GetType(), aImage.PixelFormat))
        .Add("RAW-Format: " + _
            aImage.RawFormat.ToString())
    End With
End If

End Sub

Private Sub btnExit_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnExit.Click

    Me.Close()
End Sub
End Class

```

Listing 18.6: Ermittlung von Informationen über eine Grafik [SimpleBitmapExample.sln]

Skalierung Die Skalierung vor dem Übertragen der Bitmap auf das Panel ist notwendig, weil das Bild unter Umständen zu groß für den verfügbaren Bereich sein könnte. Die Verwendung des Steuerelements `PictureBox` ist ebenfalls nicht ideal, weil deren Eigenschaft `Stretch` es zwar ebenfalls ermöglicht, das Bild in die Zeichenfläche einzupassen, allerdings dabei die Proportionen nicht beibehält. Die manuelle Skalierung ist daher erforderlich. Für das Zeichnen der Grafik auf das Panel ist die Methode `DrawImage()` verantwortlich, ebenfalls eine Methode der Klasse `Graphics`.

Einen Screenshot des Programms mit einem beliebigen Bild zeigt Abbildung 18.12.

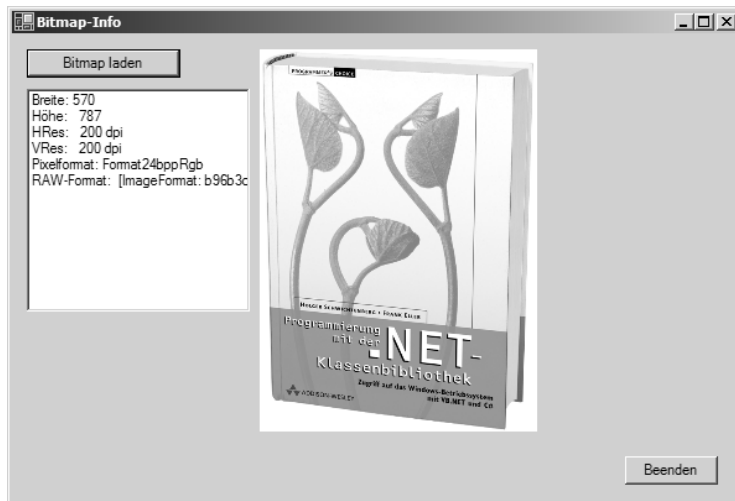


Abbildung 18.12: Informationen über eine Grafik [SimpleBitmapExample.sln]

Erstellen von Thumbnails

Sicherlich kennen Sie die kleinen Vorschaubilder, die es oft im Internet zu sehen gibt. Diese Bilder zeigen üblicherweise das Bild in Originalgröße, wenn man darauf klickt. Die Klasse `Image` besitzt eine Methode `GetThumbnailImage()`, die ein solches verkleinertes Bild zurückliefert. Dabei handelt es sich ebenfalls um ein `Image`-Objekt. Als Parameter wird die gewünschte Größe der Thumbnail-Bilder angegeben. In diesem Beispiel wird die Methode `GetThumbnailImage()` verwendet, um eine verkleinerte Version eines Bildes zusätzlich zum Originalbild anzuzeigen.

`GetThumbnailImage()` erwartet vier Parameter. Die ersten beiden geben die Größe des Bildes an, das generiert werden soll. Der dritte Parameter ist ein Delegate auf eine Abbruchmethode, der aber in GDI+ Version 1 nicht benutzt wird. An dieser Stelle wird zwar von der Online-Hilfe angegeben, dass Sie einen solchen Delegate erstellen und ihn dann übergeben müssten, dem ist aber nicht so – mit `Nothing` (bzw. `null` in C#) funktioniert es ebenso. Als letzten Parameter müssen Sie `IntPtr.Zero` übergeben.

GetThumbnailImage()

Das Beispielprogramm besteht aus zwei Panel-Komponenten, auf denen die Grafiken dargestellt werden. Dazu kommt noch ein Button zum Laden und einer zum Beenden. Die Bezeichnungen, die verwendet wurden, sollten eindeutig sein. Das gesamte Programm zeigt Listing 18.7.

```
'=====
' Thumbnail.sln
' Copyright 2002 Frank Eller
' Bitmap und Thumbnail anzeigen
'=====

Public Class Form1
    Inherits System.Windows.Forms.Form

    #Region " Windows Form Designer generated code "

        'Der Code, der von VS.NET eingefügt wurde, wird
        'aus Übersichtsgründen nicht dargestellt.

    #End Region

    Private Sub ShowThumbnail(ByVal aImage As Image)
        Dim aThumb As Image
        Dim drawWidth As Integer = 100
        Dim drawHeight As Integer = 100
        Dim scaleFactor As Single
        Dim aCanvas As Graphics
        Dim drawPos As New Point()

        If aImage.Width > aImage.Height Then
            scaleFactor = 10000.0F / aImage.Width
            drawHeight = Convert.ToInt32( _
                ((aImage.Height * scaleFactor) / 100))
        Else
            scaleFactor = (10000.0F) / aImage.Height
            drawWidth = Convert.ToInt32( _
                ((aImage.Width * scaleFactor) / 100))
        End If

        aThumb = aImage.GetThumbnailImage( _
            drawHeight, drawWidth, Nothing, IntPtr.Zero)

        ' --- Zeichnen in der Mitte des Bildes
        drawPos.X = (pn1Thumbnail.Width - drawWidth) / 2
        drawPos.Y = (pn1Thumbnail.Height - drawHeight) / 2

        aCanvas = pn1Thumbnail.CreateGraphics()
        aCanvas.DrawImage()(aThumb, drawPos.X, _
            drawPos.Y, drawWidth, drawHeight)
```

End Sub

Private Sub LoadImage(ByVal filename As String)

 ' --- Lädt das Bild

 Dim aImage As Image

 Dim aCanvas As Graphics

 ' --- Hier Bitmap laden

 aImage = Image.FromFile()(filename)

 ' --- Bild skalieren

 Dim drawWidth As Integer = pnlImage.Width

 Dim drawHeight As Integer = pnlImage.Height

 Dim ScaleFactor As Single

 If aImage.Width > aImage.Height Then

 ScaleFactor = (drawWidth * 100.0F) / _
 aImage.Width

 drawHeight = Convert.ToInt32(_
 ((aImage.Height * ScaleFactor) / 100))

 Else

 ScaleFactor = (drawHeight * 100.0F) / _
 aImage.Height

 drawWidth = Convert.ToInt32(_
 ((aImage.Width * ScaleFactor) / 100))

 End If

 ' --- Auf Panel zeichnen

 aCanvas = pnlImage.CreateGraphics()

 aCanvas.DrawImage()(aImage, 0, 0, drawWidth, drawHeight)

 ' --- Thumbnail anzeigen

 ShowThumbNail(aImage)

End Sub

Private Sub btnLoad_Click(_

 ByVal sender As System.Object, _

 ByVal e As System.EventArgs) _

Handles btnLoad.Click

 If dlgOpenImage.ShowDialog = DialogResult.OK Then

 LoadImage(dlgOpenImage.FileName)

 End If

End Sub

Private Sub btnExit_Click(_

 ByVal sender As System.Object, _

 ByVal e As System.EventArgs) _

```

Handles btnExit.Click

Me.Close()
End Sub
End Class

```

Listing 18.7: Anzeigen eines Thumbnails [Thumbnail.sln]

In der Methode `ShowThumbnail()` wurde zusätzlich noch darauf geachtet, dass das verkleinerte Bild in der Mitte der entsprechenden Panel-Komponente angezeigt wird. Das Programm zur Laufzeit zeigt Abbildung 18.13.

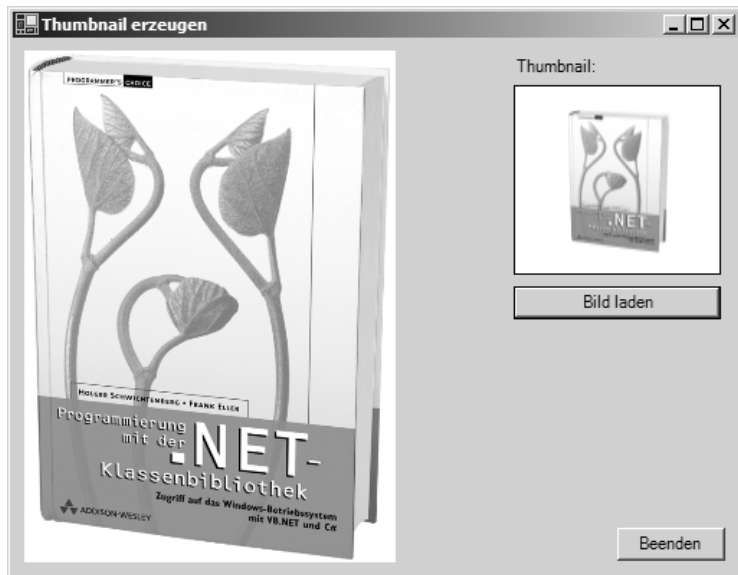


Abbildung 18.13: Eine Grafik mit Thumbnail [Thumbnail.sln]

18.5.2 Die Klasse Bitmap

Wie Sie bereits gesehen haben, ist es durchaus möglich, mit bereits vorhandenen Bildern zu arbeiten, indem Sie diese einfach in ein `Image`-Objekt laden. Die Methode `FromFile()` liefert eine komplette Implementation der eigentlich abstrakten Klasse `Image` zurück.

Anders sieht es aus, wenn Grafiken erzeugt werden sollen, d.h. wenn eine `Bitmap` erzeugt werden soll. In diesem Fall wird eine reale Implementation einer Grafik benötigt, die leer und anpassbar ist. Von `Image` selbst können Sie kein Objekt erzeugen, hier benutzen Sie eine der Klassen `Bitmap` oder `MetaFile`.

Bitmap steht, wie der Name bereits sagt, für pixelorientierte Grafiken, MetaFile für Vektorgrafiken. Da beide von Image abgeleitet sind, implementieren sie die gleichen Methoden, die auch Image bereitstellt, d.h. Sie können auch hiermit Thumbnails erstellen oder auch eine Grafik auf eine Zeichenfläche bzw. ein Graphics-Objekt zeichnen. Bitmap enthält aber auch weitere Methoden, z.B. zum Zugriff auf einzelne Pixel (und deren Farben), die Eigenschaften entsprechen denen von Image. Eine Übersicht über einige der zusätzlichen Methoden liefert Tabelle 18.5.

Bitmap und MetaFile

Methode	Funktion
GetPixel()	Liefert die Farbe eines bestimmten Pixels der Bitmap zurück
LockBits()	Sperrt die Bitmap im Speicher
MakeTransparent()	Macht eine Farbe der Bitmap zur transparenten Farbe. Dabei kann es sich um die Standard-Transparenzfarbe der Bitmap handeln oder aber um eine frei gewählte Farbe.
SetPixel()	Legt die Farbe für ein bestimmtes Pixel der Bitmap fest
SetResolution()	Legt die Auflösung der Bitmap fest
UnlockBits()	Entsperrt die Bitmap im Speicher

Tabelle 18.5: Zusätzliche Methoden der Klasse Bitmap

Zum Zeichnen auf eine Bitmap benötigen Sie, wie bei den anderen Zeichenfunktionen auch, ein Graphics-Objekt. Die Klasse Bitmap liefert keine Methode, mit der ein entsprechendes Objekt erzeugt werden könnte. Graphics selbst besitzt aber eine statische (shared) Methode FromImage(), die Sie verwenden können. Zeichenoperationen auf dieses Graphics-Objekt spiegeln sich sofort in der Bitmap wider.

Bitmap erzeugen

Das folgende Beispiel erzeugt eine Bitmap und speichert sie auf der Festplatte ab. Um dem Benutzer zu zeigen, dass wirklich eine Grafik erzeugt wurde, wird diese in verkleinerter Form auch noch in ein Panel geladen und angezeigt.

Es geht hierbei um Grafiken aus der Chaos-Theorie, um so genannte *Fraktale*. Wer an dieser Stelle an das berühmte Apfelmännchen denkt, liegt nicht ganz falsch. Es soll hier jedoch um ganz andere Fraktale gehen, nämlich solche, die ein Wachstum darstellen. Das Programm kann dabei sowohl für klassische Fraktale als auch für Bäume oder Sträucher verwendet werden.

Fraktale

Produktionsregeln Das Wachstum wird festgelegt durch Produktionsregeln, die aus verschiedenen Zeichen bestehen. Der Einfachheit halber werden hier nur simple Produktionsregeln verwendet. Dabei bedeuten

- ▶ F: Zeichnen einer Linie
- ▶ -: Drehung nach links
- ▶ +: Drehung nach rechts
- ▶ [: Speichern der Position und weitergehen
- ▶]: Abrufen der zuletzt gespeicherten Position

Turtle-Grafik Die Grafik, die nach einer festgelegten Produktionsregel gezeichnet wird, wird hinterher in einer Bitmap abgespeichert. Es wird in diesem Fall auch direkt auf die Bitmap gezeichnet. Solche Grafiken werden auch als Turtle-Grafiken bezeichnet, weil die Grafik den Weg einer Schildkröte nachzeichnet, die einem vorgegebenen Weg folgt. Das hier vorgestellte Programm heißt dementsprechend auch *TurtlePainter*.

Für die Richtungswerte wird eine Struktur verwendet, die die Winkel beinhaltet. Den Aufbau dieser Struktur zeigt Listing 18.8.

```
Public Structure TurtleAngle

    Private _angleLeft As Double
    Private _angleRight As Double

    Public Property AngleRight() As Double
        Get
            Return _angleRight
        End Get
        Set(ByVal Value As Double)
            _angleRight = Value
        End Set
    End Property

    Public Property AngleLeft() As Double
        Get
            Return _angleLeft
        End Get
        Set(ByVal Value As Double)
            _angleLeft = Value
        End Set
    End Property

End Structure
```

Listing 18.8: Die Struktur für die Winkelangaben [TurtlePainter.sln]

Die Methoden zum Zeichnen werden komplett in einer eigenen Klasse mit Namen `TPaint` untergebracht. Diese Klasse benötigt einige Felder, die die Daten für die zu erstellende Zeichnung beinhalten.

```
Public Class TPaint

    #Region "Fields"

        Private angleRight As Double = 0.0
        Private angleLeft As Double = 0.0
        Private startValue As Integer = 10
        Private iterations As Integer = 6
        Private formula As String = String.Empty
        Private aFileName As String = String.Empty

        Private dx As Double = 0.0
        Private dy As Double = 0.0

        Private turtlePos As Point = New Point(0, 0)

        Private turtleDirection As New TurtleAngle()

        Private xs As New Stack()
        Private ys As New Stack()
        Private dxs As New Stack()
        Private dys As New Stack()

        Private aCanvas As Graphics
        Private turtlePen As Pen
        Private bgPen As Pen

    #End Region
```

Die Variablen `xs`, `ys`, `dxs` und `dys` speichern jeweils sowohl die x- und y-Koordinate der aktuellen Position als auch die Richtungswerte (`dxs` und `dys`). `dx` und `dy` beinhalten die aktuellen Richtungswerte. Die Variable `turtlePos` beinhaltet die aktuelle Position der Schildkröte. `turtlePen` ist der Zeichenstift, `bgPen` beinhaltet die Hintergrundfarbe. Die übrigen Variablen sollten sich durch ihren Namen erklären.

Die Winkel werden üblicherweise im Gradmaß angegeben, womit die Winkelfunktionen im .NET Framework aber nichts anfangen können. Deshalb müssen die Winkel ins Bogenmaß umgerechnet werden. Die Umrechnung ist allerdings nicht weiter schwierig.

```

Private Sub ConvertAngle(ByVal aR As Double, _
    ByVal aL As Double)

    Me.turtleDirection.AngleLeft = _
        ((aL / 180) * Math.PI)
    Me.turtleDirection.AngleRight = _
        ((aR / 180) * Math.PI)
End Sub

```

Links- und Rechtsdrehung der Schildkröte werden ebenfalls durch eine Berechnung durchgeführt. An dieser Stelle ist etwas Mathematik gefragt, Sie können die Formeln aber auch einfach so übernehmen, wie sie hier stehen.

```

Private Sub TurnRight()

    Dim dx1 As Double = dx
    dx = ((Math.Cos(Me.turtleDirection.AngleRight)) _
        * dx1) - _
        ((Math.Sin(Me.turtleDirection.AngleRight)) _
        * dy)
    dy = ((Math.Sin(Me.turtleDirection.AngleRight)) _
        * dx1) + _
        ((Math.Cos(Me.turtleDirection.AngleRight)) _
        * dy)
End Sub

Private Sub TurnLeft()

    Dim dx1 As Double = dx
    dx = ((Math.Cos(Me.turtleDirection.AngleLeft)) _
        * dx1) + _
        ((Math.Sin(Me.turtleDirection.AngleRight)) _
        * dy)
    dy = (- (Math.Sin(Me.turtleDirection.AngleRight)) _
        * dx1) + _
        ((Math.Cos(Me.turtleDirection.AngleLeft)) _
        * dy)
End Sub

```

Speichern und Abrufen der Positionen im Falle einer Verzweigung werden durch Stack-Objekte gelöst. `Push()` legt einen Wert im Stack ab, `Pop()` holt ihn wieder heraus. Da ein Stack ein *First-In-Last-Out*-Speicher ist müssen wir uns keine Gedanken über die korrekte Position der Werte machen, da immer alle Werte gespeichert bzw. herausgeholt werden.

Die Multiplikation mit dem Wert 0,7 dient einer »natürlicheren« Darstellung. Da es sich in diesem Fall häufig um das Wachstum von Bäumen oder Sträuchern handelt, wird durch die Verkleine-

rung der zu zeichnenden Linien (denn nichts anderes geschieht durch diese Multiplikation) ein Wachstum des Strauchs oder Baums dargestellt.

```
Private Sub StorePos()
    xs.Push(Me.turtlePos.X)
    ys.Push(Me.turtlePos.Y)
    dxs.Push(dx)
    dys.Push(dy)
    dx *= 0.7
    dy *= 0.7
End Sub
```

```
Private Sub RecallPos()
    turtlePos.X = xs.Pop()
    turtlePos.Y = ys.Pop()
    dx = dxs.Pop()
    dy = dys.Pop()
End Sub
```

Das Zeichnen einer Linie gestaltet sich ebenfalls recht einfach. Die Koordinaten sind bekannt, da sie errechnet werden können. Die eigentliche Zeichenmethode (für eine Linie) entspricht daher genau einer Codezeile:

```
Private Sub DrawLine(ByVal x1 As Integer, _
    ByVal y1 As Integer, ByVal x2 As Integer, _
    ByVal y2 As Integer)

    aCanvas.DrawLine(turtlePen, x1, y1, x2, y2)
End Sub
```

Kommen wir jetzt zu einer der interessanteren Methoden. Die Methode `DrawTurtle()` zeichnet rekursiv alle Linien, die durch die Formel festgelegt sind. Die Variable `stage` repräsentiert die aktuelle Iteration.

```
Private Sub DrawTurtle(ByVal stage As Integer)
    Dim i As Integer

    If stage < 0 Then
        DrawLine(turtlePos.X, turtlePos.Y, _
            turtlePos.X + Convert.ToInt32(dx), _
            turtlePos.Y + Convert.ToInt32(dy))
        turtlePos.X += Convert.ToInt32(dx)
        turtlePos.Y += Convert.ToInt32(dy)
        Exit Sub
    Else
        stage = stage - 1
        For i = 0 To formula.Length - 1
```

```

        Select Case formula.Chars(i)
            Case "F" : DrawTurtle(stage)
            Case "-" : TurnLeft()
            Case "+" : TurnRight()
            Case "[" : StorePos()
            Case "]" : RecallPos()
        End Select
    Next
    stage = stage + 1
End If
End Sub

```

Bisher hatte das Ganze noch nicht viel mit einer Bitmap zu tun, eigentlich mehr mit der Implementation der einzelnen Hilfsroutinen für das Zeichnen der Grafik. Die einzige Methode, die (außer dem Konstruktor) in dieser Klasse aufgerufen werden muss, ist die Methode `Go()`, die die Hauptarbeit verrichtet. Hier wird die Bitmap erzeugt und aus dieser Bitmap ein `Graphics`-Objekt (`aCanvas`), das dann zum Zeichnen verwendet werden kann. In diesem Fall wird eine Bitmap mit den Ausmaßen 1024x768 Bildpunkte verwendet, mit 100 dpi Auflösung. Damit ist die erzeugte Bitmap auch passend für ein Wallpaper.

```

Public Sub Go(ByVal Iter As Integer, _
    ByVal a1 As Double, ByVal a2 As Double, _
    ByVal size As Integer, ByVal formul As String)

    ' --- Bitmap erzeugen
    Dim aBitmap As New Bitmap(1024, 768, _
        Drawing.Imaging.PixelFormat.Format24bppRgb)
    aBitmap.SetResolution(100, 100)

    ' --- Graphics-Objekt erzeugen
    aCanvas = Graphics.FromImage()(aBitmap)

    Me.ConvertAngle(a1, a2)

    Me.formula = formul
    Me.turtlePos.X = aBitmap.Width / 2
    Me.turtlePos.Y = aBitmap.Height - 10
    Me.dx = 0
    Me.dy = -size * (Iter + 1)

    ' --- Zeichenfläche leeren
    aCanvas.Clear(bgPen.Color)

    ' --- Zeichnen
    Me.DrawTurtle(Iter)

```

```

' --- Speichern
aBitmap.Save(aFileName)
End Sub

```

Der Konstruktor der Klasse erzeugt die Pen-Objekte und legt den Dateinamen zum Speichern fest:

```

Public Sub New(ByVal fColor As Color, _
    ByVal bColor As Color, ByVal fn As String)

    Me.bgPen = New Pen(bColor)
    Me.turtlePen = New Pen(fColor)
    Me.aFileName = fn
End Sub

```

Es fehlt noch der Quellcode für das Hauptformular. In diesem wird wieder die bekannte Methode zum Darstellen eines Thumbnails verwendet, mit der die Grafik angezeigt wird. Listing 18.9 zeigt den Quellcode des Hauptformulars.

```

'=====
' TurtlePainter.sln
' Copyright 2002 Frank Eller
' Zeichnet eine Grafik und speichert sie
'=====

Public Class Form1
    Inherits System.Windows.Forms.Form

    #Region " Windows Form Designer generated code "

    'Der Code, der von VS.NET eingefügt wurde, wird
    'aus Übersichtsgründen nicht dargestellt.

    #End Region

    Private Sub btnBackground_Click( _
        ByVal sender As System.Object, _
        ByVal e As System.EventArgs) _
        Handles btnBackground.Click

        If dlgColor.ShowDialog = DialogResult.OK Then
            pnlBackground.BackColor = dlgColor.Color
        End If
    End Sub

    Private Sub btnForeground_Click( _
        ByVal sender As System.Object, _
        ByVal e As System.EventArgs) _
        Handles btnForeground.Click

```

```

    If dlgColor.ShowDialog = DialogResult.OK Then
        pnlForeground.BackColor = dlgColor.Color
    End If
End Sub

Private Sub txtStartSize_KeyPress( _
    ByVal sender As Object, ByVal e As _
    System.Windows.Forms.KeyPressEventArgs) _
    Handles txtStartSize.KeyPress, _
    txtAngleLeft.KeyPress, _
    txtAngleRight.KeyPress, _
    txtIterations.KeyPress

    If Not Char.IsDigit(e.KeyChar) Then
        e.Handled = True
    End If
End Sub

Private Sub txtFormula_KeyPress( _
    ByVal sender As Object, _
    ByVal e As _
    System.Windows.Forms.KeyPressEventArgs) _
    Handles txtFormula.KeyPress

    If e.KeyChar = "f" Then
        e.Handled = True
        txtFormula.SelectedText = "F"
    End If
    If e.KeyChar <> "+" And e.KeyChar <> "-" _
        And e.KeyChar <> "[" And e.KeyChar <> "]" _
        And e.KeyChar <> vbBack Then

        e.Handled = True
    End If
End Sub

Private Sub ShowThumbnail(ByVal aImage As Image)
    Dim aThumb As Image
    Dim drawWidth As Integer = 100
    Dim drawHeight As Integer = 100
    Dim scaleFactor As Single
    Dim aCanvas As Graphics
    Dim drawPos As New Point()

    If aImage.Width > aImage.Height Then
        scaleFactor = 22000.0F / aImage.Width
        drawHeight = Convert.ToInt32( _
            ((aImage.Height * scaleFactor) / 100))
    Else
        scaleFactor = (22000.0F) / aImage.Height
        drawWidth = Convert.ToInt32( _

```



```

        ((aImage.Width * ScaleFactor) / 100))
    End If

    aThumb = aImage.GetThumbnailImage( _
        drawHeight, drawWidth, Nothing, IntPtr.Zero)

    ' --- Zeichnen in der Mitte des Bildes
    drawPos.X = (pn1Thumbnail.Width - drawWidth) / 2
    drawPos.Y = (pn1Thumbnail.Height - drawHeight) / 2

    aCanvas = pn1Thumbnail.CreateGraphics()
    aCanvas.DrawImage()(aThumb, drawPos.X, _
        drawPos.Y, drawWidth, drawHeight)
End Sub

Private Sub btnDrawGraphic_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnDrawGraphic.Click

    ' --- Deklarationen
    Dim aFileName As String
    Dim aImage As Image

    If dlgSaveBitmap.ShowDialog = _
        DialogResult.OK Then

        aFileName = dlgSaveBitmap.FileName
        Dim a1 As Double = _
            Convert.ToDouble(Me.txtAngleRight.Text)
        Dim a2 As Double = _
            Convert.ToDouble(Me.txtAngleLeft.Text)

        Dim iter As Integer = _
            Convert.ToInt32(Me.txtIterations.Text)
        Dim size As Integer = _
            Convert.ToInt32(Me.txtStartSize.Text)
        Dim formul As String = Me.txtFormula.Text

        Dim tp As New TPaint( _
            pn1Foreground.BackColor, _
            pn1Background.BackColor, aFileName)

        tp.Go(iter, a1, a2, size, formul)

        ' --- Bild in Panel laden
        aImage = Image.FromFile()(aFileName)
        ShowThumbnail(aImage)
    End If
End Sub

```

```

Private Sub btnExit_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnExit.Click

    Me.Close()
End Sub
End Class

```

Listing 18.9: Der Quellcode des Hauptformulars [TurtlePainter.sln]

Einen Screenshot des Programms nach der Erstellung einer Grafik sehen Sie in Abbildung 18.14.

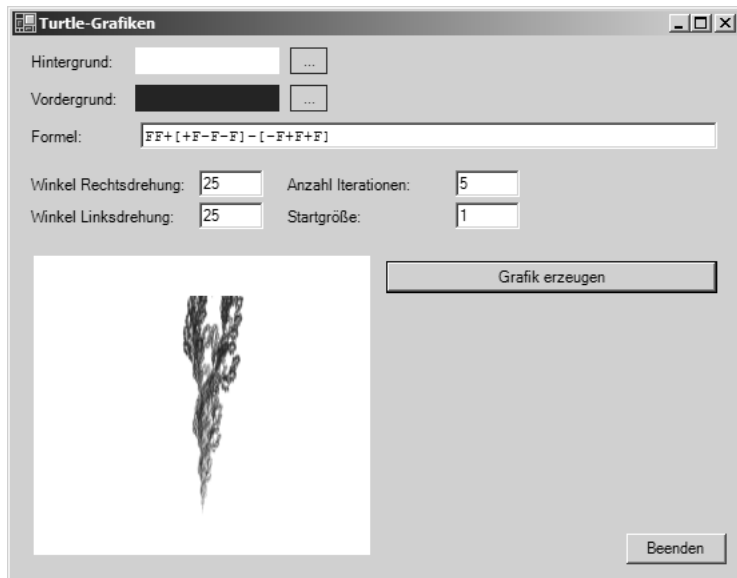


Abbildung 18.14: Ein Fraktal in Miniaturansicht [TurtlePainter.sln]

19 Namespace

System.Security.Cryptography

Das .NET-Framework beinhaltet auch eingebaute Algorithmen zum Verschlüsseln bzw. Entschlüsseln von Daten. Dieses Gebiet – nämlich die Kryptografie – ist sehr umfangreich. Verschiedenste Algorithmen sind bekannt, mit denen Nachrichten verschlüsselt bzw. entschlüsselt werden können. Eines der bekanntesten Tools dafür ist wohl *PGP* (Pretty Good Privacy), bei dem ein öffentlicher und ein privater Schlüssel gemeinsam verwendet werden, um die Sicherheit übertragener Nachrichten zu gewährleisten.

Im Namespace `System.Security.Cryptography` tummeln sich mehrere Klassen, die verschiedene Verschlüsselungsalgorithmen implementieren. Unter anderem können Sie folgende Algorithmen verwenden:

**Verschlüsselungs-
algorithmen**

- ▶ Rijndael
- ▶ DES
- ▶ TripleDES
- ▶ RC2
- ▶ RSA

Der eine oder andere wird bekannt sein. Effektiv wird eine Verschlüsselung aber meist erst durch die Verbindung verschiedener Algorithmen zu einem Gesamtprodukt, z.B. die Verbindung eines symmetrischen Algorithmus mit einem asymmetrischen. Worum es sich dabei handelt, wird im Weiteren noch genauer beschrieben werden.

19.1 Verschlüsselungstechniken

Alice und Bob Bei der Erklärung von Verschlüsselungstechniken werden meist zwei Personen herangezogen – Alice und Bob. Alice möchte Bob eine Nachricht schicken, wobei sie sicherstellen will, dass kein dritter diese Nachricht liest. Da beide in der Regel über das Internet verbunden sind, also über eine Verbindung die nicht gerade als sicher zu betrachten ist, müssen sie sich etwas ausdenken, um die Nachricht wirklich sicher zu machen.

Sichergestellt werden muss in diesem Zusammenhang nicht nur, dass die Nachricht verschlüsselt ist – es muss auch sichergestellt werden, dass die Nachricht auch wirklich von der Person kommt, die als Absender angegeben ist. Und natürlich, dass niemand, der die Nachricht möglicherweise abgefangen, dann verändert und mit der Signatur des ursprünglichen Absenders weitergeleitet hat.

19.1.1 Private Schlüssel

Symmetrische Verschlüsselung

Um dies zu gewährleisten, gibt es mehrere Möglichkeiten. Die erste Möglichkeit ist, die Nachricht mit einem so genannten privaten Schlüssel zu versehen. Dabei wird der gleiche Schlüssel sowohl zum Verschlüsseln einer Nachricht als auch zum Entschlüsseln verwendet. Ein solches Verfahren nennt man deshalb auch *symmetrisches Verschlüsselungsverfahren*. Natürlich muss auch das Verschlüsselungsverfahren das gleiche sein, d.h. Absender und Empfänger müssen beide wissen, mit welchem Verfahren sie die Nachricht unlesbar machen.

Um das Verfahren effektiv zu machen, wird die Nachricht zunächst in Blocks zerlegt. Die Größe dieser Blocks variiert, je nach Verschlüsselungsverfahren. Jeder dieser Blocks wird dann unter Verwendung des Schlüssels umgewandelt. Die resultierenden Textblöcke können beim Empfänger, der dazu den gleichen Schlüssel besitzen muss, wieder in lesbaren Text umgewandelt werden.

Da diese Methode den Nachteil hat, dass gleiche Textpassagen auch gleiche verschlüsselte Passagen ergeben, wird zusätzlich der zuletzt verschlüsselte Block in die Verschlüsselung des darauf folgenden Blocks mit einbezogen. Somit ist es für den abhörenden Spion nicht mehr möglich, anhand gleicher Passagen unter Umständen doch noch den Schlüssel zu finden.

Der wirkliche Nachteil besteht darin, dass beide Parteien, Sender und Empfänger, den gleichen Schlüssel benutzen müssen. Außerdem muss natürlich auch bekannt sein, welches Verschlüsselungsverfahren verwendet wurde, denn ein anderes Verfahren würde ja nicht die gleichen Ergebnisse bringen – trotz gleichen Schlüssels.

Nachteil

19.1.2 Öffentliche Schlüssel

Bei Verschlüsselungsmethoden mit einem öffentlichen Schlüssel (*Public-Key-Verfahren*) gibt es eigentlich zwei Schlüssel – einen öffentlichen, auf den jeder frei zugreifen kann, und einen privaten, der zusammen mit dem öffentlichen Schlüssel erzeugt wurde und so mit diesem in Zusammenhang steht. Diese Verschlüsselungsverfahren werden auch als asymmetrische Verschlüsselungsverfahren bezeichnet.

**Asymmetrische
Verschlüsselung**

Wenn nun eine Nachricht mit dem privaten Schlüssel verschlüsselt wurde, kann sie von jedem gelesen werden, der den öffentlichen Schlüssel besitzt. Umgekehrt kann aber nur derjenige, der den privaten Schlüssel besitzt, eine Nachricht lesen, die mit dem öffentlichen Schlüssel verschlüsselt wurde. Daher wird der private Schlüssel nie dazu benutzt, eine Nachricht zu verschlüsseln – jeder, der im Besitz des öffentlichen Schlüssels ist, könnte sie lesen.

Der Datenaustausch funktioniert dann so, dass die Partei, die die verschlüsselte Nachricht erhalten soll, ihren öffentlichen Schlüssel einfach weitergibt. Dieser kann ohnehin nur sinnvoll verwendet werden, um Nachrichten zu verschlüsseln. Die andere Partei, die die verschlüsselte Nachricht senden will, nutzt diesen Schlüssel nun. Zum Entschlüsseln muss der zu dem öffentlichen Schlüssel passende private Schlüssel benutzt werden.

19.1.3 Hashcodes

Hashcodes sind eine Möglichkeit, binäre Werte flexibler Länge in kleine binäre Werte fester Länge, so genannte Hashcodes, zu verwandeln. Und da auch Texte eigentlich nur Werte binärer Natur sind – jedes Zeichen lässt sich ja durch eine Zahl und somit durch einen binären Wert repräsentieren – können Texte auch durch Hashcodes verschlüsselt werden.

Hashcodes

Eine Änderung eines Textes, sei sie auch noch so klein, wird dabei einen anderen als den ursprünglichen Hashcode erzeugen. Solche Codes können zur Überprüfung einer Nachricht dienen, ähnlich einer Checksumme. Nehmen wir an, jemand möchte eine Nach-

richt verschicken und dem Empfänger ermöglichen, die Korrektheit dieser Nachricht zu überprüfen. Diese Person würde dann die Nachricht schreiben und einen Hashcode dieser Nachricht erzeugen. Auch der Empfänger könnte das tun, wenn er die Nachricht erhalten hat. Der Vergleich beider Hashcodes ermöglicht dann die Überprüfung auf Korrektheit. Ist die Nachricht nach dem Absenden geändert worden, werden die Hashcodes unterschiedlich sein, ansonsten sind sie gleich. Mit Hashcodes lässt sich also eine Art der digitalen Signatur erzeugen.

19.1.4 Digitale Signatur

Digitale Signaturen werden dazu verwendet, die Identität des Absenders zu verifizieren. Dabei wird ein öffentlicher Schlüssel verwendet, mit dessen Hilfe der Absender autentifiziert werden kann.

Hash-Algorithmus

Normalerweise wird zunächst ein Hash-Algorithmus angewandt, der einen eindeutigen Hashcode der Nachricht liefert. Dieser wird dann mit dem privaten Schlüssel des Absenders kodiert, womit sich die digitale Signatur ergibt. Die Kontrolle der Signatur erfolgt beim Empfänger: Dieser erhält die Nachricht zusammen mit der Signatur, die er mit dem öffentlichen Schlüssel des Absenders entschlüsseln kann. Den erhaltenen Hashcode kann er dann kontrollieren, indem er einfach den gleichen Hash-Algorithmus auf die empfangene Nachricht anwendet. Wir wissen bereits, dass, wenn die Nachricht nicht geändert wurde, diese beiden Codes dann gleich sein müssen. Wenn dem so ist, weiß der Empfänger, dass der angegebene Sender der Nachricht auch wirklich der Sender war und (natürlich) dass die Nachricht nicht verändert wurde.

19.2 Hash-Algorithmen

Aus einer Nachricht einen Hashcode zu erzeugen ist mit dem .NET-Framework kein Problem. Möglich sind Hashcodes mit einer Länge von 160 Bits (*SHA1*), 256 Bits (*SHA256*), 384 Bits (*SHA384*) und 5120 Bits (*SHA512*). Das folgende Beispiel erzeugt einen Hashcode aus einer Nachricht, die in eine Textbox eingegeben wurde.

Hashcode erzeugen

Unser Hashcode soll 160 Bit groß sein, also 20 Byte. Verwendet wird in diesem Fall die Klasse `SHA1Managed`, die einen solchen Hash-

Algorithmus implementiert. Da bei diesen Verschlüsselungsalgorithmen mit Byte-Werten gearbeitet wird, muss der Text zunächst in eine Bytefolge umgewandelt werden, bevor er »gehasht« werden kann.

```
Private Function GetBytes(ByVal txt As String) _
    As Byte()

    ' --- Deklarationen
    Dim result As Byte()
    Dim length As Integer
    Dim data As Char() = txt.ToCharArray()
    Dim i As Integer

    ' --- Daten konvertieren
    length = data.Length
    result = New Byte(length) {}

    For i = 0 To length
        result(i) = Convert.ToByte(data(i))
    Next

    Return result

End Function
```

Listing 19.1: HashcodeExample Teil I

Jetzt können Sie den Hashcode erzeugen.

```
Private Sub btnHash_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnHash.Click

    'Deklarationen
    Dim b As Byte

    ' --- Provider für den Hashcode
    Dim sha1 As New SHA1Managed()
    ' --- Hash-Code, hier 160 Bit = 20 Byte
    Dim hash As Byte()
    ' --- Byte-Array für die Daten
    Dim data As Byte() = GetBytes(tbxPlainText.Text)

    hash = sha1.ComputeHash(data)

    ' --- Hashcode In Textbox schreiben

    tbxHash.Clear()
    Me.Text = hash.GetLength(0).ToString()
    For Each b In hash
```

```
        tbxHash.Text += b.ToString()  
        Next
```

```
    End Sub
```

Listing 19.2: HashcodeExample Teil 2

ComputeHash() Die Methode `ComputeHash()` erzeugt den Hashcode aus dem übergebenen `Byte`-Array, das unseren Text repräsentiert. Lassen Sie sich von der Ausgabe nicht täuschen, der Hashcode hat genau 20 Byte (oder 160 Bit), nicht mehr. Falls Sie es kontrollieren wollen, können Sie die Länge einfach über die Methode `GetLength()` ermitteln:

```
int hashLength = hash.GetLength(0);
```

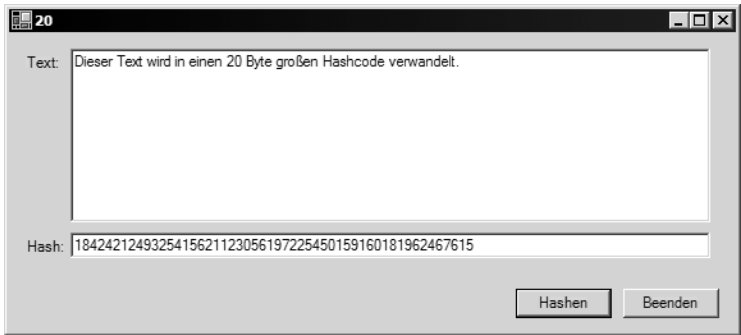


Abbildung 19.1: Text in einen Hashcode verwandelt. In der Kopfzeile steht zusätzlich die Länge des Hashcodes.

In diesem Beispiel wurde exemplarisch der Hash-Algorithmus SHA verwendet, mit einer Hashlänge von 160 Bit. Das .NET-Framework bietet aber noch weitere Implementierungen von Hash-Algorithmen, die unter anderem für Signaturen verwendet werden können.

Klasse	Hash-Algorithmus
MD5CryptoService Provider	Implementiert den MD5-Hash-Algorithmus
SHA1CryptoService Provider	Implementiert den SHA1 (160 Bit) Hash-Algorithmus
SHA1Managed	Implementiert den SHA1 (160 Bit) Hash-Algorithmus. Dieser Algorithmus verwendet ausschließlich managed Code.

Tabelle 19.1: Hash-Algorithmen im .NET-Framework

Klasse	Hash-Algorithmus
SHA256Managed	Implementiert den SHA256 (256 Bit) Hash-Algorithmus. Dieser Algorithmus verwendet ausschließlich managed Code.
SHA384Managed	Implementiert den SHA384 (384 Bit) Hash-Algorithmus. Dieser Algorithmus verwendet ausschließlich managed Code.
SHA512Managed	Implementiert den SHA512 (512 Bit) Hash-Algorithmus. Dieser Algorithmus verwendet ausschließlich managed Code.

Tabelle 19.1: Hash-Algorithmen im .NET-Framework (Fortsetzung)

Die Vorgehensweise ist in jedem Fall gleich. Die Methode `ComputeHash()` wird verwendet, um einen Hashcode aus einem Byte-Array zu erzeugen. Wichtig ist, dass Sie beim Vergleichen zweier Hash-Werte zur Kontrolle eines übertragenen Textes den gleichen Hash-Algorithmus verwenden, da Sie ansonsten unterschiedliche Werte erhalten.

Die Vererbungshierarchie der angegebenen Klassen zeigt Abbildung 19.2.

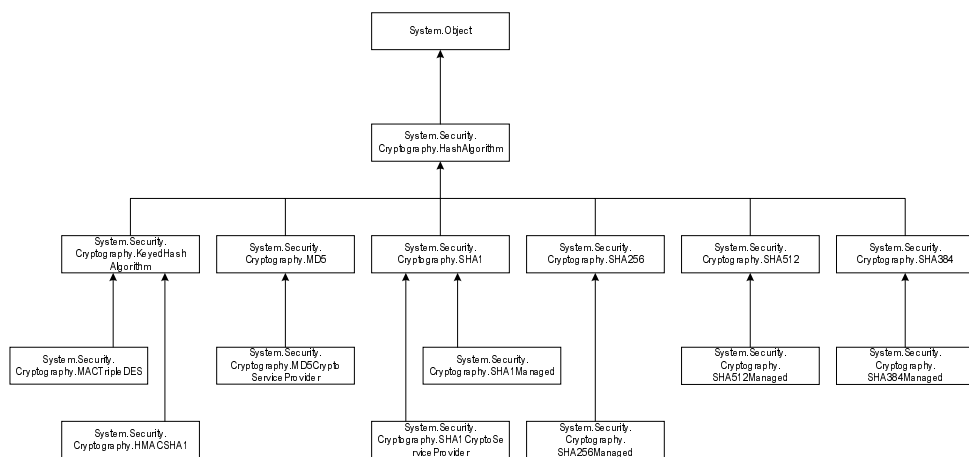


Abbildung 19.2: Die Vererbungshierarchie der Klassen für die Hash-Algorithmen

19.3 Texte verschlüsseln

Die Hauptfunktion von Verschlüsselungsalgorithmen ist natürlich das Verschlüsseln von Texten bzw. von ganzen Textdateien nach einem bestimmten Muster. Das .NET-Framework stellt folgende Klassen zur Verfügung, die eine solche Verschlüsselung durchführen können:

Klasse	Verschlüsselungsalgorithmus
DESCryptoService Provider	Verschlüsselt Texte und Dateien gemäß dem syn- chronen DES-Algorithmus
DSACryptoService Provider	Verschlüsselt Texte und Dateien gemäß dem asym- metrischen DSA-Algorithmus
RC2CryptoService Provider	Verschlüsselt Texte und Dateien gemäß dem sym- metrischen RC2-Algorithmus
RSACryptoService Provider	Verschlüsselt Texte und Dateien gemäß dem asym- metrischen RSA-Algorithmus
RijndaelManaged	Verschlüsselt Texte und Dateien gemäß dem Rijndael-Algorithmus
TripleDESCrypto ServiceProvider	Verschlüsselt Texte und Dateien gemäß dem TripleDES-Algorithmus

Tabelle 19.2: Verschlüsselungsalgorithmen im .NET-Framework

Die Vererbungshierarchie der Klassen aus Tabelle 19.2 zeigt Abbildung 19.3.

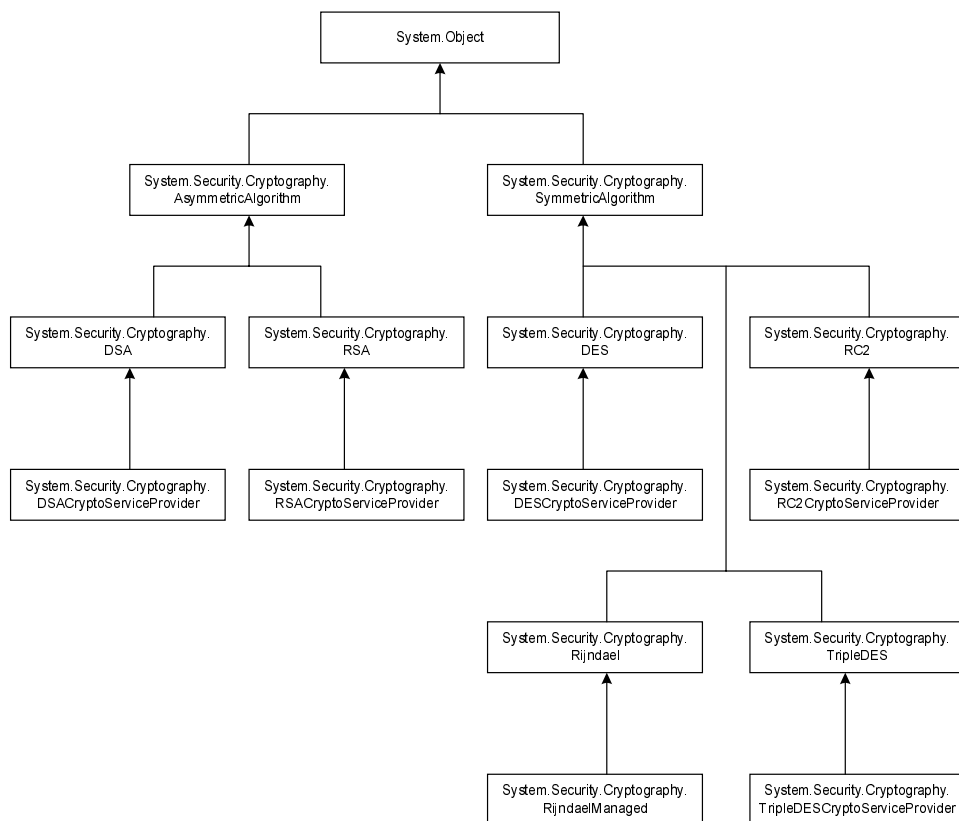


Abbildung 19.3: Vererbungshierarchie der Kryptografie-Klassen

19.3.1 Asymmetrische Verschlüsselung

Die Verschlüsselungsmethoden bzw. die Klassen, die die Verschlüsselungsmethoden implementieren, funktionieren auf die gleiche Weise. Das bedeutet, um zu wissen, wie eine Verschlüsselung durchgeführt wird, muss lediglich eine der relevanten Klassen betrachtet werden. Die Methoden und Eigenschaften dieser Klasse sind äquivalent in den anderen Klassen enthalten. Die Klasse `RSACryptoServiceProvider` soll an dieser Stelle stellvertretend für einen asymmetrischen Verschlüsselungsalgorithmus betrachtet werden.

Laden und Speichern von Schlüsseln

Erzeugen und Speichern eines Schlüsselpaars

Bei der Instanziierung eines Objekts dieser Klasse wird automatisch ein gültiges Schlüsselpaar erzeugt. Dieses kann, um mehrfach verwendet zu werden, auch gespeichert werden. Eine Möglichkeit hierzu ist die Methode `ExportParameters()`, die die Schlüssel in einer `RSAParameters`-Struktur speichert. Die Methode erwartet als Parameter die Angabe, ob der private Schlüssel mit einbezogen werden soll. Wenn Sie nur den öffentlichen Schlüssel speichern und weitergeben wollen, übergeben Sie einfach `false`.

Laden von Schlüsseln aus RSAParameters

Entsprechend kann ein solcher Schlüssel bzw. das gesamte Schlüsselpaar über die Methode `ImportParameters()` wieder eingelesen werden. Obwohl die Klasse bei ihrer Instanziierung ein Schlüsselpaar erzeugt, wird dieses durch das Laden eines Schlüssels überschrieben.

Laden und Speichern eines Schlüssels als XML

Eine weitere Möglichkeit ist, die Schlüssel als XML-String zu speichern bzw. einen Schlüssel aus einem XML-String zu laden. Dazu existieren die Methoden `ToXmlString()` und `FromXmlString()`. Auch die Methode `ToXmlString()` besitzt einen Parameter, mit dem angegeben werden kann, ob der private Schlüssel mit gespeichert werden soll oder nicht.

Verschlüsseln und Entschlüsseln

Byte-Arrays

Nachdem entweder eine Instanz der Klasse erzeugt wurde oder aber ein Schlüssel oder Schlüsselpaar durch eine der angegebenen Methoden geladen wurde, kann Text ver- bzw. entschlüsselt werden. Die Algorithmen arbeiten allerdings mit Byte-Arrays, d.h. zunächst muss der Text in ein Byte-Array umgewandelt werden. Das ist allerdings nicht weiter schwierig.

Encrypt()

Ist eine Nachricht in Form eines Byte-Arrays vorhanden, kann diese verschlüsselt werden. Dazu dient die Methode `Encrypt()`, die zwei Parameter erwartet. Der erste Parameter ist das zu verschlüsselnde Byte-Array, der zweite Parameter bestimmt, ob die direkte RSA-Verschlüsselung verwendet werden soll (nach OAEP) oder nicht (in diesem Fall würde PKCS verwendet). Zurückgeliefert wird wiederum ein Byte-Array, das die nun verschlüsselte Nachricht repräsentiert und somit versendet werden kann.

Das Entschlüsseln funktioniert äquivalent über die Methode `Decrypt()`. Die Parameter sind die gleichen, nur dass diesmal die verschlüsselte Nachricht übergeben und die entschlüsselte Nachricht zurückgeliefert wird.

RSA anwenden

In diesem Beispiel soll ein beliebiger Text verschlüsselt und wieder entschlüsselt werden. Der Schlüssel selbst soll ebenfalls angezeigt werden können, weshalb er mithilfe von `ToXmlString()` gespeichert wird.

Das Hauptformular besteht aus drei Textboxen, die den normalen Text, den verschlüsselten Text und den entschlüsselten Text anzeigen. Eine Richtextbox mit Namen `txtKeys` dient zur Anzeige der Schlüssel, die ja im XML-Format gespeichert werden. Je ein Button dient zum Verschlüsseln und zum Entschlüsseln.

Für den eigentlichen Verschlüsselungsmechanismus wird eine eigene Klasse erstellt. Diese muss folgende Features bereitstellen:

- ▶ Erzeugen einer Instanz von `RSACryptoServiceProvider`
- ▶ Ablegen der Schlüssel in Feldern
- ▶ Laden und Speichern sowohl des öffentlichen Schlüssels allein als auch beider Schlüssel
- ▶ Umwandeln von Text in ein Byte-Array
- ▶ Umwandeln eines Byte-Arrays in Text
- ▶ Zurückliefern der Schlüssel als Text zur Anzeige
- ▶ Entschlüsseln und Verschlüsseln eines Textes

Die gesamte Klasse sehen Sie in Listing 19.3.

```
'=====
' RSAEncryption.sln
' Copyright 2002 Frank Eller
' Verschlüsselung von Daten mit RSA
'=====

Imports System.Security.Cryptography
Imports System.IO

Public Class RsaEncrypt

    Private RSAEncryptor As RSACryptoServiceProvider
    Private RSAPublicKey As String
```

```

Private RSACompleteKey As String

Public Sub New()
    RSAEncryptor = New RSACryptoServiceProvider()
    RSAPublicKey = RSAEncryptor.ToXmlString(False)
    RSACompleteKey = RSAEncryptor.ToXmlString(True)
End Sub

Public Function TextToByteArray( _
    ByVal aText As String) As Byte()

    ' --- Deklarationen
    Dim buffer As Char() = aText.ToCharArray()
    Dim length As Integer = buffer.Length
    Dim result As Byte() = New Byte(length) {}
    Dim i As Integer

    ' --- Text konvertieren
    For i = 0 To length - 1
        result(i) = Convert.ToByte(buffer(i))
    Next

    Return result
End Function

Public Function ByteArrayToText( _
    ByVal theBytes As Byte()) As String

    ' --- Deklarationen
    Dim result As String
    Dim b As Byte

    ' --- Bytes In Text konvertieren
    result = String.Empty
    For Each b In theBytes
        result += Convert.ToChar(b)
    Next

    Return result
End Function

Public Sub SaveBothKeys()

    ' --- Speichern des öffentlichen
    ' --- und des privaten Schlüssels
    Dim sWriter As StreamWriter = _
        New StreamWriter("Complete.key", _
            False, System.Text.Encoding.UTF8)
    sWriter.WriteLine(RSACompleteKey)
    sWriter.Close()
End Sub

```

```

Public Sub savePublicKey()

    ' --- Speichern des öffentlichen
    ' --- Schlüssels zur Weitergabe
    Dim sWriter As StreamWriter = _
        New StreamWriter("Public.key", _
            False, System.Text.Encoding.UTF8)
    sWriter.WriteLine(RSAPublicKey)
    sWriter.Close()
End Sub

Public Sub LoadPublicKey()

    ' --- Lädt einen öffentlichen Schlüssel
    If File.Exists("Public.key") Then
        Dim sReader As StreamReader = _
            New StreamReader("Public.key", _
                System.Text.Encoding.UTF8)
        RSAPublicKey = sReader.ReadToEnd()
        sReader.Close()
    End If
End Sub

Public Sub LoadBothKeys()

    ' --- Lädt beide Schlüssel
    If File.Exists("Complete.key") Then
        Dim sReader As StreamReader = _
            New StreamReader("Complete.key", _
                System.Text.Encoding.UTF8)
        RSACompleteKey = sReader.ReadToEnd()
        sReader.Close()
    End If
End Sub

Public Function EncryptText( _
    ByVal aText As String) As Byte()

    ' --- Verschlüsseln des Textes
    If aText <> String.Empty Then
        LoadPublicKey()
        RSAEncryptor.FromXmlString(RSAPublicKey)
        Return RSAEncryptor.Encrypt( _
            TextToByteArray(aText), False)
    End If
End Function

Public Function DecryptText( _
    ByVal theBytes As Byte()) As String

```

```

' --- Entschlüsseln des Textes
LoadBothKeys()
RSAEncryptor.FromXmlString(RSACompleteKey)
Return ByteArrayToText(RSAEncryptor.Decrypt( _
    theBytes, False))

End Function

Public Function GetKeys() As String()

' --- Schlüsselwerte auslesen und
' --- als String-Array zurückliefern
Dim arl As New ArrayList()

arl.Add("Öffentlicher Schlüssel:")
arl.Add("")
arl.Add(RSAPublicKey)
arl.Add("")
arl.Add("Privater und öffentlicher Schlüssel:")
arl.Add("")
arl.Add(RSACompleteKey)

Return arl.ToArray(Type.GetType("System.String"))

End Function
End Class

```

Listing 19.3: Die Klasse RSAEncrypt [RSAEncryption.sln]

Sämtliche Funktionalität ist in dieser Klasse implementiert. Beim Erzeugen einer Instanz werden die Schlüsselwerte über die Methode `ToXmlString()` in die Felder `RSAPublicKey` und `RSACompleteKey` abgelegt. Zum Verschlüsseln wird dann nur der öffentliche Schlüssel geladen und der Text, nachdem er in ein Byte-Array umgewandelt wurde, verschlüsselt. Der umgekehrte Weg entschlüsselt einen Text, indem der gesamte Schlüssel geladen, das verschlüsselte Byte-Array entschlüsselt und dann wieder in Text zurückverwandelt wird.

Cryptographic-Exception

Die Implementierung von `RSACryptoServiceProvider` erlaubt das Verschlüsseln nur mit einem öffentlichen Schlüssel, d.h. der private Schlüsselteil muss nicht geladen sein. Zum Entschlüsseln wird er aber benötigt, ansonsten wird eine Exception vom Typ `CryptographicException` ausgelöst.

Im Hauptformular werden nur die beiden Methoden zum Verschlüsseln und Entschlüsseln, sowie ein Feld zur Aufnahme des verschlüsselten Byte-Arrays benötigt.


```
'=====
' RSAEncryption.sln
' Copyright 2002 Frank Eller
' Verschlüsselung von Daten mit RSA
'=====

Public Class Form1
    Inherits System.Windows.Forms.Form

    #Region " Windows Form Designer generated code "

        ' Der Code, der von VS.NET eingefügt wurde, wird
        ' aus Übersichtsgründen nicht dargestellt.

    #End Region

    Private encryptedBytes As Byte()

    Private Sub btnEncrypt_Click( _
        ByVal sender As System.Object, _
        ByVal e As System.EventArgs) _
        Handles btnEncrypt.Click

        ' --- Hier wird der Text verschlüsselt
        Dim aRsa As New RsaEncrypt()
        aRsa.SaveBothKeys()
        aRsa.savePublicKey()

        encryptedBytes = aRsa.EncryptText(txtNormal.Text)

        ' --- Anzeigen In zweiter Textbox
        txtEncrypted.Text = _
            aRsa.ByteArrayToText(encryptedBytes)

        ' --- Keys anzeigen
        txtKeys.Lines = aRsa.GetKeys

    End Sub

    Private Sub btnDecrypt_Click( _
        ByVal sender As System.Object, _
        ByVal e As System.EventArgs) _
        Handles btnDecrypt.Click

        ' --- Hier wird der Text entschlüsselt
        Dim aRsa As New RsaEncrypt()
        txtDecrypted.Text = _
            aRsa.DecryptText(encryptedBytes)

    End Sub
```

```

Private Sub btnExit_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnExit.Click

    Me.Close()
End Sub
End Class

```

Listing 19.4: Asymmetrisches Verschlüsselungsverfahren [RSAEncryption.sln]

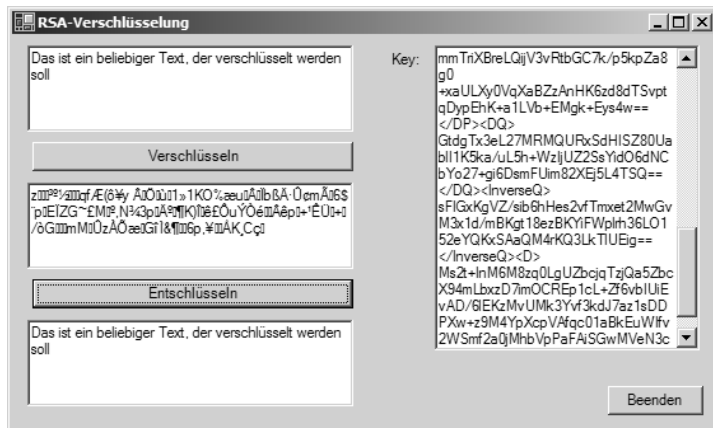


Abbildung 19.4: Ein RSA-verschlüsselter Text

Abbildung 19.4 zeigt das Programm zur Laufzeit mit eingegebenem Text. Rechts ist der Schlüssel in XML-Form zu sehen.

19.3.2 Symmetrische Verschlüsselung

Die asymmetrische Verschlüsselung funktioniert ja recht einfach, da man lediglich bestimmen muss, welcher Schlüssel benutzt wird, und dann mit den Methoden `Encrypt()` bzw. `Decrypt()` die Ver- bzw. Entschlüsselung durchführen.

ICryptoTransform

Bei der symmetrischen Verschlüsselung funktioniert es nicht ganz so einfach. Hier sind mehrere Klassen an dem Gesamtvorgang beteiligt. Zunächst wird hier der Schlüssel nicht von der Klasse selbst erzeugt, sondern muss außerhalb erzeugt werden. Dabei handelt es sich nicht nur um den Schlüssel selbst, sondern auch um einen so genannten Initialisierungsvektor, der ebenfalls benötigt wird. Danach wird eine Implementierung des Interfaces `ICrypto`

`Transform` benötigt, das anhand des Schlüssels eine Verschlüsselung oder Entschlüsselung vornimmt.

Verschlüsseln und Entschlüsseln

Das eigentliche Verschlüsseln des Texts geschieht über eine Instanz der Klasse `CryptoStream`, die sozusagen als eine Zwischenstation dient. `CryptoStream` benötigt als Parameter für den Konstruktor sowohl einen Eingangs-Stream beliebiger Art, das Interface `ICryptoTransform` und den Lesemodus. Während des Lesevorgangs führt `CryptoStream` die Änderung der Eingangsdaten durch, d.h. die Bytes, die gelesen werden, werden sofort umgewandelt und stehen dann in ver- bzw. entschlüsselter Form vor. Über ein weiteres `Stream`-Objekt können sie dann wieder geschrieben werden. Bild zeigt schematisch die Vorgehensweise.

CryptoStream

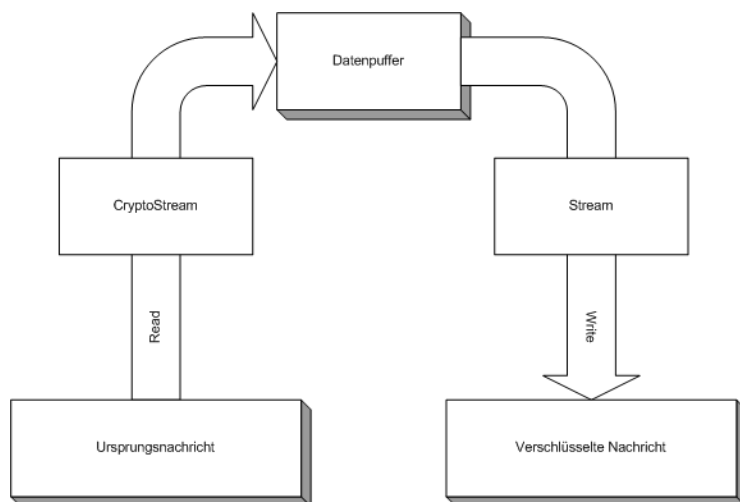


Abbildung 19.5: Verschlüsseln einer Nachricht mithilfe symmetrischer Verschlüsselung

Der Grund dafür, dass hier ein `Stream`-Objekt verwendet werden muss, liegt im eigentlichen Verschlüsselungsalgorithmus. Dieser arbeitet blockweise, verschlüsselt also Stück für Stück. Allerdings ist er, da nur ein Schlüssel verwendet wird, wesentlich schneller als ein asymmetrischer Algorithmus, der die gleiche Menge an Daten bearbeiten muss.

**CreateEncryptor()
und Create-
Decryptor()**

`CryptoStream` benötigt wie angesprochen eine Instanz des Interfaces `ICryptoTransform`, in dem die Routinen zum Verschlüsseln festgelegt sind. Die Ausdrucksweise an dieser Stelle ist etwas verwirrend, denn von einem Interface kann natürlich keine Instanz erzeugt werden. Dennoch ist der Ergebniswert der relevanten Methoden `CreateEncryptor()` bzw. `CreateDecryptor()` `ICryptoTransform`, nicht etwa ein Objekt. Reell gesehen wird natürlich ein Objekt zurückgeliefert, das die in `ICryptoTransform` deklarierten Methoden beinhaltet.

`CryptoStream` kann so mit jedem beliebigen symmetrischen Verschlüsselungsalgorithmus arbeiten. Das erzeugte Verschlüsselungsobjekt in Form des Interface `ICryptoTransform` wird zusammen mit dem Eingabepuffer und dem Modus an den Konstruktor der Klasse `CryptoStream` übergeben.

Die Methode `Read()` dient danach zum Einlesen. Dieser Methode wird unter anderem übergeben, wie viele Bytes vom Eingabestream gelesen werden sollen, zurückgeliefert wird die Anzahl der tatsächlich gelesenen Bytes. Es wird also Blockweise gearbeitet, wodurch auch große Datenmengen leicht bearbeitet werden können.

Erzeugen des privaten Schlüssels**Key und IV**

Als privater Schlüssel dient eine Kombination aus Schlüssel (*Key*) und Initialisierungsvektor (*IV*). Der IV wird benötigt, um die Verschlüsselung wirklich sicher zu machen. Angenommen, Sie möchten einen Text verschlüsseln und verwenden keinen Initialisierungsvektor. Dann würden gleiche Textblöcke auch gleiche verschlüsselte Bytecodes ergeben, was ja nicht wünschenswert ist. Erfahrene Kryptoanalytiker könnten dann durch verschiedene Techniken der verwendeten Schlüssel herausfinden.

Aus diesem Grund passieren zwei Dinge. Einmal wird jeder Textblock unter Zuhilfenahme des vorherigen Textblocks verschlüsselt, d.h. es kann nicht mehr passieren, dass zwei gleiche Textblöcke nach der Verschlüsselung auch zwei gleiche Bytesequenzen ergeben. Zusätzlich wird der IV verwendet, um den ersten Textblock zu verschlüsseln, fließt also dort mit ein. Damit ist eine Entschlüsselung schon ein gutes Stück unwahrscheinlicher geworden.

Zum Erzeugen eines Schlüssels werden zwei Dinge benötigt. Einmal benötigt man eine komplett zufällige Bytefolge. Diese kann beliebig groß sein. Aus dieser Bytefolge wird ein Hashcode ermittelt, der bekanntlich eine feste Größe hat. Aus diesem Hashcode werden dann die Bestandteile Key und IV ermittelt.

Zum Erzeugen der zufälligen Bytefolge gibt es im Namespace `System.Security.Cryptography` die Klasse `RNGCryptoServiceProvider`, die eigentlich einen Zufallszahlengenerator darstellt. Da es in der Kryptografie ein wenig sicherer zugehen soll als in der normalen Welt und diese Zufallszahlen auch ein wenig zufälliger sein sollen, als die durch die Klasse `Random` ermittelten, gibt es eben hier eine Klasse, die noch zufälliger Zahlen erzeugt.

`RNGCryptoServiceProvider` beinhaltet außerdem zwei Methoden, nämlich `GetBytes()` und `GetNonZeroBytes()`, die automatisch eine zufällige Bytefolge ermitteln. Sie erwarten als Parameter ein Byte-Array beliebiger Größe, das dann mit zufälligen Werten gefüllt wird. Bei der Methode `GetNonZeroBytes()` wird zusätzlich darauf geachtet, dass keines der Bytes gleich 0 ist.

Zum Hashen der Bytefolge kann ein beliebiger Hash-Algorithmus verwendet werden. Key und IV werden dann aus den Bytes des Hashcodes ermittelt.

**RNGCrypto
ServiceProvider**

**GetBytes() und
GetNonZero
Bytes()**

DES anwenden

In diesem Beispiel wird der DES-Verschlüsselungsalgorithmus verwendet. Auch hier gilt, dass die anderen symmetrischen Verschlüsselungsverfahren ebenso arbeiten. Zum Erstellen des Schlüssels wird ein SHA-Hash-Algorithmus verwendet, der einen 20 Bytes großen Hash ermittelt. Wie bei der asymmetrischen Verschlüsselung auch arbeiten wir hier mit einem Byte-Puffer, d.h. es kommen wieder die gleichen Methoden wie beim letzten Beispiel zum Einsatz, die wir zum Umwandeln des Textes verwenden.

Das Hauptformular besteht wieder aus drei Textboxen für den Ursprungstext, den verschlüsselten und den entschlüsselten Text. Die Anzeige des Schlüssels ist diesmal nicht implementiert, da er lediglich als Bytefolge vorliegt. Wenn Sie möchten, können Sie diese Anzeige aber noch einbauen. Ansonsten gibt es nur noch je einen Button zum Verschlüsseln und Entschlüsseln.

Die Funktionalität wurde wieder in eine eigene Klasse gepackt, diesmal genannt `DESEncrypt`. Diese implementiert im Grundsatz

die gleichen Methoden wie die asymmetrische Klasse `RSACrypt`, allerdings mit dem Unterschied, dass die Erzeugung des Schlüssels manuell geschieht. Hier die passende Methode `CreateKey()` zur Erzeugung des Schlüssels:

```
Public Sub CreateKey()
    Dim keySource As Byte() = New Byte(99) {}
    Dim rng As New RNGCryptoServiceProvider()
    Dim i As Integer

    ' --- Zufällige Bytefolge erzeugen
    rng.GetNonZeroBytes(keySource)

    ' --- Bytefolge Hashen mit SHA1 (160 Bit)
    Dim sha1 As New SHA1CryptoServiceProvider()
    sha1.ComputeHash(keySource)

    ' --- Erzeugen des privaten Schlüssels
    For i = 0 To 7
        SymmetricKey(i) = sha1.Hash(i)
        InitVector(i) = sha1.Hash(i + 8)
    Next

End Sub
```

Listing 19.5: Erzeugen eines Schlüssels für den DES-Algorithmus

Sowohl Schlüssel als auch Initialisierungsvektor bestehen bei diesem Beispiel aus 8 Bytes. Sie werden in den Variablen `SymmetricKey` und `InitVector` gespeichert, die als Felder der Klasse deklariert sind. Die eigentliche Verschlüsselungsroutine ist dann nicht mehr weiter schwierig.

Da hier direkt im Speicher gearbeitet werden kann, werden `MemoryStream`-Klassen verwendet, um den benötigten `CryptoStream` zu erzeugen. Die Instanz des benötigten Interfaces wird über die Methode `CreateEncryptor()` erzeugt. Analog würde die passende Interface-Instanz zum Entschlüsseln über `CreateDecryptor()` erzeugt, ansonsten sind die beiden Methoden zum Verschlüsseln und Entschlüsseln gleich.

```
Public Function Encrypt(ByVal source As Byte()) _
    As Byte()

    ' --- Deklarationen
    Dim result As Byte()
    Dim buffer As Byte() = New Byte(4096) {}
    Dim des As New DESCryptoServiceProvider()
```

```
Dim inStream As New MemoryStream()  
Dim outStream As New MemoryStream()  
Dim readLength As Integer = 0  
Dim crypter As ICryptoTransform  
Dim encryptor As CryptoStream  
  
' --- Zwei MemoryStreams für die Daten  
inStream = New MemoryStream(source)  
outStream = New MemoryStream()  
  
' --- Das Interface und der CryptoStream  
crypter = des.CreateEncryptor( _  
    SymmetricKey, InitVector)  
encryptor = New CryptoStream( _  
    inStream, crypter, CryptoStreamMode.Read)  
  
' --- Jetzt verschlüsseln  
Do  
    readLength = encryptor.Read(buffer, 0, 4096)  
    outStream.Write(buffer, 0, readLength)  
Loop While (readLength > 0)  
  
' --- Resultat als Byte-Array  
result = outStream.ToArray()  
  
' --- Streams schließen  
inStream.Close()  
outStream.Close()  
  
Return result  
End Function
```

Listing 19.6: Die Verschlüsselung der Daten

Die eigentliche Arbeit der Routine wird in der `do`-Schleife gemacht. Die Abbruchbedingung richtet sich nach der Länge der tatsächlich gelesenen Bytes. Ist diese 0, wird die Schleife beendet.

Es wird immer versucht, 4096 Bytes auf einen Schlag zu lesen. Diesen Wert kann man auch ändern, man muss nur dafür sorgen, dass der Puffer, der verwendet wird, groß genug ist. `readLength` enthält die Anzahl Bytes, die tatsächlich gelesen wurde. Diese benötigen wir auch zum Schreiben der nun verschlüsselten Daten in den Ausgabestream, da wir ja nur so viele Zeichen schreiben dürfen, wie tatsächlich im Puffer vorhanden sind.

Wie angesprochen entspricht die Methode zur Entschlüsselung der gezeigten Methode. In der Klasse sind außerdem noch die beiden bereits bekannten Methoden zur Umwandlung eines Strings

in ein Byte-Array und zurück implementiert. Im Hauptformular müssen nun nur noch die Umwandlung der Texte und der Aufruf der benötigten Methode implementiert werden. Den Quelltext des gesamten Hauptformulars sehen Sie in Listing 19.6.

```
Imports System.Security.Cryptography
Imports System.IO

Public Class Form1
    Inherits System.Windows.Forms.Form

    #Region " Windows Form Designer generated code "

        'Der Code, der von VS.NET eingefügt wurde, wird
        'aus Übersichtsgründen nicht dargestellt.

    #End Region

    Public Encryptor = New DESEncrypt()

    Private Sub btnEncrypt_Click( _
        ByVal sender As System.Object, _
        ByVal e As System.EventArgs) _
        Handles btnEncrypt.Click

        ' --- Umwandeln des Texts In Bytes
        Dim buffer As Byte() = _
            encryptor.TextToByteArray(tbxPlainText.Text)
        Dim result As Byte() = encryptor.Encrypt(buffer)
        tbxEncrypt.Text = _
            encryptor.ByteArrayToText(result)

    End Sub

    Private Sub btnDecrypt_Click( _
        ByVal sender As System.Object, _
        ByVal e As System.EventArgs) _
        Handles btnDecrypt.Click

        ' --- Umwandeln des Texts In Bytes
        Dim buffer As Byte() = _
            Encryptor.TextToByteArray(tbxEncrypt.Text)
        Dim result As Byte() = Encryptor.Decrypt(buffer)
        tbxDecrypt.Text = _
            Encryptor.ByteArrayToText(result)

    End Sub

    Private Sub Form1_Load( _
        ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles MyBase.Load
        Me.Encryptor.CreateKey()
```



```

End Sub

Private Sub btnExit_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnExit.Click

    Me.Close()
End Sub
End Class

```

Listing 19.7: Der Quelltext des Hauptformulars [DESEncryption.sln]

Das Ergebnis des Programms sehen Sie in Abbildung 19.6.

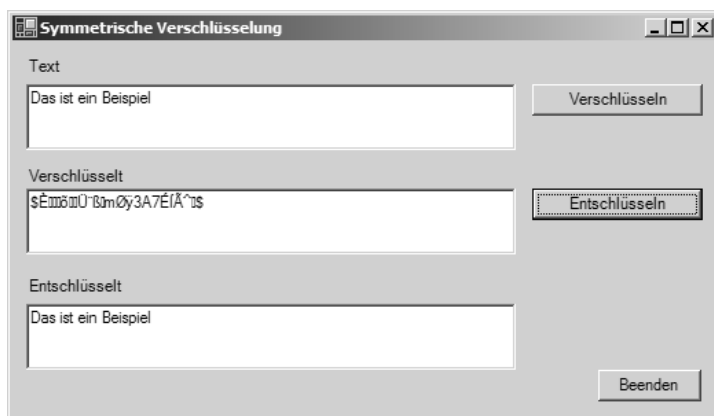


Abbildung 19.6: Der DES-Algorithmus zur Laufzeit [DESEncryption.sln]

19.4 Verschlüsselung in der Realität

An dieser Stelle soll es jetzt um ein Szenario gehen, wie es in der Realität tatsächlich Verwendung finden würde. Üblicherweise werden solche Szenarien mit zwei Personen namens Alice und Bob durchgespielt.

19.4.1 Alice und Bob

Angenommen, Alice möchte Bob eine Nachricht schicken. Sie möchte aber sicherstellen, dass niemand anderes als Bob diese Nachricht lesen kann. Das könnte natürlich sehr leicht erreicht werden. Sie müsste nur Bobs öffentlichen Schlüssel verwenden und nur Bob wäre in der Lage, die Nachricht zu entschlüsseln.

Diese Methode hat allerdings einen Nachteil. Zwar könnte Alice sicher sein, dass nur Bob die Nachricht entschlüsseln kann, aber dadurch dass sie Bobs öffentlichen Schlüssel benutzt, könnte Bob nicht sicher sein, dass die Nachricht auch wirklich von Alice kommt. Irgendjemand, der Alice' Nachricht abgefangen hat, könnte diese zwar nicht entschlüsseln, aber er könnte sie zurückhalten und seinerseits eine Nachricht an Bob schicken, die ebenfalls mit dessen öffentlichem Schlüssel chiffriert ist. Denn an diesen kommt man ja leicht heran.

Also muss eine andere Lösung gefunden werden. Um wirklich sicherzustellen, dass die Nachricht nur von einer bestimmten Person kommen kann, könnte man einfach diese Nachricht signieren, man würde also zwei asymmetrische Verfahren verwenden. Alice verschlüsselt die Nachricht mit Bobs öffentlichem Schlüssel und signiert sie unter Zuhilfenahme ihres privaten Schlüssels. Mit Alice' öffentlichem Schlüssel könnte Bob nun kontrollieren, ob die Nachricht wirklich von ihr kommt, denn mit diesem Schlüssel kann nur Alice' Signatur gelesen werden.

**Verschlüsselungs-
algorithmen
gemeinsam
verwenden**

Eine weitere Möglichkeit, um die es auch hier gehen soll, ist die Verbindung eines asymmetrischen und eines symmetrischen Verschlüsselungsalgorithmus. Alice würde in diesem Fall Bob ihren öffentlichen Schlüssel übergeben, was problemlos möglich ist. Bob hingegen erzeugt einen privaten Schlüssel eines symmetrischen Verfahrens, den er mit Alice' öffentlichem Schlüssel verschlüsselt und ihr schickt. Niemand, der diese Nachricht anfängt, kann Bobs privaten Schlüssel nun lesbar machen – außer Alice. Die entschlüsselt den privaten Schlüssel und verschlüsselt ihre Nachricht damit. Die Verschlüsselungsverfahren müssen natürlich in diesem Fall abgesprochen werden.

Bob erhält nun eine Nachricht von Alice, die mit seinem privaten Schlüssel chiffriert wurde. Damit kann er sicher sein, dass, wenn er die Nachricht entschlüsseln kann, diese von Alice kommt, denn niemand anderes war in der Lage, den privaten Schlüssel von Bob zu benutzen – er war ja mit Alice' öffentlichem Schlüssel chiffriert. Diese Situation soll nun anhand eines Programms nachgebaut werden.

19.4.2 Die Funktionalität

Es wird eine Klasse benötigt, die die angegebenen Verfahren implementiert. Diese Klasse muss folgende Dinge implementieren:

Benötigte Funktionen

- ▶ Erzeugen eines asymmetrischen Schlüsselpaars (Alice), wobei der öffentliche Schlüssel gespeichert und versendet werden kann. Der private Schlüssel soll auch gespeichert werden können, damit Alice ihn ggf. später wieder laden kann.
- ▶ Erzeugen eines symmetrischen Schlüssels (Bob), der später zur Verschlüsselung einer Nachricht benutzt wird. Auch dieser Schlüssel muss gespeichert werden können, damit Bob ihn später zum Entschlüsseln der Nachricht benutzen kann.
- ▶ Verschlüsseln des privaten Schlüssels über ein asymmetrisches Verfahren (Bob) und Versenden desselben. Das Versenden wird in diesem Fall natürlich durch Speichern der Daten simuliert.
- ▶ Entschlüsseln des verschlüsselten privaten Schlüssels (Alice)
- ▶ Verschlüsseln einer Nachricht mit dem entschlüsselten privaten Schlüssel (Alice)
- ▶ Versenden der Nachricht (Alice)
- ▶ Entschlüsseln der Nachricht mit dem privaten Schlüssel, der zuvor gespeichert worden war (Bob)

In diesem Beispiel befinden sich keine Routinen zur Übertragung der Daten, stattdessen werden diese nur normal in Dateien gespeichert und vom anderen Programm entsprechend wieder geladen. Da es sich aber lediglich um Bytefolgen handelt, könnte man die Daten auch per E-Mail verschicken – das würde ebenso funktionieren. Spezielle Routinen hierfür wurden der Einfachheit halber nicht implementiert.



Die Funktionalitäten werden in zwei verschiedene Klassen aufgeteilt, eine für die Funktionalität, die Alice benötigt und eine für die Funktionalität, die Bob benötigt. Der Name der Klassen ist `MessageExchange`. Beide Klassen haben gemeinsam, dass sie Funktionen zur Umwandlung von Text in ein `Byte-Array` und zurück implementieren. Außerdem wurden für dieses Beispiel zur Vereinfachung die Dateinamen zum Speichern und Laden der Schlüssel und des Textes von vorn herein festgelegt. Grundsätzlich lassen sich die beiden Klassen aber leicht ausbauen.

In diesem Projekt gibt es je ein Programm für Alice und eines für Bob. Daher gibt es auch zwei Klassen, die den gleichen Namen tragen können (bei unterschiedlicher Funktionalität). Den gemeinsamen Teil beider Klassen sehen Sie in Listing 19.7.

```
' --- Dateinamen der Einfachheit halber hartcodiert
Private fnRSAPublicKey As String = _
    "c:\rsapublic.key"
Private fnRSACompleteKey As String = _
    "c:\rsacomplete.key"
Private fnDESKey As String = "c:\des.key"
Private fnDESEncryptedKey As String = _
    "c:\desencrypted.key"
Private fnMessageEncrypted As String = _
    "C:\encrypted.msg"

' --- Felder für Bob und Alice
Private SymmetricKey As Byte() = New Byte(7) {}
Private InitVector As Byte() = New Byte(7) {}
Private RSAPublicKey As String = String.Empty
Private RSACompleteKey As String = String.Empty
Private SymmetricKeyEncrypted As Byte()
Private SymmetricKeyComplete As Byte() = _
    New Byte(15) {}
Private randomValue As Byte() = New Byte(99) {}

' --- Text zu Byte-Array konvertieren
Public Function TextToByteArray( _
    ByVal aText As String) As Byte()

    ' --- Deklarationen
    Dim buffer As Char() = aText.ToCharArray()
    Dim length As Integer = buffer.Length
    Dim result As Byte() = New Byte(length - 1) {}
    Dim i As Integer

    For i = 0 To length - 1
        result(i) = Convert.ToByte(buffer(i))
    Next

    Return result
End Function

' --- Byte-Array zu Text konvertieren
Public Function ByteArrayToText( _
    ByVal theBytes As Byte()) As String

    ' --- Deklarationen
    Dim result As String
    Dim b As Byte
```

```

    result = String.Empty
    For Each b In theBytes
        result += Convert.ToChar(b)
    Next

    Return result
End Function

```

Listing 19.8: Der gemeinsame Teil der MessageExchange-Klassen [BobAndAlice.sln]

Die Funktionalität von Alice

Die erste Funktion, die Alice ausführen muss, ist das Senden eines öffentlichen Schlüssels an Bob. Den dazugehörigen privaten Schlüssel speichert Alice ebenfalls ab, denn sie benötigt ihn ja später wieder, um die Daten, die Bob sendet zu entschlüsseln. Die Funktion `Alice_CreateRSAKeys()` aus Listing 19.8 erzeugt ein Schlüsselpaar. `Alice_SavePublicKey()` speichert den öffentlichen Schlüssel für Bob, `Alice_SaveCompleteKey()` speichert beide Schlüssel, denn Alice benötigt diese ja später noch einmal.

```

Public Sub Alice_CreateRSAKeys()

    ' --- Erzeugt ein RSA-Schlüsselpaar
    Dim rsa As New RSACryptoServiceProvider()
    RSAPublicKey = rsa.ToXmlString(False)
    RSACompleteKey = rsa.ToXmlString(True)
End Sub

Public Sub Alice_SavePublicKey()

    ' --- Speichert den öffentlichen Schlüssel des
    ' --- asymmetrischen Schlüsselpaars
    Dim sWriter As New StreamWriter(fnRSAPublicKey, _
        False, System.Text.Encoding.UTF8)
    sWriter.WriteLine(RSAPublicKey)
    sWriter.Close()
End Sub

Public Sub Alice_SaveCompleteKey()

    ' --- Speichert den öffentlichen und privaten
    ' --- Schlüssel des asymmetrischen Schlüsselpaars
    Dim sWriter As New StreamWriter( _
        fnRSACompleteKey, False, _
        System.Text.Encoding.UTF8)
    sWriter.WriteLine(RSACompleteKey)
    sWriter.Close()
End Sub

```

Listing 19.9: Erzeugen eines Schlüsselpaars [BobAndAlice.sln]

Alle drei Routinen sind eigentlich sehr leicht verständlich, es ist im Vergleich zu den vorhergehenden Beispielen des Kapitels kaum ein Unterschied zu sehen.

Nachdem der Schlüssel an Bob gesendet wurde, erzeugt dieser einen privaten Schlüssel für ein symmetrisches Verschlüsselungsverfahren. Diesen chiffriert er mit Alice' öffentlichem Schlüssel und schickt ihn dann an Alice. Die nächste Funktion, die Alice vornehmen muss, ist, diese Daten zu laden und zu entschlüsseln. Damit erhält sie einen privaten Schlüssel, den nur Bob und sie kennen.

```
Public Sub Alice_LoadPrivateKey()
    ' --- Lädt den privaten Schlüssel von Bob,
    ' --- dechiffriert ihn und teilt Key / IV auf
    Dim sReader As StreamReader
    Dim rsa As RSACryptoServiceProvider
    Dim fStream As FileStream
    Dim bReader As BinaryReader
    Dim i As Integer

    ' --- RSA-Schlüssel laden
    sReader = New StreamReader(fnRSACompleteKey, _
        System.Text.Encoding.UTF8)
    RSACompleteKey = sReader.ReadLine()
    rsa = New RSACryptoServiceProvider()

    ' --- RSA-Schlüssel an Algorithmus weitergeben
    rsa.FromXmlString(RSACompleteKey)
    sReader.Close()

    ' --- Verschlüsselten privaten Schlüssel Laden
    fStream = New FileStream( _
        fnDESEncryptedKey, FileMode.Open)
    bReader = New BinaryReader(fStream)

    SymmetricKeyEncrypted = bReader.ReadBytes(1000)
    fStream.Close()
    bReader.Close()

    ' --- Verschlüsselten privaten Schlüssel
    ' --- dechiffrieren
    SymmetricKeyComplete = _
        rsa.Decrypt(SymmetricKeyEncrypted, False)

    For i = 0 To 7
        SymmetricKey(i) = SymmetricKeyComplete(i)
        InitVector(i) = SymmetricKeyComplete(i + 8)
    Next

End Sub
```

Listing 19.10: Laden des privaten Schlüssels von Bob [BobAndAlice.sln]

Mit diesem privaten Schlüssel chiffriert sie nun ihre Nachricht und sendet sie an Bob. Dieser kann sicher sein, dass die Nachricht, wenn sie korrekt entschlüsselt werden kann, von Alice kommen muss. Damit wurde die Nachricht praktisch sowohl verschlüsselt übermittelt als auch quasi signiert.

```
Public Function EncryptMessage( _  
    ByVal message As String) As Byte()  
  
    ' --- Verschlüsselt eine Bytefolge  
    ' --- Der Algorithmus ist DES  
    ' --- Der Schlüssel ist In privateKey und  
    ' --- privateIV gespeichert  
    Dim result As Byte()  
    Dim buffer As Byte() = New Byte(4096) {}  
    Dim des As DESCryptoServiceProvider  
    Dim inStream As MemoryStream  
    Dim outStream As MemoryStream  
    Dim readLength As Integer  
    Dim crypter As ICryptoTransform  
    Dim encryptStream As CryptoStream  
    Dim messageBuffer As Byte()  
  
    messageBuffer = TextToByteArray(message)  
    des = New DESCryptoServiceProvider()  
  
    inStream = New MemoryStream(messageBuffer)  
    outStream = New MemoryStream()  
  
    readLength = 0  
  
    crypter = des.CreateEncryptor( _  
        SymmetricKey, InitVector)  
    encryptStream = New CryptoStream( _  
        inStream, crypter, CryptoStreamMode.Read)  
  
    Do  
        readLength = encryptStream.Read( _  
            buffer, 0, 4096)  
        outStream.Write(buffer, 0, readLength)  
    Loop While (readLength > 0)  
  
    result = outStream.ToArray()  
  
    inStream.Close()  
    outStream.Close()  
  
    Return result  
  
End Function
```

```

Public Sub Alice_SaveMessage( _
    ByVal message As Byte())

    ' --- Speichert die verschlüsselte Nachricht
    Dim fStream As FileStream
    Dim b As Byte

    fStream = New FileStream( _
        Me.fnMessageEncrypted, FileMode.Create)

    For Each b In message
        fStream.WriteByte(b)
    Next

    fStream.Close()
End Sub

```

Listing 19.11: Verschlüsseln und Speichern des Schlüssels [BobAndAlice.sln]

Damit ist der Teil von Alice erledigt, die restliche Funktionalität liegt bei Bob.

Die Funktionalität von Bob

Wenn Bob den öffentlichen Schlüssel von Alice erhält, erstellt er zunächst einen privaten Schlüssel. Diesen speichert er ab, weil er ihn später wieder zum Entschlüsseln der Nachricht benötigt.

```

Public Sub Bob_SaveKey(ByVal filename As String, _
    ByVal aKey As Byte())

    ' --- Speichern eines beliebigen Schlüssels
    ' --- In einer Datei
    Dim fStream As FileStream
    Dim b As Byte

    fStream = New FileStream( _
        filename, FileMode.Create)

    For Each b In aKey
        fStream.WriteByte(b)
    Next

    fStream.Close()
End Sub

Public Sub Bob_CreatePrivateKey()
    ' --- Erzeugt einen privaten Schlüssel für den
    ' --- DSA-Algorithmus
    Dim rng As New RNGCryptoServiceProvider()
    Dim sha1 As New SHA1CryptoServiceProvider()
    Dim i As Integer

```



```

' --- Zufallswerte für Schlüssel holen
rng.GetNonZeroBytes(randomValue)
' --- Gesamtschlüssel holen,
' --- aber noch nicht aufteilen
sha1.ComputeHash(randomValue)

For i = 0 To 15
    Me.SymmetricKeyComplete(i) = sha1.Hash(i)
Next
' --- Privaten Schlüssel speichern
Bob_SaveKey(fnDESKey, SymmetricKeyComplete)
End Sub

```

Listing 19.12: Erzeugen und Speichern des privaten Schlüssels [BobAndAlice.sln]

Jetzt wird Alice' öffentlicher Schlüssel geladen, der private Schlüssel damit verschlüsselt und gespeichert.

```

Public Sub Bob_LoadPublicKey()

' --- Lädt einen öffentlichen Schlüssel
Dim sReader As StreamReader

If File.Exists(fnRSAPublicKey) Then
    sReader = New StreamReader( _
        fnRSAPublicKey, System.Text.Encoding.UTF8)
    RSAPublicKey = sReader.ReadToEnd()
    sReader.Close()
End If

End Sub

Public Sub Bob_EncryptPrivateKey()
' --- Verschlüsseln des privaten Schlüssels
' --- mit dem öffentlichen Schlüssel von Alice
Dim rsa As New RSACryptoServiceProvider()

rsa.FromXmlString(RSAPublicKey)
SymmetricKeyEncrypted = _
    rsa.Encrypt(SymmetricKeyComplete, False)
End Sub

Public Sub Bob_SaveEncryptedPrivateKey()
Bob_SaveKey(fnDESEncryptedKey, _
    SymmetricKeyEncrypted)
End Sub

```

Listing 19.13: Laden des öffentlichen Schlüssels von Alice, Verschlüsseln des privaten Schlüssels und Speichern [BobAndAlice.sln]

Nachdem Bob seinen Schlüssel verschickt hat, wartet er auf die Nachricht von Alice. Dann lädt er den gespeicherten privaten Schlüssel und entschlüsselt die Nachricht.

```
Public Sub Bob_LoadPrivateKey()
    ' --- Laden des Private Schlüssels
    Dim fStream As FileStream
    Dim i As Integer

    fStream = New FileStream(fnDESKey, FileMode.Open)
    ' --- Der Private Schlüssel hat genau 16 Byte
    fStream.Read(SymmetricKeyComplete, 0, 16)
    ' --- Aufteilen In Key und IV
    For i = 0 To 7
        SymmetricKey(i) = SymmetricKeyComplete(i)
        InitVector(i) = SymmetricKeyComplete(i + 8)
    Next

    fStream.Close()
End Sub

Public Function Bob_DecryptMessage() As String

    ' --- Entschlüsseln einer Nachricht
    ' --- Hier wird die von Alice gesendete Nachricht
    entschlüsselt
    Dim inStream As FileStream
    Dim result As Byte()
    Dim buffer As Byte() = New Byte(4095) {}
    Dim des As New DESCryptoServiceProvider()
    Dim outStream As New MemoryStream()
    Dim readLength As Integer = 0
    Dim crypter As ICryptoTransform
    Dim decryptStream As CryptoStream

    ' --- Erzeugen der Streams
    inStream = New FileStream( _
        fnMessageEncrypted, FileMode.Open)
    outStream = New MemoryStream()

    crypter = des.CreateDecryptor( _
        SymmetricKey, InitVector)
    decryptStream = New CryptoStream( _
        inStream, crypter, CryptoStreamMode.Read)

    Do
        readLength = decryptStream.Read( _
            buffer, 0, 4096)
        outStream.Write(buffer, 0, readLength)
    Loop While readLength > 0
```

```
result = outputStream.ToArray()

inStream.Close()
outStream.Close()

Return ByteArrayToText(result)
End Function
```

*Listing 19.14: Laden des privaten Schlüssels und Entschlüsseln der Nachricht
[BobAndAlice.sln]*

Damit wäre die Funktionalität hergestellt. Die Methoden der Hauptfenster der Bob- und Alice-Anwendungen beschränken sich auf die richtigen Aufrufe in der richtigen Reihenfolge.

20 Namespace System.Reflection

Reflection bietet im .NET Framework die Möglichkeit, die Metadaten einer *Assembly* (EXE oder DLL) auszuwerten. Zu den Metadaten gehören unter anderem Informationen über verwendete Datentypen, über enthaltene Methoden einer Klasse oder deren Eigenschaften, verwendete Namespaces usw.

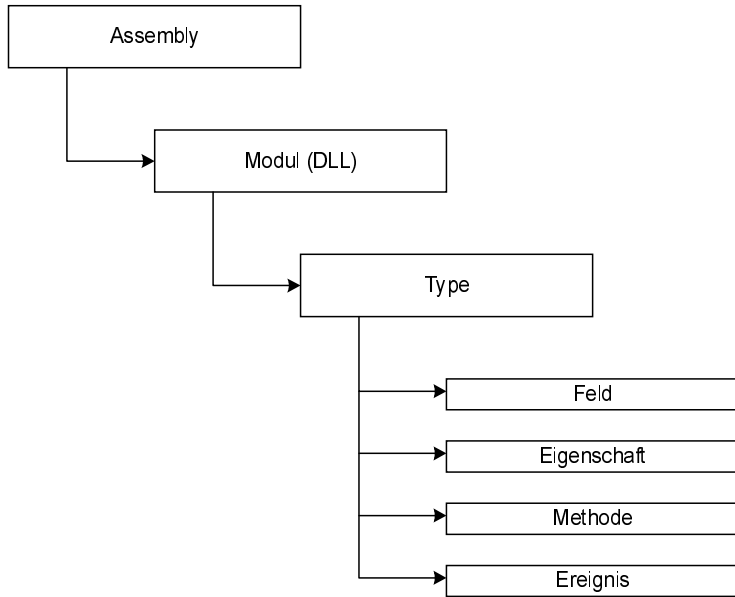
Reflection findet auch Einsatz in Visual Studio .NET selbst, z.B. bei der *IntelliSense*-Hilfefunktion. Hier werden über Reflection die aktuell wichtigen Daten über ein Objekt, eine Methode oder eine Klasse ermittelt. Aber das ist lange nicht alles, was Reflection bedeutet. Über die Klassen im *Namespace System.Reflection.Emit* ist es auch möglich, programmgesteuert ein Programm zu erstellen, dieses auf die Festplatte zu speichern oder auch gleich im Speicher auszuführen.

IntelliSense

20.1 Das Reflection-API

Das Reflection-API ist hierarchisch aufgebaut. An unterster Stelle stehen dabei die einzelnen Methoden, Felder oder Eigenschaften eines Datentyps. Darüber befindet sich der Typ selbst, dieser ist wiederum in einem Modul enthalten, das wiederum zu einer *Assembly* gehört. Eine Übersicht über das Reflection-API finden Sie in Abbildung 20.1.

Grundsätzlich können innerhalb eines Kapitels natürlich nicht alle Möglichkeiten angesprochen werden, die das Reflection-API bietet. Es wird genug sein, um Ihnen den Start zu ermöglichen; ein Vollständigkeitsanspruch kann hier nicht erhoben werden. Dennoch sind sich die Vorgehensweisen sehr ähnlich, so dass Sie schnell zu weiteren Ergebnissen kommen werden.



© Frank Eller

Abbildung 20.1: Übersicht über das Reflection-API

20.2 Die Klasse Type

Type Die Basis von Reflection liefert dabei der Datentyp, an den man am einfachsten herankommt: die Klasse `Type`. Obwohl sie im Namespace `System` deklariert ist, ist sie doch von `MemberInfo` abgeleitet, einem Datentyp, der im Namespace `System.Reflection` deklariert ist und die Basis für Informationen über Felder, Eigenschaften oder Methoden darstellt.

GetType() Ein Objekt vom Typ `Type` können Sie auf zwei Arten erhalten: Einmal über die Methode `GetType()`, die von `Object` implementiert wird und somit auch in jedem Datentyp enthalten ist. Dann über die statische (shared) Methode `GetType()` des Datentyps `Type`, der der voll qualifizierte Name eines Datentyps übergeben wird und die dann ebenfalls ein Objekt vom Typ `Type` zurückliefert.

typeof Wenn Sie mit C# arbeiten, gibt es auch noch den Operator `typeof`, der ebenfalls ein Objekt vom Typ `Type` zurückliefert, allerdings eine ganz andere Bedeutung hat als `TypeOf` in Visual Basic.

Wollen Sie z.B. mit dem Datentyp `Integer` arbeiten und alles darüber herausfinden, so können Sie folgendermaßen vorgehen:

```
Dim i As Integer
Dim typ As Type = i.GetType()
```

Wahlweise mit C# auch:

```
Type typ = TypeOf(int);
```

Über die statische Methode `GetType()` von `Type` gestaltet sich das Ganze ein wenig anders. Hier benötigen Sie den voll qualifizierten Namen des Datentyps, also den Namen inklusive der Bezeichnung des Namespaces, in dem er deklariert ist. `Integer` ist lediglich ein Alias für den .NET-Datentyp `Int32`, der in `System` deklariert ist. Damit erhalten Sie das gleiche Ergebnis wie mit den obigen Zeilen auch mit folgender Codezeile:

```
Dim typ As Type = Type.GetType("System.Int32")
```

20.2.1 Basisinformationen ermitteln

Sobald der Datentyp einmal ermittelt ist, können Sie mit Hilfe diverser Eigenschaften und Methoden mehr über den Datentyp herausfinden. In Tabelle 20.1 finden Sie einige der wichtigsten bzw. interessantesten Eigenschaften des Datentyps `Type`.

Eigenschaft	Bedeutung
<code>Assembly</code>	Liefert die kompletten Daten der <code>Assembly</code> , in der der Datentyp deklariert ist
<code>AssemblyQualifiedName</code>	Liefert den vollen Namen der <code>Assembly</code> , in der der Datentyp deklariert ist
<code>BaseType</code>	Liefert den Basistyp des angegebenen Datentyps
<code>FullName</code>	Liefert den vollen Namen des Datentyps (inkl. dem Namespace)
<code>GUID</code>	Liefert den GUID des Datentyps
<code>HasElementType</code>	Liefert <code>True</code> , wenn der Datentyp Elemente enthalten kann, z. B. im Falle eines Arrays
<code>IsAbstract</code>	Liefert <code>True</code> , wenn es sich um einen abstrakten Datentyp (<code>Must Inherit</code>) handelt
<code>IsAnsiClass</code>	Liefert <code>True</code> , wenn das Meta-Attribut » <code>AnsiClass</code> « für diesen Datentyp gesetzt ist

Tabelle 20.1: Einige interessante Eigenschaften von `Type`

Eigenschaft	Bedeutung
IsArray	Liefert True, wenn es sich bei dem Datentyp um ein Array handelt
IsAutoClass	Liefert True, wenn das Meta-Attribut »AutoClass« für diesen Datentyp gesetzt ist
IsAutoLayout	Liefert True, wenn das Meta-Attribut »AutoLayout« für diesen Datentyp gesetzt ist
IsByRef	Liefert True, wenn der Datentyp als Referenz übergeben wurde (ByRef in Visual Basic .NET, ref in C#)
IsClass	Liefert True, wenn es sich bei dem Datentyp um eine Klasse handelt
IsCOMObject	Liefert True, wenn es sich bei dem Datentyp um ein COM-Objekt handelt
IsEnum	Liefert True, wenn es sich bei dem Datentyp um einen Aufzählungstyp (Enum) handelt
IsImport	Liefert True, wenn der Datentyp von einer anderen Klasse importiert wurde
IsInterface	Liefert True, wenn es sich bei dem Datentyp um ein Interface handelt
IsPointer	Liefert True, wenn es sich bei dem Datentyp um einen Zeiger handelt
IsPrimitive	Liefert True, wenn es sich bei dem Datentyp um einen der primitiven Datentypen des .NET Frameworks handelt
IsPublic	Liefert True, wenn der Datentyp als öffentlich deklariert ist
IsSealed	Liefert True, wenn es sich um einen versiegelten (Not Inheritable) Datentyp handelt
IsSerializable	Liefert True, wenn das Meta-Attribut »Serializable« für diesen Datentyp gesetzt ist
IsValueType	Liefert True, wenn es sich bei dem Datentyp um einen Wertetyp handelt
Module	Liefert den Namen des Moduls (der DLL), in dem der Datentyp deklariert ist
Namespace	Liefert den Namen des Namespaces dieses Datentyps
UnderlyingSystemType	Liefert den Namen des Datentyps, mit dem dieser Datentyp in der CLR repräsentiert wird

Tabelle 20.1: Einige interessante Eigenschaften von Type (Fortsetzung)

Informationen über einen Typ herausfinden

Allein mit diesen Informationen ist es bereits möglich, ein kleines Beispielprogramm zu schreiben. Dazu wird zunächst einmal ein `Type`-Objekt benötigt, das einen beliebigen Datentyp repräsentiert. Im Beispiel wird der Datentyp angegeben und dann über diesen Namen ein `Type`-Objekt erzeugt. Dieses wird dann benutzt, um Genaueres über den Datentyp herauszufinden.

Listing 20.1 zeigt den relevanten Quellcode des Beispielprogramms. Der Name des Datentyps, dessen Informationen ermittelt werden sollen, wird über eine Textbox mit Namen `tbxType` ermittelt, in die der Benutzer den gewünschten Namen eingeben kann. Das Objekt vom Typ `Type`, das zur Auswertung benutzt wird, hat den Namen `infoType`.

Um dem Benutzer die Eingabe zu erleichtern, wird eine überladene Methode von `GetType()` benutzt, in der zusätzlich zum Namen des zu ermittelnden Datentyps auch noch zwei boolesche Parameter angegeben werden. Der erste gibt an, ob eine Exception ausgelöst werden soll, wenn der Datentyp nicht gefunden wird. Im Beispiel ist hier `False` angegeben. Falls der Datentyp nicht gefunden wird, enthält `infoType` den Wert `Nothing` (null in C#), der danach kontrolliert wird.

Der zweite Parameter gibt an, ob Groß-/Kleinschreibung kontrolliert werden soll. Der Wert `True` bedeutet hier, dass es egal ist, wie der Name des Datentyps eingegeben wird. Das macht es dem Benutzer später leichter, da er sich nicht an die korrekte Schreibweise halten muss.

```
'=====
' SimpleTypeInfo.sln
' Copyright 2002 Frank Eller
' Typinformationen mit Reflection auslesen
'=====

Public Class Form1
    Inherits System.Windows.Forms.Form

    #Region " Windows Form Designer generated code "

    'Der Code, der von VS.NET eingefügt wurde, wird
    'aus Übersichtsgründen nicht dargestellt.
```

```

#End Region

Private Sub btnInfo_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnInfo.Click

    If tbxType.Text = String.Empty Then
        Exit Sub
    End If

    ' --- Informationen herausfinden
    Dim infoType As Type = Type.GetType( _
        tbxType.Text, False, True)

    ' --- Wenn Typ nicht auffindbar, beenden
    If infoType Is Nothing Then
        MessageBox.Show( _
            "Dieser Typ ist nicht auffindbar", _
            "Nicht auffindbar", MessageBoxButtons.OK, _
            MessageBoxIcon.Information)

        Exit Sub
    End If

    ' --- Typinformationen anzeigen
    lblAssembly.Text = infoType.Assembly.ToString()
    lblModule.Text = infoType.Module.ToString()
    lblNamespace.Text = infoType.Namespace.ToString()

    ' --- Art des Typs auswerten
    If infoType.IsClass Then
        lblTyp.Text = "Klasse"
    ElseIf infoType.IsEnum Then
        lblTyp.Text = "Enumeration"
    ElseIf infoType.IsArray Then
        lblTyp.Text = "Array"
    ElseIf infoType.IsInterface Then
        lblTyp.Text = "Interface"
    ElseIf infoType.IsPrimitive Then
        lblTyp.Text = "Primitiver Typ"
    End If

    ' --- Weitere Infos auswerten
    lblMore.Text = ""
    If infoType.IsPublic Then
        lblMore.Text = lblMore.Text + "Public "
    End If
    If infoType.IsAbstract Then
        lblMore.Text = lblMore.Text + _

```

```
        "Not Inheritable "
    End If
    If infoType.IsSealed Then
        lblMore.Text = lblMore.Text + "Must Inherit "
    End If
    If infoType.IsImport Then
        lblMore.Text = lblMore.Text + "Importiert "
    End If
End Sub
Private Sub btnExit_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnExit.Click

    Me.Close()
End Sub
End Class
```

Listing 20.1: Informationen über einen Datentyp ermitteln [SimpleTypeInfo.sln]

Wie angesprochen müssen Sie bei diesem Beispiel daran denken, den kompletten Namen anzugeben. Wenn Sie also Informationen über den Datentyp `Integer` herausfinden wollen, müssen Sie den Namen `System.Int32` benutzen. Abbildung 20.2 zeigt die Ausgabe des Programms für den Datentyp `Type` selbst.

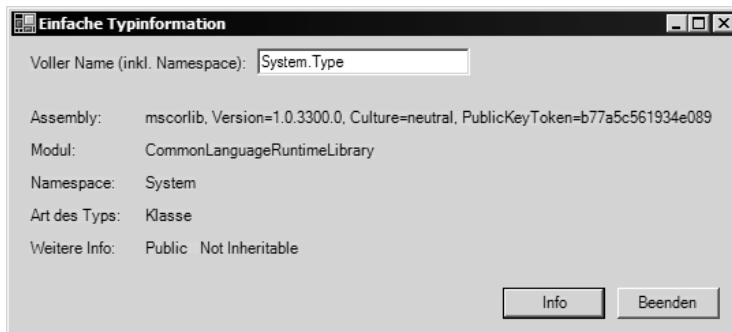


Abbildung 20.2: Typinformationen zu `System.Type` (SimpleTypeInfo.exe)

20.2.2 Detailinformationen ermitteln

Weitere Informationen können über die Methoden von `Type` ermittelt werden. Diese Informationen werden aber zum größten Teil wieder von anderen Klassen repräsentiert, die dann wiederum selbst den Zugriff auf weitere Details ermöglichen und alle im Namespace `System.Reflection` deklariert sind. Abbildung 20.3 zeigt die Vererbungshierarchie im Diagramm.

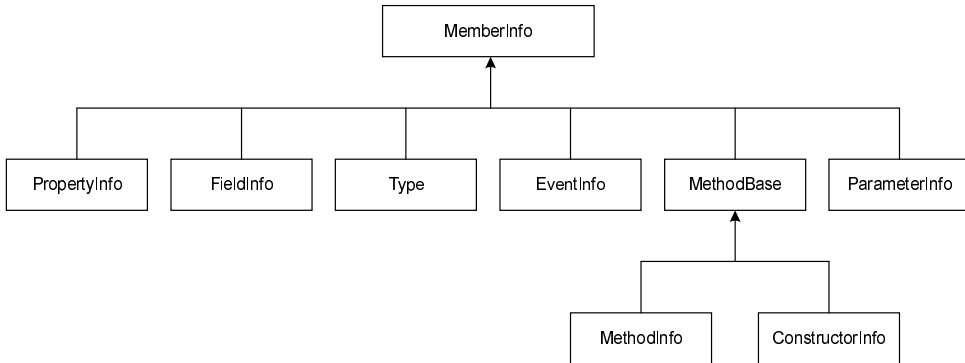


Abbildung 20.3: Die Vererbungshierarchie der Reflection-API

MemberInfo, ConstructorInfo und MethodInfo

Wie Sie sehen können, sind alle Klassen, die irgendetwas mit Informationen über einen Datentyp zu tun haben, von der Basisklasse `MemberInfo` abgeleitet. Im Falle von Konstruktoren und Methoden (`ConstructorInfo` und `MethodInfo`) liegt noch die Klasse `MethodBase` dazwischen, die speziell Informationen für Methoden und Konstruktoren liefern kann, während `MemberInfo` ein wenig allgemeiner ist.

All diese Klassen implementieren eine Unmenge von Methoden, mit denen weitere Informationen ermittelt werden können. Im Zentrum dieser ganzen Informationsermittlung steht allerdings in den meisten Fällen der Datentyp `Type`. Abbildung 20.4 zeigt die Zusammenhänge zwischen den einzelnen Klassen.

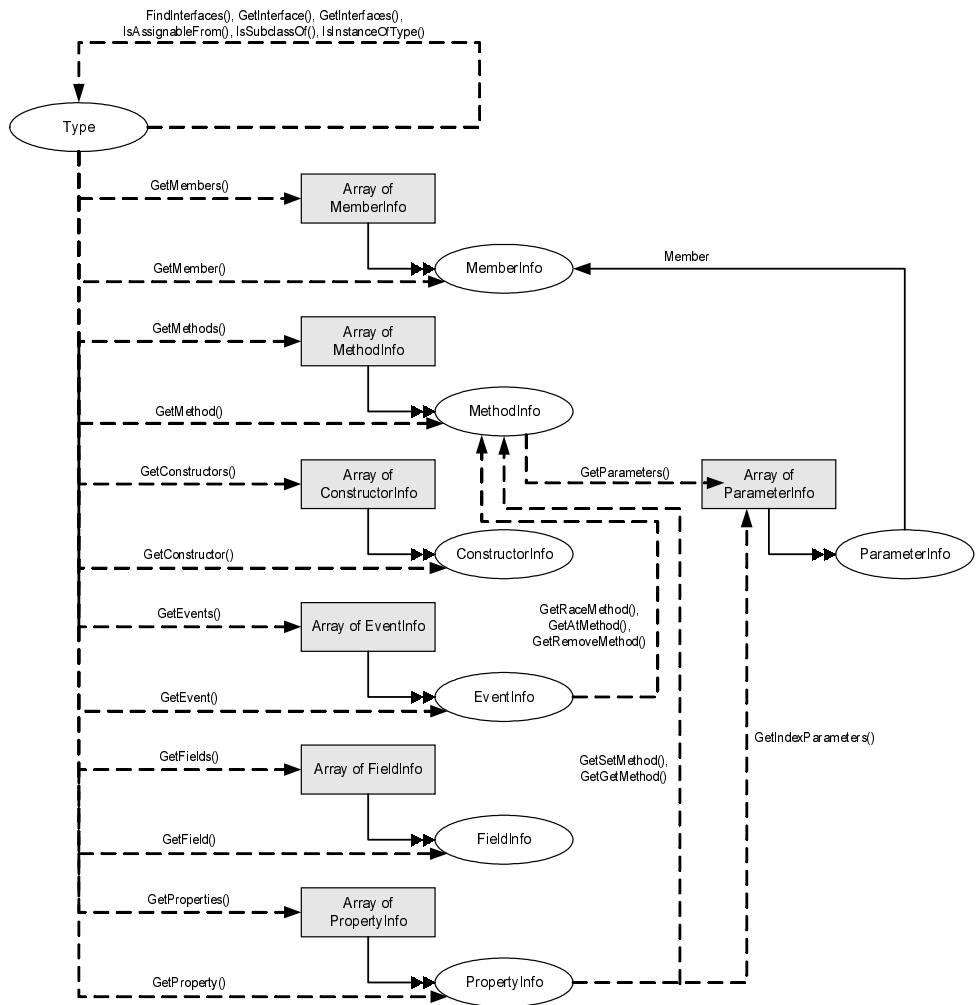


Abbildung 20.4: Übersicht über die Reflection-Klassen

Methoden eines Datentyps ermitteln

In diesem Beispiel geht es um die Methoden eines Datentyps. Wie zuvor ermitteln wir das `Type`-Objekt wieder über `Type.GetType()`. Die entsprechenden Methoden sind `GetMethod()` bzw. `GetMethods()`. Erstere liefert eine bestimmte Methode zurück, die spezifiziert werden muss, Letztere liefert alle Methoden, die in der Klasse deklariert sind.

GetMethod() und GetMethods()

Die zurückgelieferten Objekte sind vom Typ `MethodInfo`. `MethodInfo` ist von `MemberInfo` abgeleitet, daher ist es möglich, detailliertere Informationen zu erhalten. Zunächst aber eine einfache Variante, die lediglich die einzelnen Methoden samt Parametern auflistet.

Die Vorgehensweise ist einfach. Zunächst werden alle enthaltenen Methoden eines Datentyps ermittelt. Dann werden mit Hilfe einer Methode namens `EvaluateMethod()`, der ein Objekt vom Typ `Method` `Info` übergeben wird, die detaillierten Daten in eine für den Programmierer lesbare Form gebracht.

```
'=====
' MethodInfoExample.sin
' Copyright 2002 Frank Eller
' Typinformationen mit Reflection auslesen
'=====

Imports System.Reflection

Public Class Form1
    Inherits System.Windows.Forms.Form

    #Region " Windows Form Designer generated code "

        'Der Code, der von VS.NET eingefügt wurde, wird
        'aus Übersichtsgründen nicht dargestellt.

    #End Region

    Private Function EvaluateMethod( _
        ByVal infoMethod As MethodInfo) As String

        ' --- Deklarationen
        Dim result As String = ""
        Dim resValue As String = "As "
        Dim paramInfos() As ParameterInfo
        Dim i As Integer
        Dim isFunction As Boolean

        ' --- Sichtbarkeitslevel
        If infoMethod.IsPublic Then
            result = result + "Public "
        End If
        If infoMethod.IsPrivate Then
            result = result + "Private "
        End If
        If infoMethod.IsStatic Then
            result = result + "Shared "
        End If
```

```
' --- Rückgabewert
If infoMethod.ReturnType Is Nothing Then
    result = result + "Sub "
    isFunction = False
Else
    result = result + "Function "
    resValue = resValue + _
        infoMethod.ReturnType.Name.ToString()
    isFunction = True
End If

result = result + infoMethod.Name + "("

paramInfos = infoMethod.GetParameters()

If Not paramInfos Is Nothing Then
    For i = 0 To paramInfos.Length - 1
        result = result + paramInfos(i).Name + " As "
        result = result + _
            paramInfos(i).ParameterType.Name
        If (i + 1) < paramInfos.Length Then
            result = result + ", "
        End If
    Next
End If
result = result + ")"
If isFunction Then
    result = result + resValue
End If

' --- Zurückliefern
Return result
End Function

Private Sub btnInfo_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnInfo.Click

' --- Deklarationen
Dim infoType As Type
Dim methods() As MethodInfo
Dim method As MethodInfo

' --- Wenn Textbox leer, keine Aktion
If tbxType.Text = String.Empty Then
    Exit Sub
End If

' --- Typ ermitteln
infoType = Type.GetType(tbxType.Text, _
```

```

        False, True)
    If infoType Is Nothing Then
        Exit Sub
    End If

    ' --- Informationen ermitteln
    methods = infoType.GetMethods()

    For Each method In methods
        tbxInfo.AppendText(EvaluateMethod(method))
        tbxInfo.AppendText(ControlChars.CrLf)
    Next
End Sub

Private Sub btnExit_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnExit.Click

    Me.Close()
End Sub
End Class

```

Listing 20.2: Informationen über Methoden ermitteln [MethodInfoExample.sln]

Listing 20.2 zeigt die beiden relevanten Methoden, die die eigentliche Arbeit erledigen. In `btnInfo_Click()` werden zunächst alle Methoden eines Datentyps ermittelt. Dann wird jede Methode an `EvaluateMethod()` übergeben, wo weitere Informationen ausgelesen und in eine (in diesem Fall für VB-Programmierer) verständliche Form gebracht werden. Abbildung 20.5 zeigt einen Screenshot des laufenden Programms.

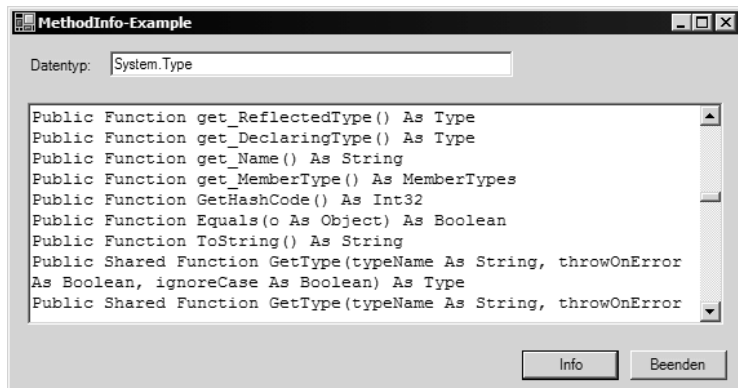


Abbildung 20.5: Die Methoden der Klasse Type [MethodInfoExample.exe]

Ebenso könnte man auch vorgehen, wenn man Eigenschaften (Properties), Ereignisse oder Interfaces auslesen wollte. Eine andere Möglichkeit, an die Daten heranzukommen, bieten die Methoden `GetMembers()` und `FindMembers()`. Beide funktionieren als eine Art Universallösung, wobei `FindMembers()` noch die Angabe eines Filters ermöglicht, d.h. man kann festlegen, welche Art von Element (Methode, Eigenschaft, Feld usw.) man auslesen will. `GetMembers()` liefert ein Array mit allen Elementen des Datentyps zurück und überlässt Ihnen danach die Auswahl, was Sie damit tun wollen.

**GetMembers()
und
FindMembers()**

Über einen Parameter vom Typ `BindingFlags` kann bei `GetMembers()` angegeben werden, welche Art von Elementen zurückgeliefert werden soll. Dabei wird allerdings nicht der Typ des Elements unterschieden, sondern z.B. ob alle öffentlichen oder alle privaten Elemente zurückgeliefert werden sollen. `BindingFlags` implementiert ein `BitArray`, d.h. Sie können mehrere Optionen angeben und teilweise müssen Sie das sogar. Wenn Sie z.B. alle öffentlichen Elemente erhalten wollen, müssen Sie zusätzlich angeben, ob es sich um die statischen (`shared`) oder die Instanzelemente handeln soll. Tun Sie das nicht, wird auch nichts zurückgeliefert.

Members eines Datentyps über GetMembers() ermitteln

Im folgenden Beispiel werden wir alle öffentlichen Elemente zurückliefern und nach dem Typ des Elements sortieren. Dazu benötigen wir die Angabe von `BindingFlags` eigentlich überhaupt nicht, da es sich bei `GetMethods()` um eine überladene Methode handelt, die ohne Angabe eines Parameters ohnehin alle öffentlichen Methoden auflistet. Dem Verständnis zuliebe wird `BindingFlags` an dieser Stelle dennoch benutzt.

```
'=====
' GetMembersExample.sln
' Copyright 2002 Frank Eller
' Ermitteln von Member-Informationen
'=====
```

```
Imports System.Reflection
```

```
Public Class Form1
```

```
    Inherits System.Windows.Forms.Form
```

```
#Region " Windows Form Designer generated code "
```

*'Der Code, der von VS.NET eingefügt wurde, wird
'aus Übersichtsgründen nicht dargestellt.*

/#End Region

```
Private Sub btnInfo_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnInfo.Click

    ' --- Deklarationen
    Dim infoType As Type = _
        Type.GetType(tbxType.Text, False, True)
    Dim members() As MemberInfo
    Dim member As MemberInfo
    Dim theFields(0) As String
    Dim theMethods(0) As String
    Dim theConstructors(0) As String
    Dim theProperties(0) As String
    Dim helpString As String

    ' --- Wenn nicht erfolgreich, beenden
    If infoType Is Nothing Then
        Exit Sub
    End If

    ' --- Arrays vorbelegen
    theFields(0) = "Felder:"
    theMethods(0) = "Methoden:"
    theConstructors(0) = "Konstruktoren:"
    theProperties(0) = "Eigenschaften:"

    ' --- Alle öffentlichen Members holen
    ' --- Hier würde auch GetMembers() genügen
    members = infoType.GetMembers( _
        (BindingFlags.Public Or BindingFlags.Instance))

    For Each member In members
        If member.MemberType = MemberTypes.Field Then
            ReDim Preserve theFields( _
                theFields.GetLength(0))
            theFields(theFields.GetUpperBound(0)) = _
                " --> " + member.Name
        End If

        If member.MemberType = _
            MemberTypes.Constructor Then
            ReDim Preserve theConstructors( _
                theConstructors.GetLength(0))
            theConstructors( _
                theConstructors.GetUpperBound(0)) = _
                " --> " + member.Name
        End If
    Next
```

```
End If

If member.MemberType = _
    MemberTypes.Property Then
    ReDim Preserve theProperties( _
        theProperties.GetLength(0))
    theProperties( _
        theProperties.GetUpperBound(0)) = _
        " --> " + member.Name
End If

If member.MemberType = MemberTypes.Method Then
    ReDim Preserve theMethods( _
        theMethods.GetLength(0))
    theMethods( _
        theMethods.GetUpperBound(0)) = _
        " --> " + member.Name
End If
Next

' --- Ausgeben des Ganzen
lboxResult.Items.Clear()
For Each helpString In theConstructors
    lboxResult.Items.Add(helpString)
Next

For Each helpString In theFields
    lboxResult.Items.Add(helpString)
Next

For Each helpString In theProperties
    lboxResult.Items.Add(helpString)
Next

For Each helpString In theMethods
    lboxResult.Items.Add(helpString)
Next
End Sub

Private Sub btnExit_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnExit.Click
    Me.Close()
End Sub
End Class
```

Listing 20.3: GetMembersExample.exe

In diesem Fall werden die ermittelten Informationen in Arrays abgelegt. Die Arrays werden dann immer wieder redimensioniert und eine neue Information (alles Strings) wird angehängt. Zum Schluss werden alle Informationen in beliebiger Reihenfolge in einer Listbox ausgegeben, die den Namen `lbtResult` trägt. Abbildung 20.6 zeigt einen Screenshot des Programms.

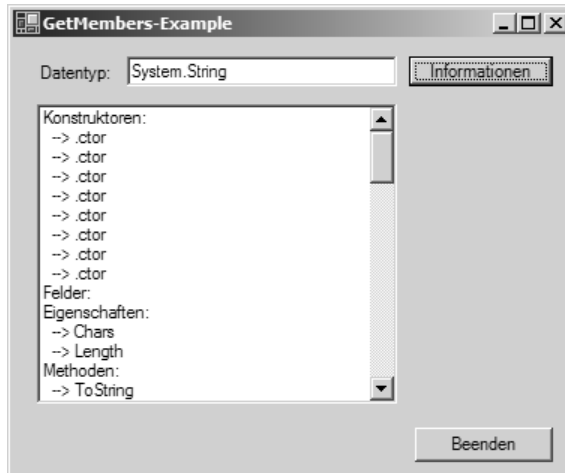


Abbildung 20.6: Members der Klasse `String` [`GetMembersExample.sln`]

20.3 Die Klassen Assembly und Module

20.3.1 Informationen über Assemblies ermitteln

Informationen über Datentypen reichen oftmals nicht aus. Beispielsweise wäre es doch sinnvoll, von einem Datentyp auf die DLL zu kommen, in der er deklariert ist, und von dort weitere Informationen zu erhalten. Weiter oben steht dann die `Assembly`, die mehrere DLLs enthalten kann und wiederum weitere Informationen bietet.

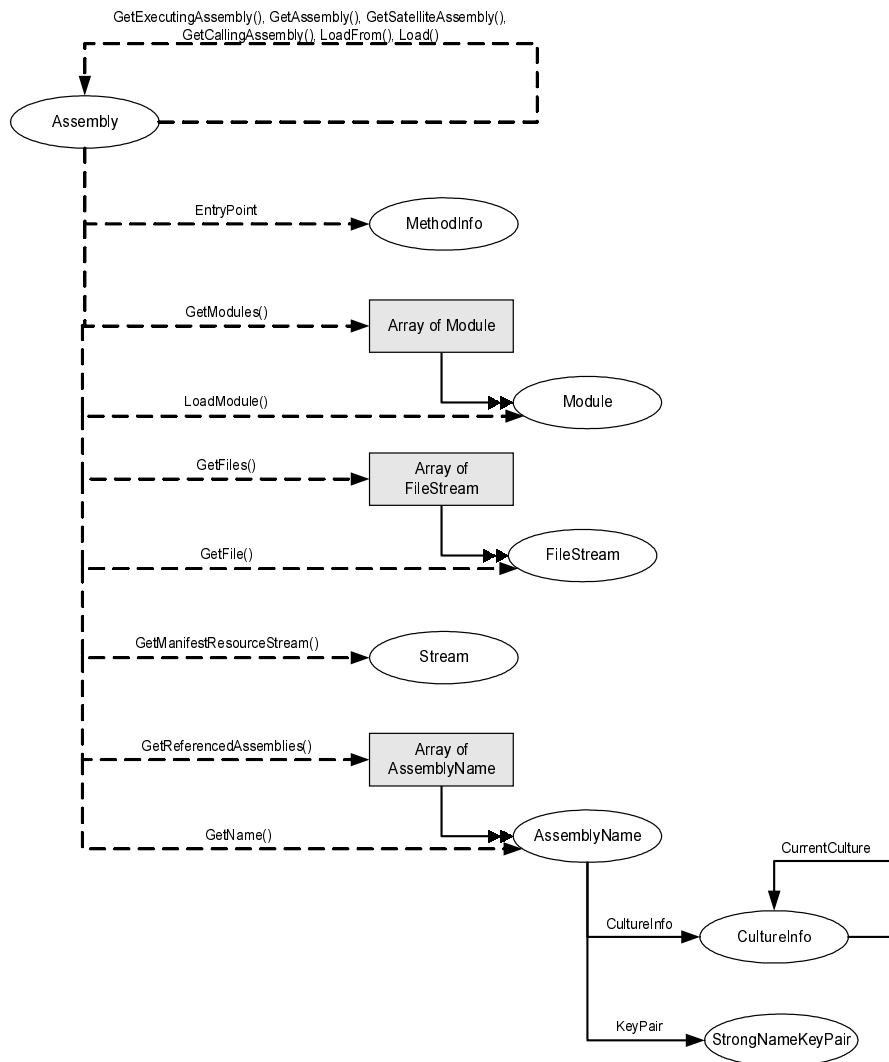


Abbildung 20.7: Die Zusammenhänge für die Klasse Application

Die Klasse `Assembly` bietet eine Menge Möglichkeiten, mit deren Hilfe man an weitere Informationen kommen kann. Beispielsweise könnte man die Daten aller Assemblies angeben, die standardmäßig im System enthalten sind. Dazu benötigt man natürlich den Pfad zum .NET Framework. Das einzige Problem dabei ist die Versionsnummer, die sich ändern könnte.

Assembly

Location Mit Hilfe der Klasse `Assembly` ist es jedoch möglich, auch diesen Pfad zu ermitteln. Die Eigenschaft `Location` liefert den Pfad einer `Assembly`, und über die Klasse `Type` (bzw. ein Objekt vom Typ `Type`) kommt man auch an die `Assembly` heran. Wir nehmen also einfach einen Basis-Datentyp, auf den wir ohnehin zugreifen können, und ermitteln dann den Pfad der `Assembly`-Dateien.

Die Zusammenhänge werden wieder anhand eines Diagramms dargestellt. Abbildung 20.7 zeigt das Objektmodell für die Klasse `Assembly`.

Assembly-Pfad ermitteln

Das folgende kleine Konsolenprogramm liefert die Pfadangabe zum .NET Framework-Installationsverzeichnis. Dazu wird der Datentyp `Integer` benutzt (bzw. `System.Int32`). Die `Assembly`, in der dieser Datentyp deklariert ist, ist die `mscorlib.dll`. Da nur der Pfad ermittelt werden soll, wird die statische Methode `GetDirectory()` der Klasse `Path` verwendet, um die reine Pfadinformation zu ermitteln. Listing 20.4 zeigt das komplette kleine Programm.

```
'=====
' GetAssemblyPath.sln
' Copyright 2002 Frank Eller
' Liefert den Pfad zu einer Assembly
'=====

Imports System.Reflection
Imports System.IO

Module Module1

    Sub Main()
        ' --- Deklarationen
        Dim baseAssembly As [Assembly]
        Dim typ As Type = Type.GetType( _
            "System.Int32", False, True)
        Dim pathString As String

        ' --- Pfad ermitteln
        baseAssembly = [Assembly].GetAssembly(typ)
        pathString = baseAssembly.Location
        pathString = Path.GetDirectoryName(pathString)

        Console.WriteLine(pathString)
    End Sub

End Module
```

Listing 20.4: Pfad zu einer Assembly ermitteln [GetAssemblyPath.sln]

Selbstverständlich funktioniert das mit allen anderen Programmen ebenfalls, denn eine `Assembly` ist ja die kleinste Verpackungseinheit unter .NET. Sobald der Pfad einmal ermittelt ist, kann auch auf die anderen in diesem Pfad enthaltenen Assemblies zugegriffen werden.

Wenn wir eine `Assembly` haben, können wir auch auf die darin enthaltenen Informationen zugreifen. Über die Methode `GetModules()` ist es beispielsweise möglich, auf alle Module (sprich: DLLs) zuzugreifen, die in der `Assembly` enthalten sind, `GetTypes()` liefert alle in der `Assembly` enthaltenen Typen. Auch hier ist die Anzahl der enthaltenen Methoden sehr groß, weshalb auf die Online-Hilfe verwiesen wird und nur einige der vielfältigen Möglichkeiten vorgestellt werden.

**GetModules() und
GetTypes()**

Typen auflisten

In diesem Beispiel werden die Module (also die DLLs) und die Klassen, die in der `mscorlib`-`Assembly` enthalten sind, aufgelistet. Dabei handelt es sich um die Basisklassen von .NET, d. h. nicht um alle Klassen, die das .NET Framework zur Verfügung stellt. Um die Module aufzulisten, müssen wir auf eine weitere Klasse des `Reflection`-Namespaces zurückgreifen, nämlich die Klasse `Module`. `Module` sind die DLLs selbst, die zusammengekommen eine `Assembly` bilden, oder einfacher: Eine `Assembly` kann mehrere DLLs beinhalten, muss es aber nicht.

Module

Der Quellcode benutzt wieder den Datentyp `System.Int32`, um an die `Assembly` zu kommen, und listet dann alle Typen und die Module auf, in denen diese Typen deklariert sind.

```
'=====
' GetMscorlibModules.sln
' Copyright 2002 Frank Eller
' Liefert alle Module der mscorlib
'=====

Imports System.Reflection
Imports System.Collections

Public Class Form1
    Inherits System.Windows.Forms.Form

    #Region " Windows Form Designer generated code "

    'Der Code, der von VS.NET eingefügt wurde, wird
    'aus Übersichtsgründen nicht dargestellt.
```

```

#End Region

Private Sub btnInfo_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnInfo.Click

    ' --- Deklarationen
    Dim baseAssembly As [Assembly]
    Dim typ As Type = Type.GetType("System.Int32")
    Dim coreModules() As [Module]
    Dim coreTypes() As Type
    Dim oneModule As [Module]
    Dim oneType As Type

    ' --- Assembly holen
    baseAssembly = [Assembly].GetAssembly(typ)

    ' --- Module holen
    coreModules = baseAssembly.GetModules()
    coreTypes = baseAssembly.GetTypes()

    ' --- Anzeigen
    For Each oneModule In coreModules
        lbxModules.Items.Add(oneModule.Name)
    Next

    For Each oneType In coreTypes
        lbxTypes.Items.Add(oneType.FullName)
    Next
End Sub

Private Sub btnExit_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnExit.Click

    Me.Close()
End Sub

Private Sub Form1_Load( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles MyBase.Load

    lbxModules.Items.Clear()
    lbxTypes.Items.Clear()
End Sub
End Class

```

Listing 20.5: Module ermitteln [GetMscorlib.Modules.sln]

Einen Screenshot des laufenden Programms zeigt Abbildung 20.8.

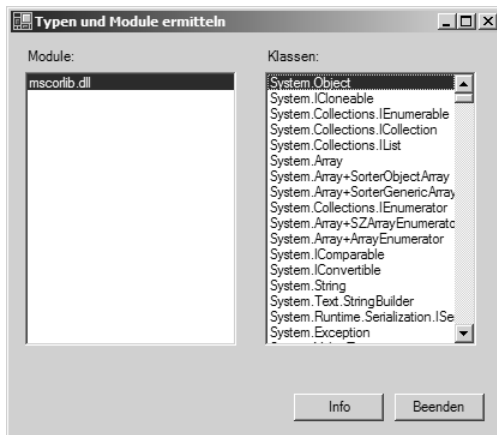


Abbildung 20.8: Alle Objekte aus mscorlib [*GetMscorlibModules.sln*]

20.3.2 Namespaces ermitteln

Bisher wurde besprochen, wie man Zugriff auf Klassen, die enthaltenen Elemente, Module und Assemblies erhalten kann. Es fehlt noch ein wichtiger Bestandteil von .NET, nämlich die Namespaces, für die auch keine eigene Klasse deklariert ist. Wie alles hat auch das seinen Grund.

Namespaces

Namespaces sind ein eigentlich virtuelles Mittel, um Programmcode zu strukturieren. Ein Namespace kann sich über mehrere Dateien erstrecken oder über mehrere Module, sogar über mehrere Assemblies. Weiterhin kann ein Namespace weitere Namespaces enthalten. Allein die Verwendung des Namespaces `System.Windows.Forms` macht das ersichtlich.

Wie also kann man an die Namespaces herankommen? Hier spielen die verschiedenen Klassen des .NET Frameworks zusammen. Zunächst müssen die Assemblies geladen und die enthaltenen Module ermittelt werden. Aus diesen Modulen heraus können dann alle Datentypen ausgelesen werden, die wiederum über die Eigenschaft `Namespace` den Zugriff auf Namespaces ermöglichen. Das Ganze hört sich etwas kompliziert an, aber es ist leider der einzige Weg.

Namespaces auflisten

Für dieses Beispiel wurden die Assembly-DLLs in einem Array vom Typ `String` zusammengefasst. Die Hauptmethode ist `GetNamespaces()`, die eine `ArrayList` zurückliefert, in der alle Namespaces enthalten sind.

Über einen beliebigen Datentyp wird eine `Assembly` ermittelt, deren Eigenschaft `Location` den Pfad zu den übrigen Assemblies liefert. Dieser wird den im Array `CoreModuleNames` enthaltenen Dateinamen hinzugefügt. Jetzt können die einzelnen Module und Assemblies geladen werden.



Nach Aussage von Microsoft ist es derzeit nicht möglich, per Programmcode auf den Global Assembly Cache (GAC) zuzugreifen. Natürlich könnte man das Windows-Verzeichnis verwenden und dann auf das darunter liegende Verzeichnis `Assembly` zugreifen. Es gibt allerdings keine Garantie dafür, dass Microsoft den internen Aufbau oder die Position des GACs in Zukunft nicht ändert. Ein anderer Weg ist allerdings derzeit nicht bekannt, und dass sich der Name dieses Verzeichnisses nicht ändert, ist auch nicht sicher. Das ist der Grund dafür, dass die Namen der Assemblies hier fest in das Programm kodiert wurden.

LoadFrom() Über die Methode `LoadFrom()` der Klasse `Assembly` werden die Dateien geladen und dann direkt mittels `GetModules` die enthaltenen Module ermittelt. Über `GetTypes()` werden dann die enthaltenen Datentypen ausgelesen und deren Namespaces in eine `ArrayList` geschrieben. Zur Sicherstellung, dass ein Namespace nicht doppelt eingetragen wird, dient die Methode `Contains()` der Klasse `ArrayList`. Wie der Name bereits vermuten lässt, dient diese Methode dazu, zu kontrollieren, ob ein Element bereits in der `ArrayList` enthalten ist.

Zum Schluss wird die Liste noch sortiert und dann zurückgeliefert. Die aufrufende Methode tut dann nichts anderes, als die Elemente dieser Liste einer `Listbox` hinzuzufügen. Der Aufbau des Formulars besteht entsprechend lediglich in dieser `Listbox` mit Namen `lbnNamespaces` und einem Button, mit dem die Auswertung gestartet wird. Die Namen der relevanten Assemblies (bzw. der DLLs, die die Datentypen enthalten) wurden in diesem Fall direkt in ein Array geschrieben und nicht dynamisch ermittelt. Listing 20.6 zeigt den kompletten Programmcode.

```

'=====
' GetNamespaces.sln
' Copyright 2002 Frank Eller
' Liefert alle Namespaces der FCL
'=====

Imports System.Reflection
Imports System.IO

Public Class Form1
    Inherits System.Windows.Forms.Form

#Region " Windows Form Designer generated code "

    ' Code der Übersichtlichkeit halber weggelassen

#End Region

Private CoreModuleNames() As String = _
    New String() _
    {"mscorlib.dll", _
    "System.dll", _
    "System.Windows.Forms.dll", _
    "System.Drawing.dll", _
    "System.Data.dll", _
    "System.Xml.dll", _
    "System.DirectoryServices.dll", _
    "System.Web.dll"}

Private Function GetNamespaces() As ArrayList

    ' --- Deklarationen
    Dim namespaceList As New ArrayList()
    Dim i As Integer
    Dim currAssembly As [Assembly]
    Dim AssemblyPath As String

    ' --- Pfad aller Assemblies ermitteln
    currAssembly = _
        Type.GetType("System.Int32").Assembly
    AssemblyPath = _
        Path.GetDirectoryName(currAssembly.Location) _
        + "\"

    ' --- CoreModuleNames Pfad zuweisen
    For i = 0 To CoreModuleNames.Length - 1
        CoreModuleNames(i) = AssemblyPath + _
            CoreModuleNames(i)
    Next

```

```

' --- Module und Typen ermitteln
For i = 0 To CoreModuleNames.Length - 1
    Dim coreModules() As [Module] = _
        [Assembly].LoadFrom( _
            CoreModuleNames(i)).GetModules()
    Dim coreClasses() As Type = _
        coreModules(0).GetTypes()

    Dim u As Integer

    For u = 0 To coreClasses.Length - 1
        If Not coreClasses(u).Namespace _
            Is Nothing Then
            If Not namespaceList.Contains( _
                coreClasses(u).Namespace) Then
                namespaceList.Add( _
                    coreClasses(u).Namespace)
            End If
        End If
    Next
Next

' --- Sortieren und zurückliefern
namespaceList.Sort()
Return namespaceList

End Function

Private Sub btnLoad_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnLoad.Click

    lbxNamespaces.Items.AddRange( _
        GetNamespaces().ToArray())

End Sub

Private Sub btnExit_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnExit.Click

    Me.Close()

End Sub

End Class

```

Listing 20.6: GetNamespaces.exe

Eigentlich sind jetzt alle notwendigen Funktionen enthalten, um so etwas wie einen ClassBrowser zusammenzustellen. Es fehlt noch eine Methode, um entsprechend einem Namespace die enthaltenen Typen aufzulisten. Diese Methode besteht allerdings nur in einem Vergleich der aktuellen Namespace-Bezeichnung mit den Namespaces der Datentypen, die ja ohnehin alle vorhanden sind. Abbildung 20.9 zeigt das Pogramm zur Laufzeit.

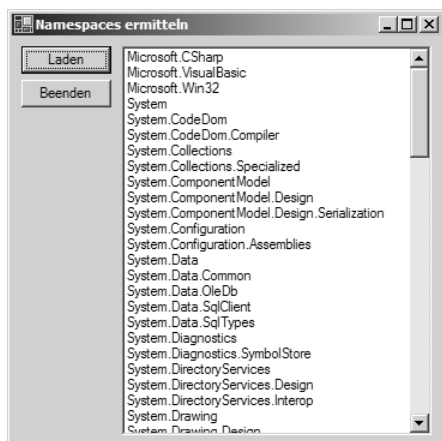


Abbildung 20.9: Ermittlung aller Namespaces [GetNamespaces.sln]

Datentypen anhand der Namespaces auflisten

Zur Ermittlung der Namespaces dient die gleiche Methode wie im vorherigen Beispiel. In einer weiteren Methode, mit der die Typen aus einem bestimmten Namespace ermittelt werden sollen, wird genauso vorgegangen, es werden also wieder alle Klassen aller Module geladen, diesmal aber mit dem Namespace verglichen, dessen enthaltene Typen aufgelistet werden sollen. Das Ergebnis wird in eine zweite Listbox geschrieben.

Der Quellcode sollte eigentlich selbsterklärend sein. Das Füllen der zweiten Listbox mit Namen `lbcClasses` wird durch die Auswahl eines Namespaces aus der ersten Listbox angestoßen. In diesem Beispiel wird allerdings nicht zwischen Klassen, Interfaces oder Enums unterschieden, was ebenfalls noch implementiert werden könnte. Zusammen mit den übrigen Beispielen dieses Kapitels kann dann ein recht komfortabler ClassBrowser erstellt werden.

```

'=====
' GetTypesFromNamespace.sln
' Copyright 2002 Frank Eller
' Liefert die Typen in einem Namespace
'=====

Imports System.Reflection
Imports System.IO

Public Class Form1
    Inherits System.Windows.Forms.Form

    #Region " Windows Form Designer generated code "

    ' Code der Übersichtlichkeit halber weggelassen

    #End Region

    Private CoreModuleNames() As String = _
        New String() {"mscorlib.dll", _
            "System.dll", _
            "System.Windows.Forms.dll", _
            "System.Drawing.dll", _
            "System.Data.dll", _
            "System.Xml.dll", _
            "System.DirectoryServices.dll", _
            "System.Web.dll"}

    Private Function GetNamespaces() As ArrayList

        ' --- Deklarationen
        Dim namespaceList As New ArrayList()
        Dim i As Integer
        Dim currAssembly As [Assembly]
        Dim AssemblyPath As String

        ' --- Pfad aller Assemblies ermitteln
        currAssembly = _
            Type.GetType("System.Int32").Assembly
        AssemblyPath = _
            Path.GetDirectoryName(currAssembly.Location) + _
            + "\"

        ' --- CoreModuleNames Pfad zuweisen
        For i = 0 To CoreModuleNames.Length - 1
            CoreModuleNames(i) = AssemblyPath + _
                CoreModuleNames(i)
        Next

        ' --- Module und Typen ermitteln
        For i = 0 To CoreModuleNames.Length - 1

```

```

Dim coreModules() As [Module] = _
    [Assembly].LoadFrom( _
        CoreModuleNames(i)).GetModules()
Dim coreClasses() As Type = _
    coreModules(0).GetTypes()

Dim u As Integer

For u = 0 To CoreClasses.Length - 1
    If Not CoreClasses(u).Namespace = _
        Is Nothing Then
        If Not namespaceList.Contains( _
            CoreClasses(u).Namespace) Then
            namespaceList.Add( _
                CoreClasses(u).Namespace)
        End If
    End If
Next
Next

' --- Sortieren und zurückliefern
namespaceList.Sort()
Return namespaceList

End Function

Private Function GetTypeNames( _
    ByVal currNamespace As String) _
    As ArrayList

    ' --- Deklarationen
    Dim result As New ArrayList()
    Dim i As Integer
    Dim u As Integer

    For i = 0 To CoreModuleNames.Length - 1
        Dim coreModules() As [Module] = _
            [Assembly].LoadFrom( _
                CoreModuleNames(i)).GetModules()
        Dim coreClasses() As Type = _
            coreModules(0).GetTypes()

        For u = 0 To coreClasses.Length - 1
            If coreClasses(u).Namespace = _
                currNamespace Then
                result.Add(coreClasses(u).Name)
            End If
        Next
    Next
Next

```

```
        result.Sort()
        Return result

End Function

Private Sub btnLoad_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnLoad.Click

    lbxNamespaces.Items.AddRange( _
        GetNamespaces().ToArray())

End Sub

Private Sub lbxNamespaces_SelectedIndexChanged( _
    ByVal sender As Object, _
    ByVal e As System.EventArgs) _
    Handles lbxNamespaces.SelectedIndexChanged

    Dim currNamespace As String = _
        lbxNamespaces.SelectedItem
    lbxClasses.Items.Clear()
    lbxClasses.Items.AddRange( _
        GetTypeNames(currNamespace).ToArray())

End Sub

Private Sub btnExit_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnExit.Click

    Me.Close()
End Sub

End Class
```

Listing 20.7: Typen aus Namespaces auslesen [GetTypeFromNamespace.sln]

Abbildung 20.10 zeigt einen Screenshot des Programms.

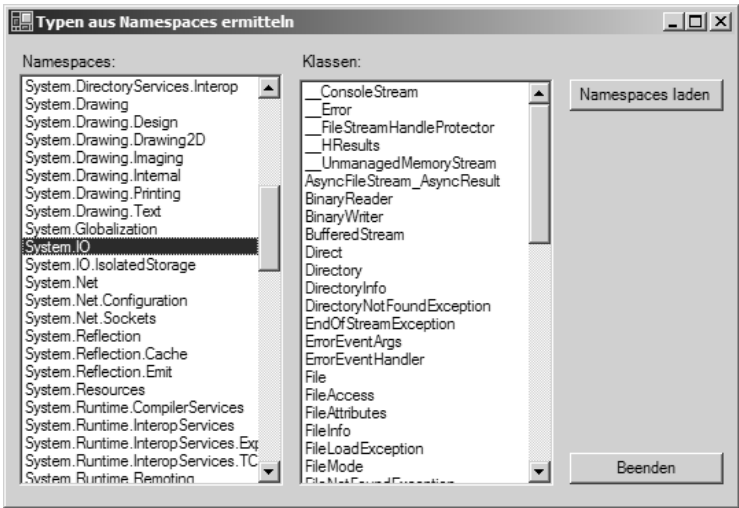


Abbildung 20.10: Namespaces und die enthaltenen Datentypen
[GetTypesFromNamespaces.sln]

20.4 Code on the fly mit Reflection.Emit

Die Auswertung bestehender Klassen oder Codebestandteile ist noch lange nicht alles, was Ihnen das .NET Framework an Möglichkeiten bietet. Eines der interessantesten Dinge ist das Erzeugen von Assemblies – also ausführbaren Dateien oder DLLs – »on the fly«, während des Programmlaufs. Diese Fähigkeit wird vor allem dann interessant, wenn Sie dem Anwender ermöglichen wollen, innerhalb Ihrer Applikation ein eigenes Programm zu schreiben und auszuführen.

Der Namespace `System.Reflection.Emit` liefert Ihnen die Klassen, Strukturen und Aufzählungstypen, die dies ermöglichen. Die wichtigsten Klassen des Namespaces sind in Tabelle 20.2 aufgelistet.

**System.
Reflection.Emit**

Klasse	Bedeutung
AssemblyBuilder	Repräsentiert eine dynamisch erzeugte Assembly
ConstructorBuilder	Repräsentiert einen dynamisch erzeugten Konstruktor

Tabelle 20.2: Wichtige Klassen von Reflection.Emit

Klasse	Bedeutung
EnumBuilder	Repräsentiert einen dynamisch erzeugten Aufzählungstyp
LocaleBuilder	Repräsentiert eine dynamisch erzeugte lokale Variable
MethodBuilder	Repräsentiert eine dynamisch erzeugte Methode einer Klasse
FieldBuilder	Repräsentiert ein dynamisch erzeugtes Feld einer Klasse
ModuleBuilder	Repräsentiert ein dynamisch erzeugtes Modul
EventBuilder	Repräsentiert ein dynamisch erzeugtes Ereignis
PropertyBuilder	Repräsentiert eine dynamisch erzeugte Eigenschaft
ParameterBuilder	Repräsentiert einen dynamisch erzeugten Parameter
TypeBuilder	Definiert und erzeugt dynamisch Instanzen von Klassen zur Laufzeit
ILGenerator	Erzeugt Intermediate-Language-Code
MethodRental	Ermöglicht das Vertauschen von Methodeninhalten zur Laufzeit

Tabelle 20.2: Wichtige Klassen von Reflection.Emit (Fortsetzung)



Wenn Sie dem Anwender wirklich ermöglichen wollen, selbst Code zu schreiben, benötigen Sie auch noch einen Parser, der die Eingaben des Benutzers auswertet und über entsprechende Anweisungen die Assembly bzw. das Programm erstellt. So weit wird in diesem Kapitel nicht gegangen, hier wird lediglich ein Programm erstellt, das lauffähig ist.

Dynamische Assembly erstellen

Das Erstellen einer dynamischen Assembly funktioniert grundsätzlich genauso wie die Erzeugung eines Programms. An erster Stelle steht die dynamische Assembly selbst, wobei es sich um eine herkömmliche Assembly handelt, die eine oder mehrere Dateien (Module) aufnehmen kann. Der Unterschied besteht darin, dass diese Assembly zunächst nur im Speicher des Rechners vorhanden ist.

Der nächste Schritt ist die Erstellung eines Moduls. Dieses Modul steht für eine ausführbare Datei oder eine DLL, die Klassen, Eigenschaften oder Methoden enthält. Danach werden innerhalb dieses Moduls die Datentypen deklariert und in diese wiederum Felder, Eigenschaften oder Methoden implementiert. Eine Übersicht darüber, welche Methoden verwendet werden, liefert Abbildung 20.11.

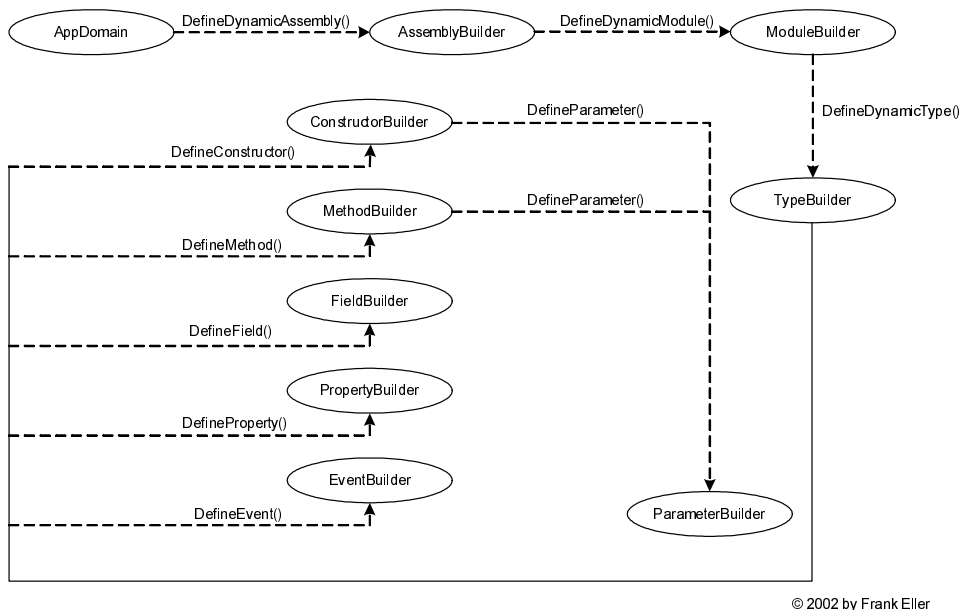


Abbildung 20.11: Zusammenhänge zwischen den Klassen von Reflection.Emit

Die hier gezeigten Methoden sind natürlich bei weitem nicht alle, denn würden alle Methoden eingefügt, wäre das Diagramm viel zu unübersichtlich, um es zu verstehen.

20.4.1 Erzeugen einer Assembly

Die Assembly erzeugen

Eine dynamische Assembly wird über die Methode `DefineDynamicAssembly()` eines `AppDomain`-Objekts erzeugt. Ein solches Objekt können Sie auf mehrere Arten erhalten:

- ▶ über die Eigenschaft `CurrentDomain()` der Klasse `AppDomain`
- ▶ über die Methode `CreateDomain()` der Klasse `AppDomain`
- ▶ über die Methode `GetDomain()` der Klasse `Thread` aus `System.Threading`

`DefineDynamicAssembly()` erwartet als Übergabeparameter einen Assembly-Namen. In diesem Fall allerdings nicht als `String`, sondern als Objekt vom Typ `AssemblyName`, das außer der Bezeichnung auch noch die Versionsnummer der Assembly enthält. Sie können also hier festlegen, welche Assembly-Version Sie erstellen möchten,

**DefineDynamic
Assembly()**

und diese ändern, wenn es eine neue Version des Programms gibt. Der folgende Code erzeugt eine `Assembly` mit dem Namen `DynAssembly` und der Versionsnummer 1.0.0.0:

```
Dim myDomain As AppDomain = AppDomain.CreateDomain("Domain1")
Dim aBuilder As AssemblyBuilder
Dim asblName As AssemblyName

asblName = New AssemblyName()
asblName.Name = "DynAssembly"
asblName.Version = New Version(1, 0, 0, 0)

aBuilder = myDomain.DefineDynamicAssembly( _
    asblName, _
    AssemblyBuilderAccess.Run)
```

AssemblyAccess und Assembly Builder

Der Parameter vom Typ `AssemblyAccess`, einem Aufzählungstyp, legt fest, wie sich die `Assembly` später verhalten soll. Der Wert `Run` bedeutet, dass diese `Assembly` lediglich im Speicher ausgeführt, aber nicht auf die Festplatte gespeichert werden kann. Ein Wert von `Save` würde bedeuten, dass die `Assembly` gespeichert, aber nicht ausgeführt werden kann, und der Wert `RunAndSave` bedeutet, dass beides möglich ist. Zurückgeliefert wird (wie am Code sichtbar) ein Objekt des Typs `AssemblyBuilder`.

Ein Modul erzeugen

`AssemblyBuilder` beinhaltet eine Methode `DefineDynamicModule()`, mit der ein Modul erzeugt werden kann. Wie auch bei einem herkömmlichen Programm ist das Modul der nächste Schritt, innerhalb des Moduls kommen dann die Methoden und Datentypen, die deklariert werden sollen.

DefineDynamic- Module()

`DefineDynamicModule()` erwartet als Parameter zumindest einen Namen für das neue Modul. Die Methode ist allerdings mehrfach überladen, so dass Sie auch festlegen können, wie der Dateiname heißen soll, in den das Modul gespeichert werden soll, oder ob Symboldaten mit in das Modul geschrieben werden sollen. Zurückgeliefert wird ein Objekt des Typs `ModuleBuilder`.

ModuleBuilder

Nachdem sowohl `Assembly` als auch Modul erzeugt sind, kann es an die eigentlichen Datentypen gehen. Die Klasse `ModuleBuilder` besitzt mehrere Methoden, mit denen Datentypen erzeugt werden können. In den meisten Fällen wird die Methode `DefineType()` ausreichen. Weitere Methoden sind z.B. `DefineEnum()`, `DefineGlobalMethod()` oder `DefineInitializedData()`.

20.4.2 Die Klasse `TypeBuilder`

Die Klasse `TypeBuilder` ist so etwas wie die Zentrale bei der Erzeugung von Datentypen, Methoden, Feldern oder Eigenschaften. Einige der Methoden der Klasse `TypeBuilder` sehen Sie in Tabelle 20.3.

Methode	Funktion
<code>AddInterfaceImplementation()</code>	Fügt ein Interface hinzu, das dieser Datentyp implementiert
<code>DefineConstructor()</code>	Fügt der Klasse einen neuen Konstruktor hinzu. Das zurückgelieferte Objekt ist vom Typ <code>ConstructorBuilder</code> .
<code>DefineDefaultConstructor()</code>	Fügt den Standard-Konstruktor hinzu.
<code>DefineEvent()</code>	Fügt der Klasse ein neues Ereignis hinzu. Das zurückgelieferte Objekt ist vom Typ <code>EventBuilder</code> .
<code>DefineField()</code>	Fügt der Klasse ein neues Feld hinzu. Das zurückgelieferte Objekt ist vom Typ <code>FieldBuilder</code> .
<code>DefineMethod()</code>	Fügt der Klasse eine neue Methode hinzu. Das zurückgelieferte Objekt ist vom Typ <code>MethodBuilder</code> .
<code>DefineProperty()</code>	Fügt der Klasse eine neue Eigenschaft hinzu. Das zurückgelieferte Objekt ist vom Typ <code>PropertyBuilder</code> .
<code>FindInterfaces()</code>	Liefert die implementierten Interfaces in einem Array aus <code>Type</code> -Objekten zurück
<code>FindMembers()</code>	Liefert die Elemente dieses Datentyps als Array aus <code>MemberInfo</code> -Objekten zurück
<code>GetConstructor()</code>	Liefert einen der Konstruktoren des Datentyps in Form eines <code>ConstructorInfo</code> -Objekts zurück
<code>GetConstructors()</code>	Liefert ein Array aus <code>ConstructorInfo</code> -Objekten zurück, das die Konstruktoren dieses Datentyps repräsentiert
<code>GetEvent()</code>	Liefert das angegebene Ereignis in Form eines <code>EventInfo</code> -Objekts zurück
<code>GetEvents()</code>	Liefert ein Array aus <code>EventInfo</code> -Objekten zurück, die die Ereignisse dieses Datentyps darstellen

Tabelle 20.3: Einige Methoden der Klasse `TypeBuilder`

Methode	Funktion
GetField()	Liefert ein bestimmtes Feld dieses Datentyps in Form eines <code>FieldInfo</code> -Objekts zurück
GetFields()	Liefert ein Array aus <code>FieldInfo</code> -Objekten zurück, die die Felder dieses Datentyps repräsentieren
GetMethod()	Liefert die angegebene Methode in Form eines <code>MethodInfo</code> -Objekts zurück
GetMethods()	Liefert die Methoden dieses Datentyps als Array aus <code>MethodInfo</code> -Objekten zurück
InvokeMember()	Ermöglicht das Ausführen einer Methode des Datentyps
IsSubclassOf()	Überprüft, ob der aktuelle Datentyp vom übergebenen Datentyp abgeleitet ist

Tabelle 20.3: Einige Methoden der Klasse `TypeBuilder` (Fortsetzung)

Die aufgeführten Methoden erklären sich schon fast von selbst anhand ihres Namens. Wichtig ist die Unterscheidung zwischen solchen Methoden, die auf ein enthaltenes Element zugreifen (sie beginnen alle mit `Get`), und solchen, die ein neues Element erstellen (sie beginnen alle mit `Define`). Neu erstellte Elemente werden als `Builder`-Objekte zurückgeliefert, z.B. `MethodBuilder` oder `EventBuilder`, bestehende Elemente als `Info`-Objekte, z.B. `MethodInfo` oder `FieldInfo`.

20.4.3 Die Klasse `ILGenerator`

Die beiden Klassen `MethodBuilder` und `ConstructorBuilder` haben eine Gemeinsamkeit, die sie von den anderen `Builder`-Klassen unterscheidet: In Methoden und Konstruktoren kann Code eingefügt werden, was z.B. bei Feldern nicht nötig ist. Außerdem können beide Parameter erhalten, die später ausgewertet werden können.

ILGenerator Die Parameter werden über die Methode `DefineParameter()` erzeugt, die ein Objekt vom Typ `ParameterInfo` zurückliefert. Zum Einfügen des Codes wird eine Instanz der Klasse `ILGenerator` benötigt, die Sie mit Hilfe der Methode `GetILGenerator()` erzeugen können. `ILGenerator` selbst liefert Methoden, mit denen der Code dann erzeugt werden kann. Die wichtigsten Methoden der Klasse `ILGenerator` finden Sie in Tabelle 20.4.

Methode	Funktion
BeginCatchBlock()	Beginnt einen Try ... Catch-Block
BeginExceptFilterBlock()	Beginnt einen Catch-Block für eine bestimmte Exception
BeginExceptionBlock()	Beginnt einen Catch-Block für alle anderen Exceptions
BeginFinallyBlock()	Beginnt einen Finally-Block
DeclareLocal()	Deklariert eine lokale Variable
DefineLabel()	Deklariert ein Label innerhalb der Methode
Emit()	Schreibt die Instruktionen hinaus, d.h. die Instruktionen werden an den Just-in-Time-Compiler weitergegeben. Dabei handelt es sich um IL-OpCodes, d.h. hier wird mit Intermediate Language gearbeitet.
EmitWriteLine()	Sendet Code an den Jitter, der einen String auf die Konsole schreibt
EndExceptionBlock()	Beendet einen Try ... Catch-Block
BeginScope()	Beginnt einen Codeblock
EndScope()	Beendet einen Codeblock

Tabelle 20.4: Methoden der Klasse ILGenerator

20.4.4 Hello World mit Reflection.Emit

Das bekannteste Programmierbeispiel aller Zeiten ist sicherlich das kleine »Hello World«-Programm, mittlerweile vermutlich in allen bekannten Programmiersprachen schon einmal programmiert worden. Deshalb soll dieses Beispiel auch für Reflection.Emit herhalten. Der Code ist relativ einfach, da es sich nur um eine einzige Methode handelt, und ausreichend kommentiert.

```
'=====
' EmitHelloWorld.sln
' Copyright 2002 Frank Eller
' Erzeugt eine dynamische Assembly
'=====

Imports System.Reflection
Imports System.Reflection.Emit
Imports System.Threading

Public Class Form1
    Inherits System.Windows.Forms.Form
```

```

#Region " Windows Form Designer generated code "

' Code der Übersichtlichkeit halber weggelassen

#End Region

Private Sub DoHelloWorld()

    ' --- Deklarationen
    Dim asblName As New AssemblyName()
    Dim asblBuilder As AssemblyBuilder
    Dim mdlBuilder As ModuleBuilder
    Dim aDomain As AppDomain
    Dim aTypeBuilder As TypeBuilder
    Dim aMethodBuilder As MethodBuilder
    Dim aGenerator As ILGenerator
    Dim aMethodInfo As MethodInfo

    ' --- Domain festlegen (aktuelle Domain)
    aDomain = Thread.GetDomain()

    ' --- AssemblyName festlegen
    asblName.Name = "Emit_HelloWorld"
    asblName.Version = New Version(1, 0, 0, 0)

    ' --- AssemblyBuilder erzeugen
    asblBuilder = aDomain.DefineDynamicAssembly( _
        asblName, AssemblyBuilderAccess.Save)

    ' --- Modul erzeugen
    ' --- gleicher Name wie Assembly
    mdlBuilder = asblBuilder.DefineDynamicModule( _
        "Emit_HelloWorld", "Emit_HelloWorld.exe")

    ' --- Typ definieren als Klasse, öffentlich,
    ' --- Name wie Assembly
    aTypeBuilder = mdlBuilder.DefineType( _
        "MyAssembly.Emit_HelloWorld", _
        TypeAttributes.Public)

    ' --- Methode Main deklarieren
    aMethodBuilder = aTypeBuilder.DefineMethod( _
        "Main", _
        (MethodAttributes.Public Or _
        MethodAttributes.Static), _
        Nothing, Nothing)

    ' --- ILGenerator erzeugen
    aGenerator = aMethodBuilder.GetILGenerator()

    ' --- Ausführbaren Code schreiben

```



```
aGenerator.EmitWriteLine("Hello World")
aGenerator.Emit(OpCodes.Ret)

' --- Klasse fertigstellen
aTypeBuilder.CreateType()

' --- Eintrittspunkt festlegen
aMethodInfo = aTypeBuilder.GetMethod("Main")
asb1Builder.SetEntryPoint(aMethodInfo)

' --- Assembly speichern
asb1Builder.Save("Emit_HelloWorld.exe")

End Sub

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

    DoHelloWorld()

End Sub
End Class
```

Listing 20.8: Hello World mit Reflection.Emit [EmitHelloWorld.sln]

Beachten Sie bitte, dass dies nur ein einfaches Beispiel ist. Wesentlich komplizierter wird es, wenn beispielsweise eine Addition oder irgendeine andere, komplexere Funktion ausgeführt werden soll. Wenn Sie mit `Reflection.Emit` arbeiten, arbeiten Sie mit IL-Code, so genannten *OpCodes* (im Beispiel wird nur der `OpCode Ret` benutzt, der den obersten Wert des Stacks an die aufrufende Funktion zurückliefert). Um `OpCodes` zu erklären, reicht der Platz an dieser Stelle leider nicht aus. Wenn Sie sehen wollen, wie ein solcher Code aussieht, können Sie das Tool *ILDASM.exe* verwenden, das Ihnen ein .NET-Programm im IL-Code anzeigt.

21 Namespaces System.Net und System.Net.Sockets

Das .NET Framework ist von Haus aus darauf ausgelegt, Internet und Desktop näher zusammenzubringen. Der Namespace `System.Net` beinhaltet eine große Anzahl von Klassen, mit denen auf das Internet, aber auch auf andere Netzwerke zugegriffen werden kann.

Diese Klassen liefern allerdings nur die Basisfunktionen. Eine vollwertige Klasse, um z.B. E-Mails abzurufen oder zu versenden, oder eine Klasse für den Zugriff auf Newsgroups finden Sie hier nicht. Dafür aber eine Menge Klassen, mit denen eine unabhängige Verbindung mit einem Netzwerk oder dem Internet aufgebaut werden kann und die auch als Grundlage für FTP, Mail oder News dienen können.

21.1 Der Namespace System.Net

Die Klassen in `System.Net` dienen dem Zugriff auf verschiedene Anwendungsprotokolle der TCP/IP-Protokollfamilie, insbesondere HTTP. Eine Übersicht über einige der Klassen in `System.Net` liefert Tabelle 21.1.

System.Net

Klasse	Funktion
Cookie	Ermöglicht die Auswertung eines Cookies, das über das Internet gesendet wurde
CookieCollection	Liefert eine Collection von Cookies, die über das Internet gesendet wurden
Dns	Ermöglicht es, eine Domain in die DNS-Entsprechung umzuwandeln oder umgekehrt

Tabelle 21.1: Einige Klassen von System.Net

Klasse	Funktion
FileWebRequest	Üblicherweise werden Anfragen zum Datei-Download oder zur Anzeige über das Web gesendet. Mit FileWebRe quest können Sie eine solche Anfrage auch an das Datei-system senden. Das entspricht einem URL im Browser, der mit file:// beginnt.
FileWebResponse	Liefert das Ergebnis der Anfrage mittels FileWebRequest
HttpWebRequest	Sendet eine Anfrage bezüglich eines URIs mittels des HTTP-Protokolls
HttpWebResponse	Liefert das Ergebnis einer Anfrage mittels HttpWebRequest
IPAddress	Steht für eine IP-Adresse im Internet bzw. Netzwerk
EndPoint	Liefert einen Netzwerk-Endpunkt bestehend aus IP- Adresse und Port-Nummer
WebClient	Liefert Standardmethoden zum Senden und Empfangen von Daten über das Internet oder das Intranet
WebRequest	Die Basisklasse für URI-Anfragen. Diese Klasse ist abstrakt.
WebResponse	Die Basisklasse für das Ergebnis einer URI-Anfrage. Auch diese Klasse ist abstrakt.

Tabelle 21.1: Einige Klassen von System.Net (Fortsetzung)

21.1.1 Die Klasse WebClient

WebClient WebClient ist eine recht einfache Klasse zum Zugriff auf Daten im Internet. Die Einfachheit der Benutzung bringt aber auch Beschränkungen mit sich. So können Sie WebClient zwar beispielsweise zum Upload oder Download von Dateien benutzen, für komplexere Operationen aber empfiehlt sich der Einsatz einer WebRequest- bzw. WebResponse-Klasse. WebClient selbst macht ebenfalls Gebrauch von der Funktionalität dieser Klassen.

Für einfache Aufgaben ist ein WebClient-Objekt dennoch eine gute Wahl, da die Verwendung recht unkompliziert ist. WebClient bietet verschiedene Methoden, mit denen Daten gesendet oder empfangen werden können. Eine Übersicht liefert Tabelle 21.2.

Methode	Funktion
DownloadData()	Lädt Daten von einer Ressource. Die Ressource wird über einen URI angegeben.
DownloadFile()	Lädt eine Datei von einer angegebenen Ressource herunter

Tabelle 21.2: Einige Methoden der WebClient-Klasse

Methoden	Funktion
OpenRead()	Öffnet einen Stream zum Lesen von einer Ressource
OpenWrite()	Öffnet einen Stream zum Versenden von Daten an eine Ressource
UploadData()	Lädt Daten zu einer Ressource
UploadFile()	Lädt eine Datei zu einer Ressource hoch

Tabelle 21.2: Einige Methoden der WebClient-Klasse (Fortsetzung)

HTML-Daten downloaden

Ein einfaches Beispiel für die WebClient-Klasse ist das Herunterladen einer HTML-Datei von einem beliebigen Server im Internet. Die HTML-Datei wird allerdings nicht verarbeitet, sondern im Quelltext angezeigt.

Das Hauptformular dieser kleinen Anwendung besteht aus einer Textbox für den URL, einer Richtextbox, in die die Daten gespeichert werden, und je einem Button für den Download und zum Beenden des Programms. Den Quelltext sehen Sie in Listing 21.1.

```
'=====
' GetHomepage.sln
' Copyright 2002 Frank Eller
' Homepage aus dem Internet abrufen
'=====

Imports System.Net
Imports System.IO
Imports System.Collections.Specialized

Public Class Form1
    Inherits System.Windows.Forms.Form

    #Region " Windows Form Designer generated code "

    'Der Code, der von VS.NET eingefügt wurde, wird
    'aus Übersichtsgründen nicht dargestellt.

    #End Region

    Private Function GetHomepage(ByVal url As String) _
        As String()

        ' --- Deklarationen
        Dim wClient As WebClient
        Dim sReader As StreamReader
        Dim line As String
```

```

Dim resultList As New StringCollection()
Dim result() As String

' --- Homepage-Inhalt mittels WebClient holen
wClient = New WebClient()
sReader = New StreamReader( _
    wClient.OpenRead(txtUrl.Text))
Do
    line = sReader.ReadLine
    If Not (line Is Nothing) Then
        resultList.Add(line)
    End If
Loop While Not (line Is Nothing)

ReDim result(resultList.Count - 1)
resultList.CopyTo(result, 0)
Return result

End Function

Private Sub btnGetHp_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnGetHp.Click

    If Not (txtUrl.Text = String.Empty) Then
        rtHomepage.Lines = GetHomepage(txtUrl.Text)
    End If

End Sub

Private Sub btnExit_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnExit.Click

    Me.Close()

End Sub
End Class

```

Listing 21.1: Downloaden einer Homepage im Quelltext [GetHomepage.sln]

OpenRead() Die relevante Methode des Programms ist `GetHomepage()`. Zunächst wird eine `WebClient`-Instanz erzeugt. Die Methode `OpenRead()` dieser Instanz liefert ein `Stream`-Objekt, das dem Konstruktor eines `StreamReader`-Objekts übergeben wird. Nun kann über dessen Methode `Read()` der gesamte Inhalt der angegebenen Datei ausgelesen und in der Richtextbox dargestellt werden.

Da die Anfrage tatsächlich über HTTP ausgeführt wird, wird auch entsprechender serverseitiger Code ausgeführt, bevor die Antwort zurückgeliefert wird. Wenn Sie also einen URL angeben, der beispielsweise mit ASP endet (oder mit PHP), wird zunächst der in der Homepage stehende Code ausgeführt und dann die Antwort zurückgeliefert. Zum Downloaden des Quellcodes eines Scripts ist diese Möglichkeit nicht geeignet.

Serverseitiger Code

Das Ergebnis des Codes zeigt Abbildung 21.1.

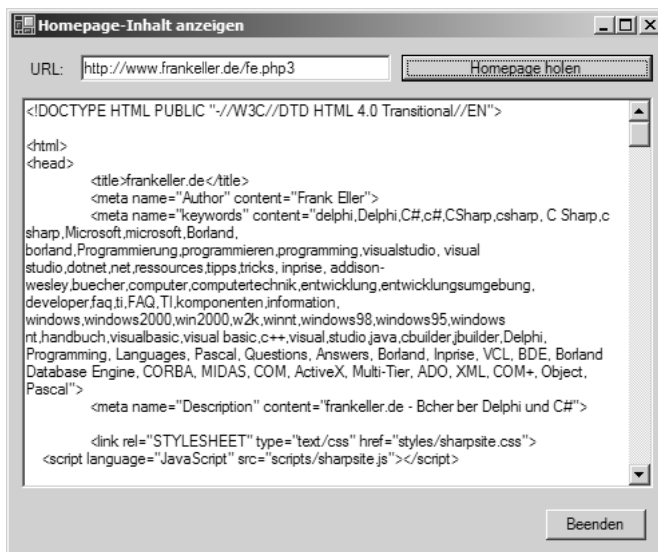


Abbildung 21.1: Der Inhalt der Homepage von Frank Eller [GetHomepage.sln]

21.1.2 Die Klasse WebRequest

Wesentlich flexibler als `WebClient` ist die Klasse `WebRequest` bzw. die von ihr abgeleiteten Klassen. Für den Zugriff auf das Internet empfiehlt sich die Verwendung der Klasse `HttpWebRequest`; wollen Sie auf das Dateisystem des lokalen Rechners zugreifen, verwenden Sie die Klasse `FileWebRequest`.

WebRequest

Eine Instanz dieser Klassen erzeugen Sie nicht mit Hilfe eines Konstruktors, sondern über die statische (shared) Methode `Create()`, die die Klasse `WebRequest` implementiert. Sie haben dabei zwei Möglichkeiten: Entweder Sie benutzen direkt eine der Klassen `HttpWebRequest` oder `FileWebRequest` oder Sie erzeugen eine Instanz der eigentlich abstrakten Klasse `WebRequest`, was ersterem

HttpWebRequest und FileWebRequest

aber gleichkommt. Die Methode `Create()` erwartet einen URL als Parameter. Wenn Sie der Methode `Create()` von `WebRequest` einen URL übergeben, der mit »http://« beginnt, wird automatisch eine Instanz von `HttpWebRequest` erzeugt und in ein `WebRequest`-Objekt verpackt. Damit ist es auf folgende Art und Weise möglich, ein `HttpWebRequest`-Objekt zu erzeugen:

```
Dim wReq As WebRequest
Dim httpReq As HttpWebRequest

wReq = WebRequest.Create("http://www.frankeller.de")
httpReq = CType(wReq,HttpWebRequest)
```

Beginnt der übergebene URL entsprechend mit »file://«, wird ein `FileWebRequest`-Objekt erzeugt.

**HttpWeb
Response und
FileWebResponse**

21.1.3 Die Klasse WebResponse

Ein `WebRequest`-Objekt sendet eine Anfrage an einen Server und erhält als Ergebnis ein Objekt vom Typ `WebResponse`. Sie können über die Methode `GetResponse` auf dieses Objekt zugreifen und es verwenden. Auch für die Klasse `WebResponse`, die wie `WebRequest` abstrakt ist, gibt es fertige Implementationen in Form der Klassen `HttpWebResponse` und `FileWebResponse`.

Mit den `Response`-Klassen sind Sie nun in der Lage, einen `Stream` für Daten zu öffnen, den Status der Verbindung zu ermitteln oder auch den Namen des Servers, der die Antwort letztendlich geschickt hat. Hier wird vermutlich hauptsächlich die Klasse `HttpWebResponse` zum Einsatz kommen. Die Eigenschaften dieser Klasse liefern einige interessante Informationen. Tabelle 21.3 zeigt eine Übersicht.

Eigenschaft	Bedeutung
<code>ContentEncoding</code>	Liefert die Encoding-Methode für die Antwort
<code>ContentLength</code>	Liefert die Länge der angeforderten Information
<code>ContentType</code>	Liefert die Art des Inhalts der angeforderten Information
<code>Cookies</code>	Liefert die Cookies, die möglicherweise im HTTP-Header der Antwort enthalten sind, in Form eines <code>Cookie-Collection</code> -Objekts

Tabelle 21.3: Die Eigenschaften der `HttpWebResponse`-Klasse

Eigenschaft	Bedeutung
Headers	Liefert die verschiedenen HTTP-Header der Antwort in Form eines <code>WebHeaderCollection</code> -Objekts
LastModified	Liefert das Datum, an dem die angeforderte Datei zum letzten Mal geändert wurde
Method	Liefert die Methode, die zum Zurückliefern der Antwort verwendet wird
ProtocolVersion	Liefert die Version des HTTP-Protokolls
ResponseURI	Liefert den URI, der letztendlich auf die Anfrage geantwortet hat
Server	Liefert den Server, der die Antwort gesendet hat
StatusCode	Liefert den Statuscode der Antwort
StatusDescription	Liefert die Statusbeschreibung der Antwort

Tabelle 21.3: Die Eigenschaften der `HttpWebResponse`-Klasse (Fortsetzung)

Eigenschaften eines Requests ermitteln

Sobald eine Verbindung mit einer Datei bzw. einem URL hergestellt ist, kann ein `HttpWebResponse`-Objekt erzeugt und die enthaltenen Informationen ausgelesen werden. Z.B. könnten Sie kontrollieren, ob der Server von Microsoft der IIS 5.0 ist oder schon IIS 5.1 – oder arbeitet Microsoft gar schon mit IIS 6.0?

Ein kleines Programm liefert alle Daten zu einem URL. Wenn Sie lediglich den URL selbst angeben, also beispielsweise `http://www.microsoft.com/`, werden die Daten der Standard-Datei ermittelt; wenn Sie einen Dateinamen angeben, erhalten Sie die Daten der Datei. Wenn der URL nicht existiert oder eine Datei nicht gefunden werden kann, wird eine Exception ausgelöst.

```
'=====
' GetResponseProperties.sln
' Copyright 2002 Frank Eller
' Eigenschaften eines Requests ermitteln
'=====

Imports System.Net
Imports System.IO

Public Class Form1
    Inherits System.Windows.Forms.Form

    #Region " Windows Form Designer generated code "

    'Der Code, der von VS.NET eingefügt wurde, wird
```

'aus Übersichtsgründen nicht dargestellt.

#End Region

Private Sub GetResponseInfo(**ByVal** url **As** String)

' --- Deklarationen

Dim aRequest **As** HttpWebRequest

Dim aResponse **As** HttpWebResponse

aRequest = HttpWebRequest.Create(url)

Try

aResponse = aRequest.GetResponse()

' --- Response-Properties schreiben

lbxResult.Items.Clear()

With aResponse

lbxResult.Items.Add(_

"Server: " + .Server)

lbxResult.Items.Add(_

"URI: " + .ResponseUri.ToString())

lbxResult.Items.Add("Protokoll: " + _

.ProtocolVersion.ToString())

lbxResult.Items.Add("")

lbxResult.Items.Add("Status: " + _

.StatusDescription)

lbxResult.Items.Add("")

lbxResult.Items.Add("Größe: " + _

.ContentLength.ToString() + " Bytes")

lbxResult.Items.Add("Typ: " + _

.ContentType)

lbxResult.Items.Add("Modifiziert: " + _

.LastModified.ToString())

End With

Catch

MessageBox.Show(_

"Die Datei konnte nicht gefunden werden", _

"Fehler", MessageBoxButtons.OK, _

MessageBoxIcon.Hand)

End Try

End Sub

Private Sub btnInfo_Click(_

ByVal sender **As** System.Object, _

ByVal e **As** System.EventArgs) _

Handles btnInfo.Click

GetResponseInfo(txtUrl.Text)

```
End Sub

Private Sub btnExit_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnExit.Click

    Me.Close()

End Sub
End Class
```

Listing 21.2: Eigenschaften eines Requests ermitteln [GetResponseProperties.sln]

Das Listing ist recht einfach und sollte keine Probleme bereiten. Falls Sie sich aber jetzt wirklich fragen, welchen Server Microsoft benutzt, hier die Antwort in Abbildung 21.2.

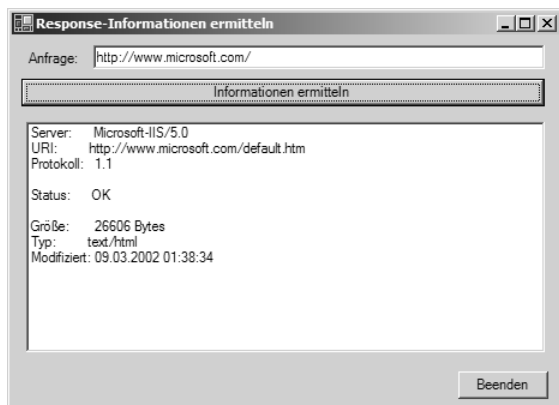


Abbildung 21.2: Das Beispielprogramm im Einsatz [GetResponseProperties.sln]

Datei-Download

Die Klassen `WebRequest` und `WebResponse` eignen sich auch sehr gut, um z.B. eine Datei aus dem Internet herunterzuladen. Über die Methode `GetResponseStream()` wird ein `Stream`-Objekt zurückgeliefert, über das die Daten downgeloadet werden können. Da nicht bekannt sein muss, um welche Art von Daten es sich handelt, bietet es sich an, ein binäres Verfahren zu verwenden. Über ein Objekt vom Typ `BinaryReader` und eines vom Typ `BinaryWriter` kann die Datei auf die Festplatte kopiert werden.

Da `BinaryReader` und `BinaryWriter` mit einem Byte-Puffer arbeiten und somit immer bekannt ist, wie viele Bytes gerade heruntergela-

den wurden, ist es auch einfach, eine Progressbar zu implementieren, so dass man den aktuellen Fortschritt des Download-Vorgangs verfolgen kann. Listing 21.3 zeigt ein Beispielprogramm, mit dem eine beliebige Datei aus dem Web heruntergeladen werden kann.

```
'=====
' GetFileFromWeb.sln
' Copyright 2002 Frank Eller
' Datei aus dem Internet downloaden
'=====

Imports System.Net
Imports System.IO

Public Class Form1
    Inherits System.Windows.Forms.Form

    #Region " Windows Form Designer generated code "

        'Der Code, der von VS.NET eingefügt wurde, wird
        'aus Übersichtsgründen nicht dargestellt.

    #End Region

    Private Sub btnChooseLocation_Click( _
        ByVal sender As System.Object, _
        ByVal e As System.EventArgs) _
        Handles btnChooseLocation.Click

        If dlgSaveFile.ShowDialog() = DialogResult.OK _
        Then
            txtTarget.Text = dlgSaveFile.FileName
        End If

    End Sub

    Private Sub GetFile(ByVal url As String)

        ' --- Deklarationen
        Dim source As String = txtUrl.Text
        Dim target As String = txtTarget.Text
        Dim webreq As HttpWebRequest
        Dim webResp As HttpWebResponse
        Dim bReader As BinaryReader
        Dim bWriter As BinaryWriter
        Dim outStream As FileStream
        Dim buffer() As Byte = New Byte(1024) {}
        Dim bytesRead As Integer
```

```
webreq = HttpWebRequest.Create(source)

Try
    webResp = webreq.GetResponse

    prgBar.Maximum = webResp.ContentLength

    outStream = New FileStream( _
        target, FileMode.Create)
    bReader = New BinaryReader( _
        webResp.GetResponseStream())
    bWriter = New BinaryWriter(outStream)

    Do
        bytesRead = bReader.Read(buffer, 0, 1024)
        bWriter.Write(buffer, 0, bytesRead)
        prgBar.Value += bytesRead
        prgBar.Update()
    Loop Until bytesRead = 0

    bReader.Close()
    bWriter.Close()
    outStream.Close()

    MessageBox.Show("Download beendet", _
        "Download", MessageBoxButtons.OK, _
        MessageBoxIcon.Information)
Catch
    MessageBox.Show( _
        "Die angegebene Datei kann nicht " _
        + "gefunden werden", _
        "Fehler", MessageBoxButtons.OK, _
        MessageBoxIcon.Hand)
End Try
End Sub

Private Sub btnDownload_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnDownload.Click

    GetFile(txtUrl.Text)

End Sub

Private Sub btnExit_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnExit.Click
```

```
Me.Close()  
End Sub  
End Class
```

Listing 21.3: Datei aus dem Internet downloaden [GetFileFromWeb.sln]

Als Lese- und Schreibpuffer wird in diesem Beispiel ein Puffer der Größe 1024 Bytes verwendet. Die Methode `Read()` von `BinaryReader` liefert die Anzahl der tatsächlich gelesenen Bytes, die dann für die Anzeige der Progressbar benutzt werden. Wird die Datei nicht gefunden, wird eine Exception ausgelöst, die innerhalb des Programms abgefangen wird. Stattdessen wird eine MessageBox angezeigt. Das Programm zur Laufzeit zeigt Abbildung 21.3.

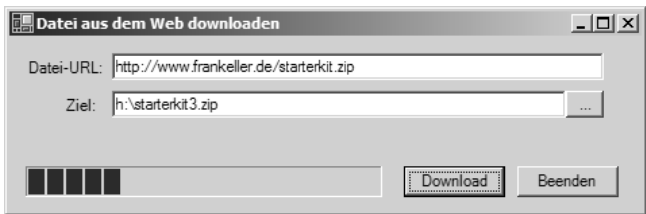


Abbildung 21.3: Datei-Download mit Fortschrittsanzeige [GetFileFromWeb.sln]

21.2 Der Namespace System.Net.Sockets

Der Namespace `System.Net.Sockets` enthält eine Winsock-Implementation (also TCP/IP für Microsoft Windows), aufgeteilt auf mehrere Klassen und komplett in Managed Code programmiert. Die hier verfügbaren Klassen bilden die Basis für Applikationen, die mit dem Internet kommunizieren können, z.B. Mail-Clients, FTP-Clients oder auch Telnet-Clients. Eine Übersicht über die enthaltenen Klassen liefert Tabelle 21.4.

Klasse	Funktion
MulticastOption	Enthält Adresswerte für Multicast-Pakete
NetworkStream	Die Klasse <code>NetworkStream</code> repräsentiert den Datenstrom für alle Arten von Netzwerkzugriffen.
Socket	Eine Implementation des Berkeley-Socket-Interfaces
TcpClient	Eine der Hauptklassen dieses Namespaces. Alle TCP-Client-Verbindungen werden über diese Klasse abgewickelt, z.B. Zugriff auf Mail, FTP, Telnet usw.

Tabelle 21.4: Die Klassen des Namespaces System.Net.Sockets

Klasse	Funktion
TcpListener	Eine Klasse zum Überwachen von Netzwerkverbindungen
UdpClient	Eine Klasse, die Netzwerkdienste über das User-Data-gram-Protokoll zur Verfügung stellt. Mit dieser Klasse kann z.B. ein Netzwerk-Chat implementiert werden.

Tabelle 21.4: Die Klassen des Namespaces System.Net.Sockets (Fortsetzung)

21.2.1 Die Klasse NetworkStream

`NetworkStream` implementiert einen Datenstrom. Anders als die bereits bekannten `Stream`-Klassen, die in der Regel nur in eine Richtung arbeiten, ist ein `NetworkStream`-Objekt in der Lage, sowohl Daten zu empfangen als auch zu senden, natürlich nur, wenn die Gegenstelle dies unterstützt.

Ob das Senden von Daten möglich ist, kann über die Eigenschaft `CanWrite` ermittelt werden. Entsprechend liefert die Eigenschaft `CanRead` einen booleschen Wert zurück, der angibt, ob von diesem Datenstrom gelesen werden kann. Abhängig ist das vom Host, mit dem er verbunden ist.

Ob Daten verfügbar sind, kann über die Eigenschaft `DataAvailable` ermittelt werden. Die Methode `Read()` liest aus dem Datenstrom, `Write()` schreibt in den Datenstrom. Das Lesen und Schreiben kann auch asynchron erfolgen, dazu dienen die Methoden `BeginRead()/EndRead()` sowie `BeginWrite()/EndWrite()`. Zum byteweisen Lesen gibt es noch die Methoden `ReadByte()` und `WriteByte()`.

**BeginRead(),
EndRead(),
BeginWrite()
und EndWrite()**

21.2.2 Die Klasse TcpClient

`TcpClient` ist so etwas wie die Hauptklasse dieses Namespaces, da über sie die meisten Verbindungen abgewickelt werden. TCP kommt immer dann zum Einsatz, wenn irgendwelche Daten aus dem Internet oder einem auf TCP/IP basierenden Netzwerk abgerufen werden sollen.

Eine Instanz der Klasse `TcpClient` wird über die Methode `Connect` mit einem Netzwerk-Endpunkt verbunden. Die Methode `GetStream()` liefert dann ein `NetworkStream`-Objekt, das die Daten überträgt. Verschiedene Eigenschaften von `TcpClient` liefern noch weitere Informationen, z.B. die Zeit, die der Client wartet, bis die Daten gesendet werden, nachdem er eine Verbindung instanziiert hat.

**Verbinden über
TcpClient**

Wichtig bei der Verwendung von `TcpClient` ist, dass hier alle Arten von Daten übermittelt werden können. `TcpClient` arbeitet mit einem Datenstrom vom Typ `NetworkStream`, so dass Sie nach erfolgreicher Verbindung Daten sowohl lesen als auch schreiben können.

Whois-Abfrage

Ein einfaches Beispiel für die Verwendung von `TcpClient` ist eine Whois-Abfrage. Das gleiche Ergebnis erhalten Sie natürlich auch auf der Seite von Denic, allerdings nur für die deutschen Domains. Wollen Sie z.B. auch internationale Domains überprüfen, müssten Sie zum Internic und dort eine Abfrage durchführen. Mit einem kleinen Programm kann die Abfrage auf beide Stellen verteilt werden, je nach Endung. Den Quelltext zeigt Listing 21.4.

```
'=====
' WhoIs.sln
' Copyright 2002 Frank Eller
' Whois-Anfrage im Internet
'=====

Imports System.Net
Imports System.Net.Sockets
Imports System.IO

Public Class Form1
    Inherits System.Windows.Forms.Form

    #Region " Windows Form Designer generated code "

    'Der Code, der von VS.NET eingefügt wurde, wird
    'aus Übersichtsgründen nicht dargestellt.

    #End Region

    Private Function GetWhoisServer( _
        ByVal domain As String) As String

        Dim topLevelDomain As String

        topLevelDomain = _
            domain.Substring(domain.LastIndexOf("."))

        Select Case topLevelDomain
            Case _
                ".biz", ".com", ".edu", ".info", ".museum", _
                ".int", ".org", ".net"
```



```
        Return "rs.internic.net"
    Case ".de"
        Return "whois.denic.de"
    Case Else
        Return ""
    End Select

End Function

Private Function doWhois( _
    ByVal domain As String) As String

    ' --- Deklarationen
    Dim tcp As TcpClient
    Dim sWriter As StreamWriter
    Dim sReader As StreamReader
    Dim nStream As NetworkStream
    Dim whoisServer As String

    ' --- Server ermitteln
    whoisServer = GetWhoisServer(domain)

    If whoisServer = String.Empty Then
        MessageBox.Show( _
            "Der passende Server ist nicht verfügbar", _
            "Fehler", MessageBoxButtons.OK, _
            MessageBoxIcon.Exclamation)
        Exit Function
    End If

    tcp = New TcpClient(whoisServer, 43)
    nStream = tcp.GetStream()
    sWriter = New StreamWriter(nStream)
    sReader = New StreamReader(nStream)

    sWriter.AutoFlush = True
    sWriter.WriteLine(domain)

    Return sReader.ReadToEnd()

End Function

Private Sub btnWhois_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnWhois.Click

    If Not (txtDomain.Text = String.Empty) Then
        txtResult.Text = doWhois(txtDomain.Text)
    End If
End Sub
```

```

End Sub

Private Sub btnExit_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnExit.Click

    Me.Close()

End Sub
End Class

```

Listing 21.4: Whois-Abfrage mit automatischer Serverwahl [Whois.sln]

Diese kleine Applikation führte bei einer Abfrage der Domain *microsoft.com* (Domain-Namen müssen immer ohne das führende »www« angegeben werden) zu einem recht verwunderlichen Ergebnis, wie Abbildung 21.4 zeigt.

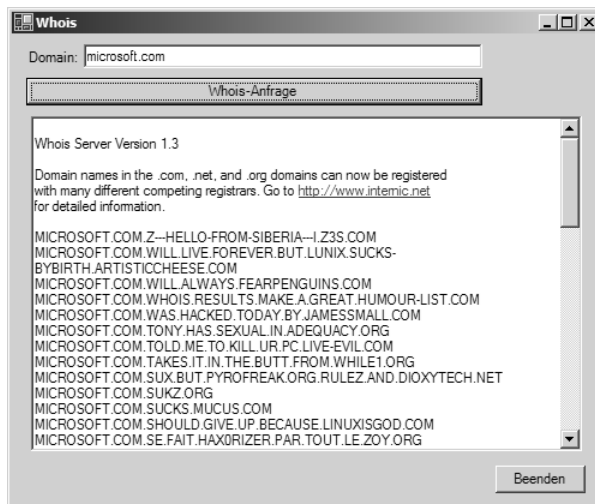


Abbildung 21.4: Abfrage von *microsoft.com* am 09. März 2002 [Whois.sln]

Die Abfrage auf *www.internic.com* zeigte allerdings das korrekte Ergebnis. Übrigens stammen diese Ergebnisse aus der Datenbank des Internic und nicht von Microsoft direkt – entgegen der Aussage in der Grafik ist Microsoft also nicht gehackt, sondern lediglich die Ausgabe der Internic-Datenbank verändert worden.

E-Mails kontrollieren

Ein weiteres Beispiel soll zeigen, wie die Klasse `TcpClient` dazu verwendet werden kann, E-Mails zu kontrollieren. Der Zugriff ist ja auf jeden Server möglich, also auch auf einen Pop3-Server.

Wenn der Zugriff unter Angabe von Benutzername und Passwort erfolgt ist, liefert der Server seine Angaben zurück. Die Auswertung dieser Angaben geschieht wieder über ein `NetworkStream`-Objekt. Bei E-Mails müssen nun noch die richtigen Informationen aus den Daten herausgefischt und interpretiert werden.

Die Anmeldung an einem Pop3-Server geschieht durch Übermittlung von Benutzername und Passwort nacheinander. Der Server erwartet ein bestimmtes Kommando, damit er weiß, welche Daten jetzt gesendet werden. Das Kommando für die Übergabe des Benutzernamens heißt »user«, gefolgt vom eigentlichen Benutzernamen und einem Return. Das Kommando für das Passwort heißt »pass«, gefolgt wieder vom Passwort und einem Return.

Anmeldung am Server

Schlägt die Anmeldung fehl, liefert der Server die Zeichenfolge »-ERR«. Somit kann leicht kontrolliert werden, ob die Anmeldung erfolgreich verlaufen ist.

Fehler

In diesem Beispiel sollen die E-Mails lediglich kontrolliert werden, d.h. es sollen keine weiteren Daten übertragen werden. Zum Auflisten des Inhalts dient das Kommando »list«. Ist alles in Ordnung, liefert der Server die Zeichenfolge »+OK« und im nächsten String die Anzahl der Mails. Diese steht am Anfang der Zeichenkette, d.h. alles andere muss abgeschnitten werden.

Der einfachste Weg hierzu ist die Methode `Split()` der Klasse `String`, die einen String in ein String-Array aufteilt. Getrennt ist die Anzahl der Mails vom Rest durch ein Leerzeichen, das den Trenner für die `Split()`-Methode darstellt. Aus dem resultierenden Array wird dann das erste Element, also das Element mit dem Index 0 benötigt.

Umgesetzt in eine Programmzeile sieht diese Auswertung dann so aus:

```
GetMailCount = Convert.ToInt32(resultStr.Split(splitChar)(0))
```

Für diese Zeile gilt, dass `splitChar` ein Leerzeichen enthält und `resultStr` der zurückgelieferte String ist. Die gesamte Klasse ist in Listing 21.5 dargestellt.

```
'=====
' MailCheck.sln
' Copyright 2002 Frank Eller
' Klasse PopChecker
'=====

Imports System.Net
Imports System.Net.Sockets
Imports System.IO

Public Class PopChecker

    Private _serverName As String = String.Empty
    Private _userName As String = String.Empty
    Private _userPort As Integer = -1
    Private _password As String = String.Empty

    Public Property ServerName() As String
        Get
            Return _serverName
        End Get
        Set(ByVal Value As String)
            _serverName = Value
        End Set
    End Property

    Public Property UserName() As String
        Get
            Return _userName
        End Get
        Set(ByVal Value As String)
            _userName = Value
        End Set
    End Property

    Public Property UserPort() As Integer
        Get
            Return _userPort
        End Get
        Set(ByVal Value As Integer)
            _userPort = Value
        End Set
    End Property

    Public Property Password() As String
        Get
            Return _password
        End Get
        Set(ByVal Value As String)
            _password = Value
        End Set
    End Property
End Class
```

```

End Property

Public Sub New()

End Sub

Public Sub New(ByVal server As String, _
    ByVal port As String, ByVal user As String, _
    ByVal password As String)

    Me.ServerName = server
    Me.UserPort = Convert.ToInt32(port)
    Me.UserName = user
    Me.Password = password

End Sub

Public Function GetMailCount() As Integer

    ' --- Deklarationen
    Dim client As TcpClient
    Dim nStream As NetworkStream
    Dim sReader As StreamReader
    Dim sendStr As String
    Dim outBuffer As Byte()
    Dim resultStr As String = String.Empty
    Dim okPosition As Integer = 0
    Dim splitChar As Char = " "

    ' --- E-Mails checken
    ' --- TCPClient wird instanziiert
    client = New TcpClient(_serverName, _userPort)

    Project.MainForm.lblStatus.Text = _
        "Verbindungsaufbau ..."

    Try
        nStream = client.GetStream()
        sReader = New StreamReader(nStream)

        sReader.ReadLine()

        ' --- Namen schicken
        sendStr = Global.Cmd_User + Me._userName + _
            vbCrLf

        outBuffer = _
            System.Text.Encoding.ASCII.GetBytes( _
                sendStr.ToCharArray())
        nStream.Write(outBuffer, 0, outBuffer.Length)
        sReader.ReadLine()
    
```

```

' --- Passwort schicken
sendStr = Global.Cmd_Pass + Me._password + _
vbCrLf
outBuffer = _
    System.Text.Encoding.ASCII.GetBytes( _
        sendStr.ToCharArray())
nStream.Write(outBuffer, 0, outBuffer.Length)

' --- Fehler checken
' --- zurückgeliefert wird "-ERR" ...
If sReader.ReadLine().IndexOf("-ERR") >= 0 Then
    Project.MainForm.lblStatus.Text = _
        "Anbindung an Server fehlgeschlagen"
    GetMailCount = -1
    Exit Try
End If

Project.MainForm.lblStatus.Text = _
    "Anmeldung erfolgreich"

' --- Jetzt die Mails zählen
sendStr = Global.Cmd_List + vbCrLf
outBuffer = _
    System.Text.Encoding.ASCII.GetBytes( _
        sendStr.ToCharArray())
nStream.Write(outBuffer, 0, outBuffer.Length)
resultStr = sReader.ReadLine()

' --- Jetzt im gelieferten String nach
' --- Mail-Anzahl suchen
okPosition = resultStr.IndexOf("+OK")
If okPosition >= 0 Then
    resultStr = sReader.ReadLine()
    resultStr.Trim()
    GetMailCount = Convert.ToInt32( _
        resultStr.Split(splitChar)(0))
End If

Finally
    sReader.Close()
    nStream.Close()
    client.Close()
End Try

End Function

End Class

```

Listing 21.5: Die Klasse PopChecker zum Überprüfen der E-Mail-Anzahl
[MailCheck.sln]

Die verschiedenen Kommandos an den Server sind in einer Klasse namens `Global` als statische (*shared*) Attribute deklariert. Dementsprechend einfach sieht diese Klasse auch aus.

```
Public Class Global
    Public Shared Cmd_User As String = "user "
    Public Shared Cmd_Pass As String = "pass "
    Public Shared Cmd_List As String = "list"
End Class
```

Listing 21.6: Die Klasse Global [Mailcheck.sln]

Der Rest des Programms dient lediglich der Anzeige der E-Mail-Anzahl bzw. der Eingabe der Daten. Die Auslagerung der Methode `Main()` in eine andere Klasse als die des Hauptformulars ist notwendig, da aus der Klasse `PopChecker` heraus ebenfalls auf das Hauptformular zugegriffen wird.

```
'=====
' MailCheck.sln
' Copyright 2002 Frank Eller
' Anzahl der Mails auf einem Server checken
'=====

Public Class Project

    Public Shared MainForm As New Form1()

    <STAThread(>> _
    Public Shared Sub Main()
        Application.Run(MainForm)
    End Sub

End Class

Public Class Form1
    Inherits System.Windows.Forms.Form

    #Region " Windows Form Designer generated code "

        'Der Code, der von VS.NET eingefügt wurde, wird
        'aus Übersichtsgründen nicht dargestellt.

    #End Region

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

```

Dim result As Integer
Dim pChecker As New PopChecker(txtServer.Text, _
    txtPort.Text, txtUser.Text, txtPass.Text)

result = pChecker.GetMailCount()
If result > -1 Then
    lblStatus.Text += result.ToString() + " Mails"
End If

End Sub

Private Sub Button2_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles Button2.Click

    Me.Close()

End Sub
End Class

```

Listing 21.7: Anzahl der Mails auf einem Server abfragen [MailCheck.sln]

21.3 Hilfsklassen

Im Namespace `System.Net` befinden sich noch einige Klassen, die man grob als »Hilfsklassen« bezeichnen könnte. Unter anderem handelt es sich dabei um die Klassen `IPAddress`, `EndPoint`, `IPHostEntry` und `Dns`.

- IPAddress** `IPAddress` steht für eine IP-Adresse irgendwo im Netzwerk. Anstatt also stets eine IP-Adresse angeben zu müssen, können Sie auch ein `IPAddress`-Objekt erzeugen und dieses verwenden. Die Klasse `IPAddress` besitzt ein statisches Element namens `Loopback`, das die Adresse des Localhost-Servers beinhaltet (in der Regel 127.0.0.1).
- EndPoint** `EndPoint` verbindet eine IP-Adresse mit einem Port und stellt somit einen Endpunkt irgendwo im Netzwerk oder Internet dar.
- IPHostEntry** `IPHostEntry` steht für einen Host-Eintrag. Wenn Sie zu einer Seite surfen, steht der von Ihnen eingegebene String für den Host-Eintrag. Über ein `IPHostEntry`-Objekt können Sie mehr herausfinden, z.B. über Aliase dieses Hosts oder sämtliche IP-Adressen, die er verwendet.

Die Klasse `Dns` ermöglicht das Herausfinden von Host-Informationen oder das Umwandeln einer Klartext-Bezeichnung für einen Host in das IP-Äquivalent.

Dns

Das folgende Beispiel zeigt einige der Hilfsklassen im Einsatz. Es handelt sich um ein kleines Programm zur Ermittlung weiterer Informationen über einen Host, also eine Internet-Adresse. Das Programm ermittelt sowohl die Aliase als auch die verwendeten IP-Adressen sowie den realen Namen des Hosts. Listing 21.8 zeigt den Quellcode.

```
'=====
' GetDns.sln
' Copyright 2002 Frank Eller
' DNS-Auflösung
'=====

Imports System.Net

Public Class Form1
    Inherits System.Windows.Forms.Form

    #Region " Windows Form Designer generated code "

    'Der Code, der von VS.NET eingefügt wurde, wird
    'aus Übersichtsgründen nicht dargestellt.

    #End Region

    Private Sub btnAddress_Click( _
        ByVal sender As System.Object, _
        ByVal e As System.EventArgs) _
        Handles btnAddress.Click

        Dim host As IPHostEntry
        Dim aIpAddress As IPAddress
        Dim aliasStr As String

        host = Dns.Resolve(txtHost.Text)
        txtRealHost.Text = host.HostName

        For Each aIpAddress In host.AddressList
            lbxaddress.Items.Add(aIpAddress)
        Next

        For Each aliasStr In host.Aliases
            lbxAlias.Items.Add(aliasStr)
        Next

    End Sub
```

```
Private Sub btnExit_Click( _  
    ByVal sender As System.Object, _  
    ByVal e As System.EventArgs) _  
    Handles btnExit.Click  
  
    Me.Close()  
  
End Sub  
End Class
```

Listing 21.8: Informationen über einen Host ermitteln [GetDns.sln]

Das Resultat sehen Sie in Abbildung 21.5.



Abbildung 21.5: Die ermittelten Informationen [GetDns.sln]

22 Namespace System.Management

Die FCL enthält 2246 öffentliche Klassen (System.* und Microsoft.*). Darf es noch etwas mehr sein? Im Namespace System.Management verstecken sich weitere 2400 Klassen für den Zugriff auf das Windows-Betriebssystem. Dieses Kapitel zeigt, wie Sie diese Klassen finden und nutzen können.

22.1 Windows Management Instrumentation (WMI)

Der Namespace System.Management ist die .NET-Implementierung eines Meta-Objektmodells für die Windows Management Instrumentation (WMI). Die WMI ist ein übergreifender Ansatz zum Zugriff auf alle möglichen Arten von System- und Netzwerkinformationen. WMI ermöglicht auch den Zugriff auf Informationen aus Quellen wie Registry, Dateisystem und Ereignisprotokollen, die durch andere COM- und .NET-Komponenten bereits abgedeckt sind. Während WMI den Vorteil der Einheitlichkeit bietet, sind die speziellen Komponenten im konkreten Anwendungsfall oft etwas einfacher zu handhaben. WMI ist weitaus komplexer als andere Komponenten.

WMI-Grundlagen

Für den Zugriff auf WMI gibt es sowohl eine COM- als auch eine .NET-Komponente. In diesem Kapitel wird die .NET-Komponente vorgestellt, die als Namespace Teil der .NET Framework Class Library (FCL) ist.

Allgemeine Informationen zu WMI finden Sie in Anhang C.



Meta-Objektmodell

In COM ist WMI ein Meta-Objektmodell, d.h. es gibt ein (einfaches) COM-Objektmodell für den Zugriff auf das (umfangreiche) WMI-Objektmodell. Auch in .NET gibt es ein Meta-Objektmodell für WMI, jedoch besteht alternativ die Möglichkeit, direkt mit »gewrappten« WMI-Klassen zu arbeiten. Das reduziert nicht nur die Komplexität von WMI, sondern ermöglicht sogar, die Intellisense-Unterstützung von Visual Studio .NET (oder anderen Editoren, die die .NET-Reflection zur Eingabeunterstützung nutzen) zu verwenden. Für einige Operationen (z.B. die Suche mit der WMI Query Language) ist aber weiterhin das Meta-Objektmodell zwingend erforderlich.

22.2 Programmieren mit dem Meta-Objektmodell

Zunächst soll das Meta-Objektmodell besprochen werden. Leider hat Microsoft auch hier gegenüber COM stark geändert, ohne dass es dafür einen zwingenden Grund gab.



Hinweise zum Begriff Meta-Objektmodell finden Sie in Anhang C.

22.2.1 Überblick über das Objektmodell

Zentrale Klassen

Zentrale Klassen des neuen Objektmodells sind:

► `ManagementObject`

Diese Klasse repräsentiert ein WMI-Objekt.

► `ManagementClass`

Diese Klasse repräsentiert eine WMI-Klasse. `ManagementClass` ist von `ManagementObject` abgeleitet.

► `ManagementBaseObject`

Beide Klassen sind von `ManagementBaseObject` abgeleitet. Diese Klasse ist nicht abstrakt, sondern wird an verschiedenen Stellen im Objektmodell auch verwendet.



Verwechseln Sie nicht die Begriffe »Managed Object« und »Management-Object«. Managed Object ist der allgemeine Begriff für ein WMI-Objekt oder eine WMI-Klasse. ManagementObject ist der Name einer Meta-Klasse zur Verwaltung von beliebigen Managed Objects.

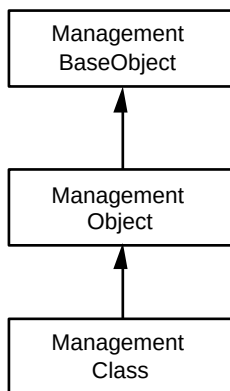


Abbildung 22.1: Vererbungshierarchie in System.Management

Die folgende Abbildung zeigt den Hauptteil des neuen Meta-Objektmodells für WMI. **Objektmodell**

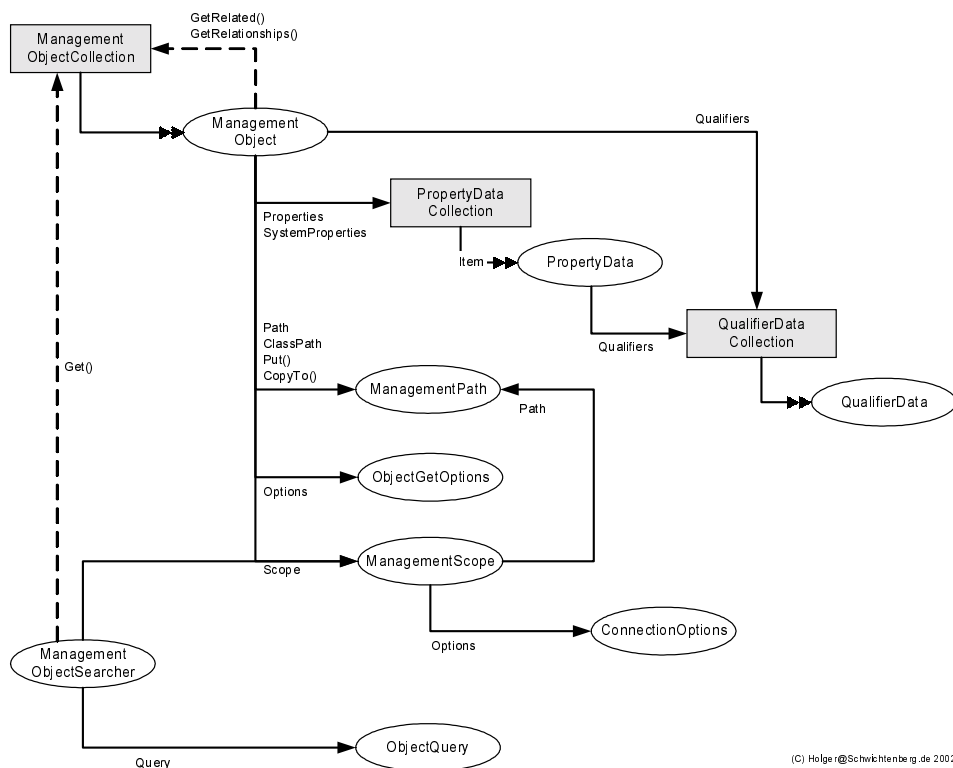


Abbildung 22.2: Erster Ausschnitt aus dem Objektmodell von System.Management

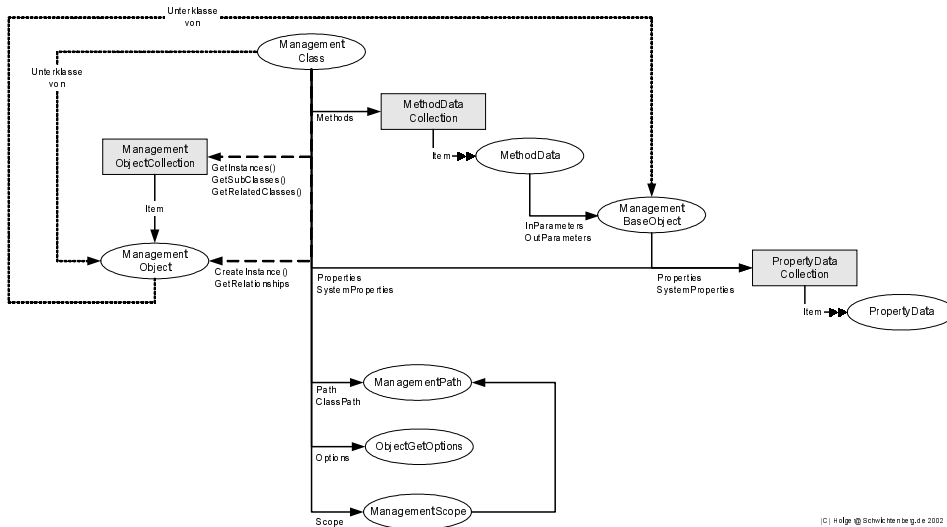


Abbildung 22.3: Zweiter Ausschnitt aus dem Objektmodell von System.Management

22.2.2 Objektbindung

WMI-Pfad Die Verwendung von WMI beginnt mit der Bindung eines Meta-Objekts (`ManagementObject` oder `ManagementClass`) an ein WMI-Objekt oder eine WMI-Klasse. Das WMI-Objekt bzw. die -Klasse wird über einen WMI-Pfad spezifiziert. Dieser Pfad hat die Form:

`\\Server\Namespace:Class.Key1=»wert1«,Key2=»Wert2«`

Mit Ausnahme von `Class` sind alle anderen Teile optional. Weitere Informationen über WMI-Pfade finden Sie in Anhang C.



Bitte beachten Sie, dass im Gegensatz zur Verwendung in COM die COM-Moniker-ID »WinMgmts:« nicht mehr vor den Pfad gehört.

Späte Fehlererkennung

Ein solcher Pfad kann dem Konstruktor der Klassen `ManagementObject` und `ManagementClass` übergeben werden. Damit existiert die Bindung dann zunächst nur auf dem WMI-Client. Die Anfrage an das lokale oder entfernte WMI-Repository wird erst ausgeführt beim ersten Zugriff auf ein WMI-Attribut, beim ersten Aufruf einer WMI-Methode oder wenn explizit die `Get()`-Methode des Meta-Objekts aufgerufen wird. Erst dann wird festgestellt, ob die angeforderte WMI-Klasse bzw. das angeforderte WMI-Objekt existiert. Lediglich grundsätzliche Fehler in der Syntax des Pfades werden direkt bei der Instanziierung erkannt.

```

Dim mo As Management.ManagementObject
Dim mc As Management.ManagementClass

' --- Bindung an eine lokale WMI-Klasse
mc = New ManagementClass("Win32_LogicalDisk")
mc.Get()
out("Gebunden an: " & mc.ToString)

' --- Bindung an ein entferntes WMI-Objekt
mo = New Management.ManagementObject _
    ("\\minbar\root\cimv2:Win32_LogicalDisk.DeviceID='c:'")
mo.Get()
out("Gebunden an: " & mo.ToString)

```

Listing 22.1: Ausschnitt aus *WMI_Bindung()* [*wmi.vb*]

Objektbindung mit Optionen

Bindungsoptionen

Es ist möglich, die verschiedenen Parameter für die Bindung zu setzen. Dies sind insbesondere:

- ▶ Benutzername und Kennwort eines Benutzerkontos, unter dem der Zugriff erfolgen soll (WMI unterstützt Impersonifizierung)
- ▶ Die DCOM-Optionen für Authentifizierung und Impersonifizierung (WMI verwendet DCOM zum Remoting)
- ▶ Eine Timeout-Zeit in Form eines `TimeSpan`-Objekts

Im klassischen WMI konnten diese Optionen zum Teil durch den WMI-Pfad festgelegt werden. Dies ist in der .NET-Implementierung von WMI nicht vorgesehen.

Die Bindung mit Optionen ist etwas aufwendiger.

Vorgehensweise

1. Man benötigt eine Instanz der Klasse `ConnectionOptions`, die die Optionen aufnimmt.
2. Über eine Instanz von `ManagementScope` definiert man den Server und den Namespace, den man verwenden will. Diesem Objekt übergibt man das `ConnectionOptions`-Objekt.
3. Die Klasse und die Instanz definiert man in einem `ManagementPath`-Objekt.
4. Dem Konstruktor von `ManagementObject` übergibt man das `ManagementScope`- und das `ManagementPath`-Objekt. Der dritte Parameter für weitere Optionen ist in der Regel auf `Nothing` zu setzen.

```

' --- Bindung mit Optionen
' Optionen definieren
Dim opt As New ConnectionOptions()
opt.Username = "minbar\Delen"
opt.Password = "egal"
opt.Authentication = AuthenticationLevel.PacketPrivacy
opt.Impersonation = ImpersonationLevel.Impersonate
opt.Timeout = New TimeSpan(0, 0, 4) ' 4 Sekunden
' Entfernten Computer definieren
Dim scope As New ManagementScope( _
    "\\minbar\root\cimv2")
scope.Options = opt
' Objekt definieren
Dim path As New _
ManagementPath("Win32_LogicalDisk.DeviceID='c:'")
' Objekt binden
mo = New Management.ManagementObject(scope, _
    path, Nothing)
mo.Get()
out("Gebunden an: " & mo.ToString)

```

Listing 22.2: Ausschnitt aus WMI_Bindung() [wmi.vb]

22.2.3 Zugriff auf Attribute und Methoden

Leider hat Microsoft auch bei WMI das Objektmodell stark verändert, ohne dass es dafür einen zwingenden Grund gibt. Nicht nur dass die Entwickler neue Klassen- und Member-Namen lernen müssen; die Art und Weise des Zugriffs auf die Attribute und Methoden eines WMI-Objekts ist dadurch wesentlich aufwendiger geworden, dass ein direkter Zugriff mit der Punkt-Notation (Objektname.Attributname) nur für die Members der Meta-Klasse, nicht aber für die Members des WMI-Objekts aus dem WMI-Repository möglich ist.

Attribute

Attribute lesen

Alle Lesezugriffe auf Attribute des WMI-Objekts müssen daher über String-Parameter erfolgen. Es gibt drei Möglichkeiten:

- ▶ über die Properties-Collection


```
mo.Properties("Attributname").Value
```
- ▶ über die Methode `GetPropertyValue()`

```
mo.GetPropertyValue("Attributname")
```
- ▶ oder – am einfachsten – über das `Item()`, ein indiziertes Attribut (Indexer)


```
mo("Attributname")
```


Für Schreibzugriffe gibt es äquivalent drei Möglichkeiten.

**Attribute
schreiben**

```
mo.Properties("Attributname").Value = "NeuerWert"
mo.SetPropertyValue("Attributname", "NeuerWert")
mo("Attributname") = "NeuerWert"
```

Methoden

Methodenaufrufe müssen über `InvokeMethod()` abgebildet werden.

Methodenaufrufe

```
mo.InvokeMethod("Methodenname", parameterarray)
```

Datentypen

Die folgende Tabelle zeigt die Umsetzung der Datentypen zwischen WMI und .NET.

WMI-Daten- typ	.NET-Datentyp (CLR-Datentyp)
Boolean	System.Boolean
Char16	System.Char
Sint8	System.Sbyte
Sint16	System.Int16
Sint32	System.Int32
Sint64	System.Int64
UInt8	System.Byte
UInt16	System.UInt16
UInt32	System.UInt32
UInt64	System.UInt64
String	System.String
Datetime	System.String
Object	System.Management.ManagementBaseObject
Real32	System.Single
Real64	System.Double
Reference	System.String

Tabelle 22.1: Entsprechung zwischen Datentypen in WMI und .NET

22.2.4 Lesezugriff

Die Klasse `Management.ManagementObject` ist die Meta-Klasse für ein einzelnes WMI-Objekt. Bei der Instanziierung wird über den Konstruktor ein WMI-Pfad angegeben, der die Instanz der Meta-Klasse an ein bestimmtes WMI-Objekt bindet.

**Management-
Object**

Beispiel

Zugriff auf Laufwerk

Das folgende Beispiel zeigt das Auslesen von Eigenschaften des Laufwerks »C«, das durch eine Instanz der WMI-Klasse `Win32_LogicalDisk` in WMI abgebildet wird. Diese Instanz wird durch folgenden WMI-Pfad eindeutig identifiziert:

```
\root\cimv2:Win32_LogicalDisk.DeviceID='C:'
```

Bei der Instanziierung wird eine Instanz der Meta-Klasse `Management.ManagementObject` mit Namen `mo` an das obige WMI-Objekt gebunden:

```
mo = New Management.ManagementObject(PFAD)
```

Mit dem Meta-Objektmodell ist ein direkter Zugriff nur auf die Mitglieder des Meta-Objekts möglich, also auf die Mitglieder von `mo`. Die Klasse `Management.ManagementObject` enthält selbst keine Daten, sondern nur Verweise auf Unterobjekte. So führt das Attribut `Path` zu einem Objekt des Typs `ManagementPath`, das allgemeine Informationen über das WMI-Objekt wie z.B. den relativen Pfad, den Computer, auf dem das Objekt liegt, und den WMI-Klassen-namen liefert.

```
out("Computer: " & mo.Path.Server)
```

Die Attribute des WMI-Objekts (also die Eigenschaften des Laufwerks in diesem Beispiel) kann man nur über String-Parameter ansprechen:

```
out("Name: " & mo("Caption"))
out("FileSystem: " & mo("FileSystem"))
out("Freie Bytes: " &
    System.Convert.ToInt32(mo("FreeSpace")).ToString)
```

Das folgende Listing zeigt das komplette Beispiel.

```
' =====
' Beispiel aus FCL-Buch.sln
' Zugriff auf ein bestimmtes WMI-Objekt
' (C) Holger@Schwichtenberg.de
' =====

Sub wmi_mol()
    Const PFAD = _
        "\root\cimv2:Win32_LogicalDisk.DeviceID='C:'"

    Dim mo As Management.ManagementObject

    ' --- Zugriff auf WMI-Objekt
    mo = New Management.ManagementObject(PFAD)
```

```

' --- Attribute des Meta-Objekts
out("-- Metainformation:")
out("kompletter WMI-Pfad:" & _
    mo.Path.Path)
out("Relativer Pfad:" & _
    mo.Path.RelativePath)
out("Computer: " & _
    mo.Path.Server)
out("Namespace:" & _
    mo.Path.NamespacePath)
out("Standardpfad: " & _
    mo.Path.DefaultPath.Path)
out("Klassenname: " & _
    mo.Path.ClassName)

' --- Attribute des WMI-Objekts
out("-- Zugriff auf einzelne Attribute")
out("Name: " & mo("Caption"))
out("Dateisystem: " & mo("FileSystem"))
out("Freie Bytes: " &
    System.Convert.ToUInt32(mo("FreeSpace")).ToString)
End Sub
    
```

Listing 22.3: Zugriff auf ein bestimmtes WMI-Objekt [WMI.vb]

22.2.5 Schreibzugriff

Das Vorgehen beim Schreibzugriff ist analog zum Lesezugriff. Zu beachten ist, dass nach einem Schreibzugriff die Änderungen erst wirksam werden, wenn die Methode `Put()` in der Meta-Klasse `ManagementObject` aufgerufen wird.

Beispiel

Im folgenden Beispiel wird die Laufwerksbezeichnung des Laufwerks »C« geändert.

```

' =====
' Beispiel aus FCL-Buch.sln
' Ändern von Attributwerten in einem WMI-Objekt
' (C) Holger@Schwichtenberg.de
' =====
Sub wmi_mo3()
    Const PFAD = _
        "\root\cimv2:Win32_LogicalDisk.DeviceID='c:'"
    Dim mo As Management.ManagementObject
    ' --- Zugriff auf WMI-Objekt
    mo = New Management.ManagementObject(PFAD)
    ' --- WMI-Attribute ändern
    mo("VolumeName") = "neuerName"
End Sub
    
```

```
' --- Änderungen schreiben
mo.Put()
out("Name geändert!")
End Sub
```

Listing 22.4: Ändern von Attributwerten in einem WMI-Objekt [WMI.vb]

22.2.6 Methodenaufrufe

InvokeMethod() Der Aufruf einer Methode in einem WMI-Objekt über das Meta-Objektmodell ist sehr komplex, weil zunächst alle Parameter einzeln in eine Datenstruktur eingefügt werden müssen, bevor die WMI-Methode mit `InvokeMethod()` aufgerufen werden kann. Als Datenstruktur kommt `InvokeMethod()` entweder mit einem Array of Object oder einem Objekt des Typs `ManagementBaseObject` klar. Im ersteren Fall liefert `InvokeMethod()` als Rückgabewert ein Objekt vom Typ `Object`, im zweiten Fall ein `ManagementBaseObject`.

Beispiel

Laufwerk prüfen Das Beispiel zeigt den Aufruf der Methode `CheckDisk()` in der WMI-Klasse `Win32_LogicalDisk`. Der Aufruf der Methode wird in zwei Varianten gezeigt.

Alternative 1 Die erste Alternative unter Verwendung von `ManagementBaseObject` ist deutlich länger: Zunächst muss ein passendes `ManagementBaseObject` mit dem Aufruf von `GetMethodParameters()` gewonnen werden. Dieses `ManagementBaseObject` besitzt Unterobjekte entsprechend den erwarteten Eingabeparametern. Der Rückgabewert wird ebenfalls als `ManagementBaseObject` geliefert.

Alternative 2 Dieser Weg ist deutlich länger als die Verwendung eines Arrays als Eingabeparameter für den Methodenaufruf. Jedoch hat der Weg über ein Array den Nachteil, dass die Parameter in der exakt richtigen Reihenfolge übergeben werden müssen. Außerdem bietet der erste Weg über `ManagementBaseObject` noch den Vorteil, dass beim Aufruf von `InvokeMethod()` als dritter Parameter optional eine Timeout-Zeit in Form eines Objekts des Typs `InvokeMethodOptions` mitgegeben werden kann.

```
' =====
' Beispiel aus FCL-Buch.sln
' Aufruf der Methode CheckDisk
' (C) Holger@Schwichtenberg.de
' =====
Sub wmi_mo2()
```

```

Dim mo As Management.ManagementObject
Dim inParams As _
    System.Management.ManagementBaseObject = Nothing
Dim outParams As _
    System.Management.ManagementBaseObject
Dim wert As UInt32

' --- Zugriff auf WMI-Objekt
mo = New Management.ManagementObject _
    ("//MARS\root\cimv2:Win32_LogicalDisk.DeviceID='f:'")
' --- Parameterliste einlesen
inParams = mo.GetMethodParameters("Chkdsk")
' --- Parameter füllen
inParams("FixErrors") = False
inParams("ForceDismount") = False
inParams("OkToRunAtBootUp") = False
inParams("RecoverBadSectors") = False
inParams("SkipFolderCycle") = False
inParams("VigorousIndexCheck") = False
' --- Methode aufrufen
out("Aufruf von CHKDSK...#1")
outParams = mo.InvokeMethod("Chkdsk", _
    inParams, Nothing)

' --- Rückgabewert ausgeben
wert = System.Convert._
    ToUInt32(outParams.Properties("ReturnValue").Value)
out("Ergebnis von CHKDSK: " & wert.ToString)

' --- Alternative mit Array of Object
Dim params() As Object = {False, False, False, _
    False, False, False}
Dim ergebnis As Object
out("Aufruf von CHKDSK...#2")
ergebnis = mo.InvokeMethod("Chkdsk", params)
out("Ergebnis von CHKDSK: " & ergebnis.ToString)
End Sub
    
```

Listing 22.5: Aufruf der Methode CheckDisk (Dieses Beispiel benötigt WMI 5.1.)
 [WMI.vb]

22.2.7 Informationen über WMI-Objekte

Oft ist es sinnvoll, sich alle Attribute und Methoden eines WMI-Objekts auflisten zu lassen. Das leistet das Listing 22.11.

Die Attribute können mit einer `For Each`-Schleife über die `PropertiesCollection` aufgelistet werden. Eine Liste der Methoden erhält man über die Klassendefinition. Das Meta-Objektmodell ermöglicht es, eine Verzeichnisdienst-Klasse zu binden. Dazu gibt es die spezielle

Properties

Meta-Klasse `Management.ManagementClass`. Die Klasse `ManagementClass` bietet eine **Collection** `Methods` an, in der es für jede Methode ein `MethodData`-Objekt gibt.

Von Management-Object zu ManagementClass

Das passende `ManagementClass`-Objekt ermittelt man aus einem `ManagementObject` über den WMI-Pfad:

```
Dim mc As ManagementClass _
    = New ManagementClass(mo.ClassPath.ClassName)
```

Ein Attribut in der Klasse `ManagementObject`, das direkt das zugehörige `ManagementClass`-Objekt liefert, gibt es leider nicht.

Beispiel

Die folgende Routine listet alle Attribute und Methoden der WMI-Klasse `Win32_LogicalDisk` auf.

```
' =====
' Beispiel aus FCL-Buch.sln
' Ausgabe aller Attribute und Methoden eines WMI-Objekts
' (C) Holger@Schwichtenberg.de
' =====
Sub wmi_mo4()
    Dim mo As Management.ManagementObject
    Dim mc As Management.ManagementClass
    Dim p As Management.PropertyData
    Dim m As MethodData
    Dim wert As Object
    ' --- Zugriff auf WMI-Objekt
    mo = New Management.ManagementObject _
        ("//MARS\root\cimv2:Win32_LogicalDisk.DeviceID='C:'")

    ' --- Alle Attribute des WMI-Objekts auflisten
    out("-- ALLE ATTRIBUTE:")
    For Each p In mo.Properties
        wert = p.Value
        If Not wert Is Nothing Then
            out(p.Name & "=" & wert.ToString())
        Else
            out(p.Name & "=" & (nicht belegt))
        End If
    Next

    ' --- Alle Methoden des Meta-Objekts auflisten
    out("-- ALLE METHODEN:")
    mc = New Management.ManagementClass _
        (mo.ClassPath.ClassName)
    For Each m In mc.Methods
        out(m.Name)
```

Next

End Sub

Listing 22.6: Ausgabe aller Attribute und Methoden eines WMI-Objekts [WMI.vb]

Ausgabe

Das Beispiel liefert folgende Ausgabe:

```

1  -- ALLE ATTRIBUTE:
2  Access= (nicht belegt)
3  Availability= (nicht belegt)
4  BlockSize= (nicht belegt)
5  Caption=C:
6  Compressed=False
7  ConfigManagerErrorCode= (nicht belegt)
8  ConfigManagerUserConfig= (nicht belegt)
9  CreationClassName=Win32_LogicalDisk
10 Description=Lokale Festplatte
11 DeviceID=C:
12 DriveType=3
13 ErrorCleared= (nicht belegt)
14 ErrorDescription= (nicht belegt)
15 ErrorMethodology= (nicht belegt)
16 FileSystem=NTFS
17 FreeSpace=1551724544
18 InstallDate= (nicht belegt)
19 LastErrorCode= (nicht belegt)
20 MaximumComponentLength=255
21 MediaType=12
22 Name=C:
23 NumberOfBlocks= (nicht belegt)
24 PNPDeviceID= (nicht belegt)
25 PowerManagementCapabilities= (nicht belegt)
26 PowerManagementSupported= (nicht belegt)
27 ProviderName= (nicht belegt)
28 Purpose= (nicht belegt)
29 QuotasDisabled=True
30 QuotasIncomplete=False
31 QuotasRebuilding=False
32 Size=5239468032
33 Status= (nicht belegt)
34 StatusInfo= (nicht belegt)
35 SupportsDiskQuotas=True
36 SupportsFileBasedCompression=True
37 SystemCreationClassName=Win32_ComputerSystem
38 SystemName=MARS
39 VolumeDirty=False
40 VolumeName=NeuerName3
41 VolumeSerialNumber=90CF0051
42 -- ALLE METHODEN:
```

```

43 SetPowerState
44 Reset
45 Chkdsk
46 ScheduleAutoChk
47 ExcludeFromAutochk

```

22.2.8 Auflisten von Objektmengen

WMI bietet nicht nur die Möglichkeit, einzelne Instanzen einer Klasse anzusprechen, sondern auch alle Instanzen einer Klasse. Dazu dienen die Klassen `ManagementClass` und `ManagementObjectCollection`.

- ManagementClass** ► Die `ManagementClass` repräsentiert eine WMI-Klasse und bietet die Methode `GetInstances()`.
- ManagementObjectCollection** ► Rückgabewert dieser Methode ist eine Objektmenge vom Typ `ManagementObjectCollection`.

Über `ManagementObjectCollection` kann mit `For Each` iteriert werden, wobei die einzelnen Objekte vom Typ `ManagementObject` sind.

Beispiel

Die folgende Routine listet alle Instanzen der WMI-Klasse `Win32_LogicalDisk` auf.

```

' =====
' Beispiel aus FCL-Buch.sln
' Auflisten aller Instanzen der WMI-Klasse
' Win32_LogicalDisk
' (C) Holger@Schwichtenberg.de
' =====
Sub wmi_Instanzen()
    Dim mo As Management.ManagementObject
    Dim mc As Management.ManagementClass
    Dim menge As ManagementObjectCollection

    ' --- Zugriff auf Klasse
    mc = New ManagementClass("Win32_LogicalDisk")
    ' --- Instanzen holen
    menge = mc.GetInstances
    ' --- Instanzen auflisten
    For Each mo In menge
        out(mo("Name"))
    Next
End Sub

```

Listing 22.7: Auflisten aller Instanzen der WMI-Klasse [WMI.vb]

22.3 Programmieren mit Wrapper-Klassen

Während die Arbeit mit dem Meta-Objektmodell noch die einzige Möglichkeit in .NET Beta1 war, hat Microsoft zur Beta2 die Rufe der geplagten WMI-Entwickler erhört und eine Möglichkeit geschaffen, WMI-Objekte auch mit früher Bindung zu nutzen. Der Vorgang heißt *Management Strongly Typed Class Generation*.

**Strongly Typed
Class Generation**

Die Klasse `ManagementClass` aus dem Meta-Objektmodell bietet eine Methode `GetStronglyTypedClassCode()`, die für eine beliebige WMI-Klasse den Quellcode für eine Wrapper-Klasse in Visual Basic .NET, C# oder JScript .NET erzeugt. Diese Wrapper-Klasse kann man dann seinen eigenen Projekten hinzufügen und fortan ist ein Zugriff auf die WMI-Klasse wie auf jede andere .NET-Klasse möglich. Im Inneren nutzt die Wrapper-Klasse das WMI-Meta-Objektmodell.

**GetStronglyTyped
ClassCode()**

Wohlgemerkt steigert dieses Vorgehen nicht die Performance: Dieser Weg ist keine neue Programmierschnittstelle zu WMI, sondern nur ein Code-Generator, der es dem Programmierer leichter macht.



22.3.1 Erzeugung einer Wrapper-Klasse

Es gibt zwei Wege, die Erzeugung einer Wrapper-Klasse anzustoßen:

1. per Programmcode
2. über das Kommandozeilen-Tool `mgmtclassgen.exe`

Erzeugung eines Wrappers per Programmcode

Ein Zweizeiler reicht: Bei der Instanziierung der `ManagementClass` wird die zu wrappende WMI-Klasse festgelegt. `GetStronglyTypedClassCode()` erwartet dann nur eine Sprachangabe und eine Ausgabedatei.

```
' =====
' Beispiel aus FCL-Buch.sln
' Diese Routine erzeugt eine früh bindende .NET-Klasse
' für eine WMI-Klasse
' (C) Holger@Schwichtenberg.de
' =====
Sub Wmi_direkt_erzeugen()
    Dim mc As New ManagementClass(Nothing, _
```

```

        "Win32_LogicalDisk", Nothing)
    mc.GetStronglyTypedClassCode(CodeLanguage.VB, _
        "H:\code\Logicaldisk.vb", String.Empty)
End Sub

```

Listing 22.8: Diese Routine erzeugt eine früh bindende .NET-Klasse für eine WMI-Klasse [WMI_direkt.vb]

Erzeugung eines Wrappers per Kommandozeilen-Tool

Der Management Strongly Typed Class Generator (*mgmtclassgen.exe*) ist ein kleines Kommandozeilen-Tool, das obigen Code ausführt. Das Werkzeug ist Teil des .NET SDKs und damit kostenlos.

Der folgende DOS-Kommandozeilenbefehl erzeugt eine Visual Basic .NET-Quellcode-Datei für eine Wrapper-Klasse für die WMI-Klasse *Win32_LogicalDisk*. Anzugeben sind

- ▶ der Name der WMI-Klasse,
- ▶ der Namespace, in dem sich die Klasse befindet,
- ▶ die Ausgabesprache (Kürzel »vb«, »cs« oder »js«)
- ▶ Name und Pfad der Ausgabedatei.

```

mgmtclassgen Win32_LogicalDisk /n root\cimv2 /l vb
/p Win32_logicaldisk.vb

```

Wrapper-Klasse für Win32_LogicalDisk

Das folgende Listing zeigt Ausschnitte aus der generierten Datei *Win32_LogicalDisk.vb*. Das Gesamtlisting ist 1924 Zeilen lang. Der generierte Code ist sehr ausführlich kommentiert.

```

Imports System
Imports System.ComponentModel
Imports System.Management
Imports System.Collections
Imports Microsoft.VisualBasic

Namespace ROOT.CIMV2.Win32
    'An Early Bound class generated for the WMI
    class Win32_LogicalDisk
        Public Class LogicalDisk
            Inherits System.ComponentModel.Component
            ...
            Public Sub New(ByVal keyDeviceID As String)

```

```

        Me.New(CType(Nothing, System.Management.ManagementScope),
        CType(New
        System.Management.ManagementPath(LogicalDisk.ConstructPath(key
        DeviceID)), System.Management.ManagementPath),
        CType(Nothing, System.Management.ObjectGetOptions))
    End Sub
    ...

    <Browsable(true), _

    DesignerSerializationVisibility(DesignerSerializationVisibilit
    y.Hidden), _
    Description("Die Eigenschaft "FreeSpace" zeigt an, wie
    viel freier Speicherplatz in Bytes auf " & _
    "dem logischen Datenträger verfügbar ist."), _
    TypeConverter(GetType(WMIValueTypeConverter))> _
    Public ReadOnly Property FreeSpace As System.UInt64
    Get
        If (curObj("FreeSpace") Is Nothing) Then
            Return System.Convert.ToUInt64(0)
        End If
        Return CType(curObj("FreeSpace"), System.UInt64)
    End Get
    End Property
    ...
    
```

Listing 22.9: Ausschnitt aus der generierten .NET-Klasse für die WMI-Klasse
Win32_LogicalDisk [Win32_LogicalDisk.vb]

22.3.2 Verwendung der erzeugten Klasse

Die erzeugte .NET-Klasse können Sie in ein beliebiges Projekt einbinden. Sehr hilfreich ist, dass die erzeugte Klasse auch einen Konstruktor bietet, der einfach den Laufwerksbuchstaben erwartet, um die Klasse zu lokalisieren. Der *Management Strongly Typed Class Generator* bedient sich des WMI-Schemas, um Informationen über die Schlüsselattribute der betreffenden WMI-Klasse zu bekommen. Damit kann der Generator einen entsprechenden Konstruktor aufbauen.

Es gibt aber auch noch andere Konstruktoren, z.B. zur Übergabe eines `ManagementScope`-Objekts und/oder eines `ManagementPath`-Objekts, um ein entferntes WMI-Objekt zu verwenden bzw. Verbindungsoptionen zu setzen (siehe Kapitel 1.2.2).

Überladungen des Konstruktors

- ▶ Sub New()
- ▶ Sub New(ByVal keyDeviceID As String)

- ▶ Sub New(ByVal mgmtScope As System.Management.ManagementScope, ByVal keyDeviceID As String)
- ▶ Sub New(ByVal path As System.Management.ManagementPath, ByVal getOptions As System.Management.ObjectGetOptions)
- ▶ Sub New(ByVal mgmtScope As System.Management.ManagementScope, ByVal path As System.Management.ManagementPath)
- ▶ Sub New(ByVal path As System.Management.ManagementPath)
- ▶ Sub New(ByVal mgmtScope As System.Management.ManagementScope, ByVal path As System.Management.ManagementPath, ByVal getOptions As System.Management.ObjectGetOptions)
- ▶ Sub New(ByVal theObject As System.Management.ManagementObject)
- ▶ Sub New(ByVal theObject As System.Management.ManagementBaseObject)

Beispiel

Dazu ein Beispiel, das die generierte Klasse `ROOT.CIMV2.LogicalDisk` verwendet:

```
' =====
' Beispiel aus FCL-Buch.sln
' Verwendung einer Wrapper-Klasse am Beispiel eines
' Laufwerks
' (C) Holger@Schwichtenberg.de
' =====
Sub wmi_direkt()
    Dim Disk As New ROOT.CIMV2.LogicalDisk("f:")
    out(Disk.Caption)
    out(Disk.Chkdsk(False, False, False, False, False, _
        False).ToString)
End Sub
```

*Listing 22.10: Verwendung einer Wrapper-Klasse am Beispiel eines Laufwerks
[WMI_direkt.vb]*

22.3.3 Verwendung der erzeugten Objektmengen

Es wird nicht nur eine Klasse für ein einzelnes WMI-Objekt, sondern auch eine passende Mengenklasse erzeugt. Diese Mengenklasse erhält den Namen der Einzelklasse mit dem angehängten Begriff »Collection«.

WMI-Klasse	Win32_LogicalDisk
.NET-Einzelklasse	LogicalDisk
.NET-Mengenklasse	LogicalDiskCollection

Die Einzelklasse bietet die statische Methode `GetInstances()`, um eine Menge der jeweiligen WMI-Objekte zu holen. `GetInstances()` ist mehrfach überladen:

GetInstances()

- ▶ `GetInstances()`
- ▶ `GetInstances(ByVal condition As String)`
- ▶ `GetInstances(ByVal selectedProperties() As System.String )`
- ▶ `GetInstances(ByVal condition As String, ByVal selectedProperties() As System.String )`
- ▶ `GetInstances(ByVal mgmtScope As System.Management.ManagementScope, ByVal enumOptions As System.Management EnumerationOptions)`
- ▶ `GetInstances(ByVal mgmtScope As System.Management.ManagementScope, ByVal condition As String)`
- ▶ `GetInstances(ByVal mgmtScope As System.Management.ManagementScope, ByVal selectedProperties() As System.String )`
- ▶ `GetInstances(System.Management.ManagementScope, ByVal condition As String, ByVal selectedProperties() As System.String )`

Sie können nach `GetInstances()` nicht direkt einen WMI-Pfad als String angeben. Um einen entfernten Computer als Ziel zu definieren, können Sie ein `ManagementScope`-Objekt übergeben. Eine Bedingung können Sie als String übergeben, z.B. liefert `GetInstances(New ManagementScope("\\mars"), "name>'d: '")` alle Laufwerke auf dem Rechner »MARS«, deren Laufwerksbuchstabe alphabetisch größer als »d« ist.



Beispiel

In der folgenden Routine wird eine Liste der Laufwerke eines Computersystems unter Verwendung der automatisch generierten Klassen `LogicalDisk` und `LogicalDiskCollection` ausgegeben.

```
' =====
' Beispiel aus FCL-Buch.sln
' Verwendung einer Wrapper-Klasse am Beispiel der Menge der
' Laufwerke
' (C) Holger@Schwichtenberg.de
' =====
```

```

Sub wmi_direkt_menge()
    Dim Disk As New ROOT.CIMV2.LogicalDisk()
    Dim menge As _
        ROOT.CIMV2.LogicalDisk.LogicalDiskCollection
    ' --- Menge holen
    menge = ROOT.CIMV2.LogicalDisk.GetInstances_
        (New ManagementScope("\\MARS"), "name>'d:'")
    ' --- Menge auflisten
    out("Liste der Laufwerke:")
    For Each Disk In menge
        out(Disk.Caption)
    Next
End Sub

```

Listing 22.11: Verwendung einer Wrapper-Klasse am Beispiel der Menge aller Laufwerke [WMI_direkt.vb]

22.4 WQL-Abfragen

WQL Die WMI Query Language (WQL) erlaubt es, Instanzen von WMI-Klassen auf Basis von Attributwerten zu selektieren. Dabei sind zwei Typen von Abfragen möglich:

► **Datenabfragen**

Eine Datenabfrage wählt unter allen Instanzen einer Klasse gemäß den angegebenen Kriterien aus.

► **Ereignisabfragen**

Diese Abfragen achten auf Veränderungen in WMI-Objekten oder WMI-Klassen.

Auch die Arbeit mit Abfragen in der WMI Query Language (WQL) hat sich in .NET gegenüber COM stark geändert. Gegenüber dem COM-Meta-Objektmodell gibt es nun eigene Klassen für die Ausführung von WQL-Abfragen.

Klassen

Zentrale Klassen für die Ausführung von WQL-Abfragen sind:

► **WqlEventQuery**

Diese Klasse definiert eine WQL-Ereignisabfrage.

► **WqlObjectQuery**

Diese Klasse definiert eine Abfrage nach WMI-Objekten oder WMI-Klassen.

► SelectQuery

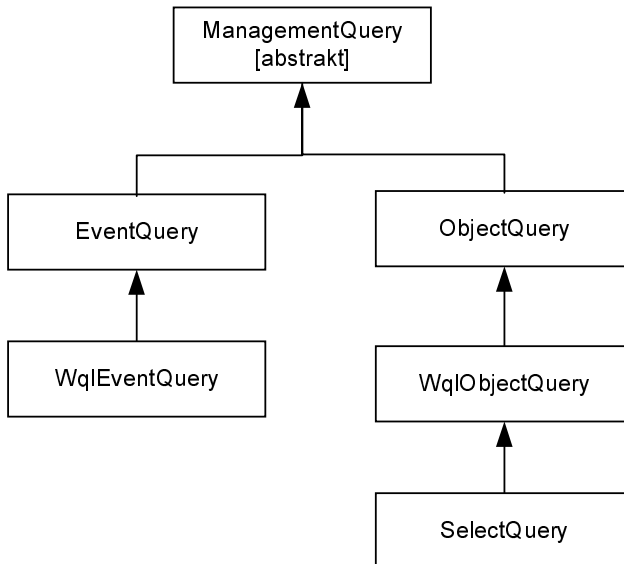
Diese Klasse ist eine Unterklasse von `WqlObjectQuery` mit speziellen Attributen und Methoden zur Erzeugung einer SQL-SELECT-Anweisung.

► ManagementObjectSearcher

Diese Klasse dient der Ausführung einer beliebigen WMI-Abfrage, wobei auch andere Sprachen als WQL möglich, derzeit aber nicht definiert sind.

Vererbungshierarchie

Die Vererbungshierarchie der verschiedenen Abfrageklassen beweist, dass die Integration anderer Abfragesprachen vorgesehen ist.



(C) Holger@Schwichtenberg.de 2002

Abbildung 22.4: Vererbungshierarchie der Abfrageklassen in System.Management

Objektmodell

Das Objektmodell zeigt den Zusammenhang der oben beschriebenen Klassen zur Laufzeit bzw. die Beziehung zum Kern des WMI-Meta-Objektmodells mit den Klassen `ManagementObject` und `ManagementObjectCollection`.

WQL	Erläuterung
SELECT IPAddress FROM Win32_NetworkAdapterConfiguration WHERE IPEnabled=TRUE	Das mehrwertige Attribut IPAddress einer Netzwerk- karte, die für das IP-Protokoll zugelassen ist
SELECT RecordNumber, Message FROM Win32_NTLogEvent WHERE Logfile='Application'	Eintragsnummer und Nachricht aller Einträge in das Application- Ereignisprotokoll

Tabelle 22.2: Beispiele für WQL-Datenabfragen (Fortsetzung)

Ausführung einer Datenabfrage in .NET

Um eine Datenabfrage auszuführen, gibt es zwei Wege:

- ▶ Der längere Weg ist, zunächst ein Objekt des Typs `SelectQuery` zu erzeugen und dies dann als Eingabe für ein Objekt vom Typ `ManagementObjectSearcher` zu verwenden.
- ▶ Alternativ kann direkt bei der Instanziierung der Klasse `ManagementObjectSearcher` eine WQL-Abfrage angegeben werden.

Danach muss mit der Methode `Get()` auf `ManagementObjectSearcher` die Abfrage gestartet werden. `Get()` liefert eine Objektmenge vom Typ `ManagementObjectCollection` zurück, durch die mit `For Each` iteriert werden kann.

Die folgenden Beispiele zeigen die Auflistung aller Dienste, die den Status »Gestartet« haben.

Beispiel I

Das erste Listing zeigt die Variante mit direkter Übergabe der WQL-Abfragezeichenkette an `ManagementObjectSearcher`.

```
' =====
' Beispiel aus FCL-Buch.sln
' WQL-Datenabfrage: Liste aller laufenden Dienste
' Variante #1
' (C) Holger@Schwichtenberg.de
' =====
Sub WMI_DataQuery1()
    out("WQL-Datenabfrage: Liste aller laufenden Dienste")

    Dim mo As ManagementObject
    Dim suche As New ManagementObjectSearcher()
    Dim menge As ManagementObjectCollection
```

```

    Const ABFRAGE = "select name,state from Win32_Service where
state='running'"
    ' --- Abfrage definieren
    suche = New ManagementObjectSearcher(ABFRAGE)
    ' --- Abfrage ausführen
    menge = suche.Get
    ' --- Ergebnisse ausgeben
    For Each mo In menge
        out("Dienst: " & mo("Name") & " Zustand: " & _
            mo("state"))
    Next
End Sub

```

Listing 22.12: WQL-Datenabfrage: Liste aller laufenden Dienste, Variante #1
[WMI_Abfragen.vb]

Beispiel 2

Das zweite Listing zeigt die Variante unter Verwendung der Klasse `SelectQuery`.

```

' =====
' Beispiel aus FCL-Buch.sln
' WQL-Datenabfrage: Liste aller laufenden Dienste
' Variante #2
' (C) Holger@Schwichtenberg.de
' =====
Sub WMI_DataQuery2()
    out("WQL-Datenabfrage: Liste aller laufenden Dienste")

    Dim mo As ManagementObject
    Dim suche As New ManagementObjectSearcher()
    Dim menge As ManagementObjectCollection
    Dim sq As SelectQuery

    Const ABFRAGE = "select name,state from Win32_Service where
state='running'"
    ' --- Abfrage definieren
    sq = New SelectQuery(ABFRAGE)
    suche = New ManagementObjectSearcher(sq)
    ' --- Abfrage ausführen
    menge = suche.Get
    ' --- Ergebnisse ausgeben
    For Each mo In menge
        out("Dienst: " & mo("Name") & " Zustand: " & _
            mo("state"))
    Next
End Sub

```

Listing 22.13: WQL-Datenabfrage: Liste aller laufenden Dienste, Variante #2
[WMI_Abfragen.vb]

22.4.2 Ereignisabfragen

Eine Ereignisabfrage (Event Query) unterscheidet sich von einer Datenabfrage dadurch, dass sie nicht sofort ein Ergebnis liefert, sondern Veränderungen an WMI-Objekten durch Ereignisse zurückmeldet.

Aufbau einer Ereignisabfrage

Eine WMI-Ereignisabfrage bezieht sich immer auf einen WMI-Ereignistyp. Mit der `WHERE`-Klausel werden das zu beobachtende WMI-Objekt und die in dem Objekt zu beobachtenden Attribute definiert.

Erweiterte WQL-Syntax

```
SELECT * FROM eventklasse WHERE bedingung
```

Ereignisabfragen unterstützen als zusätzliche SQL-Schlüsselwörter `WITHIN`, `GROUP BY` und `HAVING`:

- ▶ Dabei gibt `WITHIN 10` das Polling-Intervall in Sekunden an.
- ▶ `GROUP` bündelt eine Anzahl von Einzelereignissen zu einem Gesamtereignis.
- ▶ `HAVING` dient der Definition einer Bedingung innerhalb der Gruppierung.

Ereignistypen in WMI

Die folgende Grafik zeigt die in WMI definierten Ereignistypen. Diese sind in einer Vererbungshierarchie angeordnet.

Bitte beachten Sie, dass das Ereignis `__InstanceModificationEvent` wirklich nur ausgeführt wird, wenn sich ein Attributwert ändert. Wenn Sie beispielsweise die Prozessorlast auf die Überschreitung der 80%-Grenze prüfen, dann bekommen Sie ein Ereignis beim Überschreiten der Grenze. Wenn danach der Wert konstant bei 100% liegt, bekommen Sie keine weiteren Ereignisse. Sie erhalten erst wieder ein Ereignis, wenn der Wert sich nochmals ändert (z.B. von 100% auf 99%).



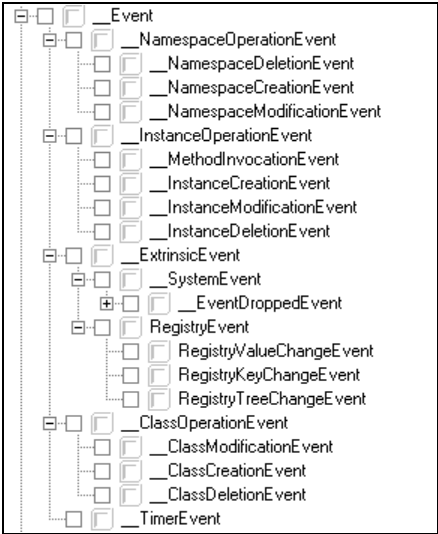


Abbildung 22.6: Ereignistypen in WMI 5.1
(angezeigt mit dem Werkzeug »CIM Studio«)

**Zusätzliche
Schlüsselwörter**

Beispiele für Ereignisabfragen

Die folgende Tabelle zeigt Beispiele für WMI-Ereignisabfragen.

WQL	Erläuterung
<pre>SELECT * FROM __InstanceModificationEvent WITHIN 5 WHERE TargetInstance ISA "Win32_Service" AND TargetIns- tance.State="Stopped"</pre>	Alle fünf Sekunden wird geprüft, ob ein Dienst den Status <i>Stopped</i> bekommen hat.
<pre>SELECT FROM EmailEvent GROUP WITHIN 600 HAVING NumberOfEvents > 5</pre>	Wenn innerhalb von zehn Minuten mehr als fünf E-Mail-Ereignisse auftreten, wird dieses Ereignis ausgelöst.
<pre>SELECT * FROM __InstanceCreationEvent WHERE TargetInstance ISA "Win32_NTLogEvent" AND Target Instance.Logfile="Application" OR TargetInstance.Logfile="System"</pre>	Jeder neue Eintrag in den Ereignisprotokollen <i>System</i> und <i>Application</i> löst ein Event aus.

Tabelle 22.3: Beispiele für Event Queries

Ausführung einer WMI-Ereignisabfrage in .NET

Da die Ergebnisse als Ereignisse eintreffen, unterscheidet sich die Ausführung einer Ereignisabfrage natürlich von der Ausführung einer Datenabfrage.

Die Verwendung einer Ereignisabfrage sieht so aus:

- ▶ Zunächst wird ein `EventQuery`-Objekt erzeugt.
`Dim eq As New EventQuery(ABFRAGE)`
- ▶ Zusätzlich wird ein `EventWatcher` benötigt, den das `EventQuery`-Objekt verwendet.
`Dim watcher As New ManagementEventWatcher(eq)`
- ▶ Der `EventWatcher` bietet ein Ereignis `EventArrived()`, das an eine Ereignisbehandlungsroutine gebunden werden muss.
`AddHandler watcher.EventArrived, AddressOf EreignisBehandlung`
- ▶ Dann wird der `EventWatcher` gestartet.
`watcher.Start()`
- ▶ Während das Hauptprogramm wartet, ruft der `EventWatcher` die entsprechende Ereignisbehandlungsroutine unter Angabe der Ereignisdaten auf.
`Public Sub EreignisBehandlung(ByVal sender As Object, ByVal e As EventArrivedEventArgs)`

Beispiel

Im folgenden Listing werden Zustandsänderungen in den Windows-Diensten überwacht.

```
' =====
' Beispiel aus FCL-Buch.sln
' Ereignisabfrage für Dienste
' (C) Holger@Schwichtenberg.de
' =====
Sub wmi_ereignisabfrage()

    Const ABFRAGE = "SELECT * FROM __InstanceModificationEvent
    WITHIN 1 WHERE TargetInstance ISA 'Win32_Service'"

    ' --- Abfrage definieren
    Dim eq As New EventQuery(ABFRAGE)
    ' --- Überwachung definieren
    Dim watcher As New ManagementEventWatcher(eq)
    ' --- Ereignisbehandlungsroutine festlegen
```

```

AddHandler watcher.EventArrived, _
    AddressOf EreignisBehandlung
' --- Überwachung starten
watcher.Start()

MsgBox("Klicken Sie hier, um Überwachung zu beenden!")

watcher.Stop()

End Sub

Public Sub EreignisBehandlung(ByVal sender As Object, ByVal e
As EventArgs)
    Dim mo As ManagementBaseObject
    Dim p As PropertyData
    Dim wert As Object
    mo = e.NewEvent

    out("Veränderung in Dienst: " & mo.ClassPath.Path)
    For Each p In mo.Properties
        wert = p.Value
        If Not wert Is Nothing Then
            out(p.Name & "=" & wert.ToString())
        Else
            out(p.Name & "= (nicht belegt)")
        End If
    Next

End Sub

```

Listing 22.14: Ereignisabfrage für Dienste [WMI_Abfragen.vb]

22.5 WMI im Visual Studio .NET Server Explorer

Management (WMI) Extensions for VS.NET Server Explorer

Der Server Explorer in Visual Studio .NET ist ein Werkzeug, um verschiedene Systembausteine (z.B. Ereignisprotokolle, Dienste, Leistungsindikatoren, Datenbanken) zu betrachten, zu verändern (z.B. starten, stoppen, leeren) und per Drag&Drop in eigene Anwendungen zu integrieren.

In der Beta1 von Visual Studio .NET gab es auch noch einen Zweig für WMI-Klassen. Dieser ist nun ein kostenloses Add-on für Visual Studio .NET mit dem Namen »Management (WMI) Extensions for VS.NET Server Explorer« [CD:/Install/VS.NET Add on/VS.NET Server Explorer Extensions for WMI].

Nach der Installation dieses Add-ons gibt es im Server Explorer zwei neue Äste: MANAGEMENT CLASSES und MANAGEMENT EVENTS.

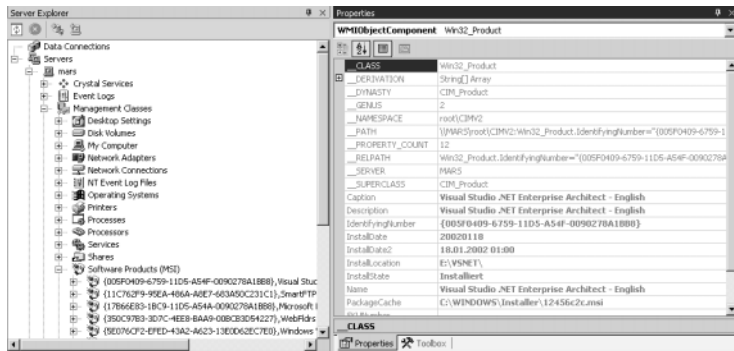


Abbildung 22.7: WMI-Erweiterungen im Server Explorer

MANAGEMENT CLASSES zeigt eine vordefinierte Menge von WMI-Klassen an. Sie können jede andere Klasse dort integrieren (Kontextmenüpunkt ADD). Das Eigenschaftsfenster zeigt die Attribute der Klasse. Dabei stehen auch die Beschreibungstexte, die im WMI-Repository hinterlegt sind, zur Verfügung.

Klassen

Unterhalb jeder Klasse kann man die Instanzen einblenden; das Eigenschaftsfenster zeigt dann die Attributwerte der gewählten Instanz. Im Kontextmenü können die Methoden der WMI-Klasse auf dem betreffenden Objekt aufgerufen werden.

Instanzen

Der Zweig MANAGEMENT EVENTS dient dazu, WMI-Ereignisabfragen zu definieren (siehe Abbildung 22.8).

WMI-Ereignisse

Nutzung im Designer

Visual Studio .NET besitzt auch einen Designer für nicht-visuelle Komponenten im Rahmen der VS.NET-Elementtypen WINDOWS FORM, WEB FORM oder COMPONENT. Jeder per Drag&Drop auf den Designer abgelegte Systembaustein erscheint als ein Icon mit Namen und kann über das Eigenschaftsfenster konfiguriert werden.

Nicht-visuelle Komponenten

Da weder die Anordnung der Elemente konfiguriert werden kann, noch eine weitere grafische Nutzung (z.B. Verbindung von Attributen durch Linien) möglich ist, ist der Begriff »Designer« hier aber etwas hochgegriffen.

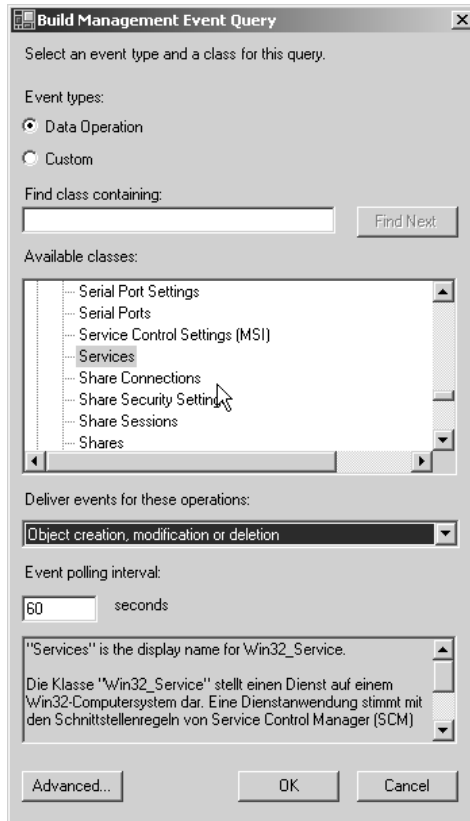


Abbildung 22.8: Definition einer WMI-Ereignisabfrage im Server Explorer



Keine Design-Time-Controls

Automatische Klassen-generierung

Auch WMI-Klassen, die visuelle Systembausteine repräsentieren, werden wie nicht-visuelle Systembausteine behandelt, da es bisher keine besonderen Design-Time-Controls für WMI-Klassen gibt.

Neben der Möglichkeit, die Attribute der Instanz zu betrachten, liegt der Vorteil darin, dass der Server Explorer automatisch die *Management Strongly Typed Class Generation* anstößt und den Code für die Instanziierung generiert. Dieser Code wird in einer Region mit Namen »Component Designer Generated Code« zusammengefasst. Aus dem eigenen Programmcode heraus kann der Entwickler dann die WMI-Instanz über den im Designer vergebenen Namen ansprechen.



Abbildung 22.9: Vier WMI-Instanzen im Designer [WMI_visuell.vb]

Beispiel

Das folgende Listing zeigt den vom Visual Studio .NET-Designer für die vier in Abbildung 22.9 dargestellten WMI-Instanzen generierten Programmcode. Außerdem wird anhand der selbst geschriebenen Routine `test_printer()` gezeigt, wie man die `Printer_HP` genannte Instanz der Klasse `CIMV2.Printer` nutzen kann. Die Routine druckt erst die Testseite aus und gibt den Drucker dann als Netzdrucker frei.

```
' =====
' Beispiel aus FCL-Buch.sln
' Drucken einer Testseite und Freigeben des Druckers
' Variante #1
' (C) Holger@Schwichtenberg.de
' =====

Public Class WMI_visuell
    Inherits System.ComponentModel.Component

    #Region " Component Designer generated code "

    Public Sub New(Container As System.ComponentModel.IContainer)
        MyClass.New()
```

```

'Required for Windows.Forms Class Composition Designer
support
    Container.Add(me)
End Sub

Public Sub New()
    MyBase.New()

    'This call is required by the Component Designer.
    InitializeComponent()

    'Add any initialization after the InitializeComponent() call

End Sub

'Component overrides dispose to clean up the component list.
Protected Overloads Overrides Sub Dispose(ByVal disposing As
Boolean)
    If disposing Then
        If Not (components Is Nothing) Then
            components.Dispose()
        End If
    End If
    MyBase.Dispose(disposing)
End Sub

'Required by the Component Designer
Private components As System.ComponentModel.IContainer

'NOTE: The following procedure is required by the Component
Designer
'It can be modified using the Component Designer.
'Do not modify it using the code editor.
Friend WithEvents AppLog As
FCL_Buch.ROOT.CIMV2.NTEventLogFile
Friend WithEvents Laufwerk_C As
FCL_Buch.ROOT.CIMV2.LogicalDisk
Friend WithEvents Printer_HP As FCL_Buch.ROOT.CIMV2.Printer
Friend WithEvents Fax_Dienst As FCL_Buch.ROOT.CIMV2.Service
<System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()
    Me.AppLog = New FCL_Buch.ROOT.CIMV2.NTEventLogFile()
    Me.Laufwerk_C = New FCL_Buch.ROOT.CIMV2.LogicalDisk()
    Me.Printer_HP = New FCL_Buch.ROOT.CIMV2.Printer()
    Me.Fax_Dienst = New FCL_Buch.ROOT.CIMV2.Service()
    '
    'AppLog
    Me.AppLog.Path = New
System.Management.ManagementPath("\\MARS\root\CIMV2:Win32_NTEv
entLogFile.Name=""C:\WINDOWS\system32\config\AppEv" & _
"ent.Evt""")

```

```

'
' Laufwerk_C
Me.Laufwerk_C.Path = New
System.Management.ManagementPath("\\MARS\root\CIMV2:Win32_
LogicalDisk.DeviceID=""C: """)
'
' Printer_HP
Me.Printer_HP.Path = New
System.Management.ManagementPath("\\MARS\root\CIMV2:Win32_
Printer.DeviceID=""HP2100 """)
'
' Fax_Dienst
Me.Fax_Dienst.Path = New
System.Management.ManagementPath("\\MARS\root\CIMV2:Win32_
Service.Name=""Fax """)

End Sub

#Region "Eigener Code von Holger Schwichtenberg"
Sub test_printer()
Dim name As String = Printer_HP.Name
out("Drucke Testseite auf " & name)
Printer_HP.PrintTestPage()
out("Testseite gedruckt!")

out("Drucker wird freigegeben")
Printer_HP.Shared = True
Printer_HP.ShareName = "HP2100"
out("Drucker wurde freigegeben!")
End Sub
#End Region

```

Listing 22.15: [WMI_visuell.vb]

Ein Aufruf der Methode `test_printer()` liefert eine Ausgabe der **Ausgabe** Form

```

1  Drucke Testseite auf HP2100
2  Testseite gedruckt!
3  Drucker wird freigegeben
4  Drucker wurde freigegeben!

```

Hinweise

Das WMI-Add-in für den Server Explorer war bei Drucklegung dieses Buchs an einigen Stellen ebenso wenig ausgereift wie der visuelle Designer. Der Designer verschluckt manchmal die Icons von hinzugefügten Objekten. Das Add-in generiert eine Quell-codetei mit der Klasse für den Direktzugriff immer direkt im Wurzelverzeichnis des Projekts. Wenn man die Datei in ein unter-

Bugs

geordnetes Verzeichnis verschiebt, generiert das Add-in bei der nächsten Verwendung eine neue Quellcodedatei im Wurzelverzeichnis. Dadurch ist die Klasse dann doppelt definiert und der Compiler streikt wegen eines Namenskonflikts.

22.6 Weitere Beispiele

Dieses Kapitel enthält ausgewählte Beispiele zur Verwendung von WMI-Klassen. In allen Fällen wird der Direktzugriff über die generierten Klassen verwendet.

22.6.1 Abmelden, Herunterfahren und Neustart

Reboot Mit der Methode `Reboot()` in der Klasse `Win32_OperatingSystem` kann ein System heruntergefahren werden.

Auf jedem Rechner (also in jedem WMI-Repository) gibt es nur eine Instanz von `Win32_OperatingSystem`. Diese zu binden ist aber nicht einfach: Wenn Sie den genauen Namen einer Instanz nicht kennen (dieser ist im Fall von `Win32_OperatingSystem` kompliziert und setzt sich aus dem Namen der Windows-Version, dem Pfad zum Windows-Verzeichnis und dem Namen der Start-Partition zusammen), können Sie nicht direkt an das einzelne Objekt binden. Die Lösung besteht darin, alle Instanzen der Klasse anzusprechen. Sie iterieren dann mit `For Each`, auch wenn es, wie in diesem Fall, nur genau eine Instanz geben kann. Mit `Win32shutdown(0,0)` wird der aktuelle Benutzer abgemeldet, mit `Shutdown()` der Rechner heruntergefahren.

```
' =====
' Beispiel aus FCL-Buch.sln
' Informationen über ein Betriebssystem und Reboot
' via generierte Klasse
' (C) Holger@Schwichtenberg.de
' =====
Sub WMI_ComputerReboot1()

    Dim os As New OperatingSystem()
    Dim col As OperatingSystem.OperatingSystemCollection

    ' --- Entfernten Computer festlegen
    Dim sc As New ManagementScope("\\minbar\root\CIMV2")
    ' --- "Menge" der OS-Objekte holen
    col = OperatingSystem.GetInstances(sc, "")
    ' --- Schleife über alle OS-Objekte (genau 1!)
```

```

For Each os In col
    out("Informationen über das Betriebssystem auf: " & _
        & os.Path.Server)
    out("Name: " & os.Name)
    out("Hersteller: " & os.Manufacturer)
    out("Typ: " & os.OSType)
    out("Sprache: " & os.OSLanguage.ToString)
    out("Version: " & os.Version)
    out("Systemverzeichnis: " & os.SystemDirectory)
    out("Registrierter Benutzer: " & os.RegisteredUser)
    os.Reboot()
    'os.Win32Shutdown(0, 0)
    'os.Shutdown()
Next

```

End Sub

Listing 22.16: Informationen über ein Betriebssystem und Reboot via generierte Klasse [WMI_Beispiele.vb]

Zum Vergleich soll hier auch noch einmal die Implementierung ohne die generierte Klasse gezeigt werden.

```

' =====
' Beispiel aus FCL-Buch.sln
' Informationen über ein Betriebssystem und Reboot
' via spätes Binden
' (C) Holger@Schwichtenberg.de
' =====
Sub WMI_ComputerReboot2()

    Dim mo As Management.ManagementObject
    Dim mc As Management.ManagementClass
    Dim menge As Management.ObjectCollection

    ' --- Zugriff auf Klasse
    mc = New ManagementClass_
        ("\\minbar\root\CIMV2:Win32_OperatingSystem")
    ' --- Instanzen holen
    menge = mc.GetInstances

    For Each mo In menge
        out("Informationen über das Betriebssystem auf: " & _
            & mo.Path.Server)
        out("Name: " & mo("Name"))
        out("Hersteller: " & mo("Manufacturer"))
        out("Typ: " & mo("OSType").ToString)
        out("Sprache: " & mo("OSLanguage").ToString)
        out("Version: " & mo("Version"))
        out("Systemverzeichnis: " & mo("SystemDirectory"))
        out("Registrierter Benutzer: " & mo("RegisteredUser"))
    
```

```

out("Reboot wird initiiert...")
mo.InvokeMethod("reboot", Nothing)
out("Reboot eingeleitet!")
Next

End Sub

```

Listing 22.17: Informationen über ein Betriebssystem und Reboot via spätes Binden

Ausgabe

```

Informationen über das Betriebssystem auf: MINBAR
Name: Microsoft Windows 2000
Professional|C:\WINNT|\Device\Harddisk0\Partition1
Hersteller: Microsoft Corporation
Typ: 18
Sprache: 1031
Version: 5.0.2195
Systemverzeichnis: C:\WINNT\System32
Registrierter Benutzer: Holger Schwichtenberg

```

22.6.2 Netzwerkverbindungen

Win32_Network- Connection

Die folgende Routine verwendet die Klasse Win32_NetworkConnection, um den Status der Netzwerkverbindungen auszugeben. Dabei wird durch GetInstances() auf die Verbindungen eingeschränkt, die als »persistent« gekennzeichnet sind.

```

' =====
' Beispiel aus FCL-Buch.sln
' Status der persistenten Netzwerkverbindungen auflisten
' (C) Holger@Schwichtenberg.de
' =====

Sub WMI_Netzwerkverbindungen()
    Dim verb As New ROOT.CIMV2.NetworkConnection()
    Dim menge As _
    ROOT.CIMV2.NetworkConnection.NetworkConnectionCollection
    menge = verb.GetInstances("Persistent=true")
    For Each verb In menge
        out(verb.Name & "=" & verb.Status)
    Next
End Sub

```

*Listing 22.18: Status der persistenten Netzwerkverbindungen auflisten
[WMI_Beispieler.vb]*

Ausgabe

```

1  \\Sonne2000\DEVTOOLS (L:)=OK
2  \\Sonne2000\MEGAPLATTE (M:)=OK
3  \\Sonne2000\AlteDaten (N:)=OK
4  \\Sonne2000\ITV.DE_Web (W:)=OK
5  \\Sonne2000\c$ (x:)=OK
6  \\minbar\f$ (y:)=Degraded
7  \\minbar\e$ (z:)=Degraded

```

22.6.3 Speichernutzung

Über die Klasse `Win32_Process` bietet WMI ähnliche Informationen und Methoden wie die FCL-Klasse `System.Diagnostics.Process`.

```
' =====
' Beispiel aus FCL-Buch.sln
' Ausgabe der Speichernutzung der laufenden Prozesse
' (C) Holger@Schwichtenberg.de
' =====
Sub WMI_Speichernutzung1()

    Dim P As ROOT.CIMV2.Process
    Dim PMenge As ROOT.CIMV2.Process.ProcessCollection

    PMenge = P.GetInstances()

    out("Speichernutzung der laufenden Prozesse:")
    For Each P In PMenge
        out(P.Name & ": " & _
            Convert.ToDouble(P.WorkingSetSize) / 1024 & " KB")
    Next

End Sub
```

*Listing 22.19: Ausgabe der Speichernutzung der laufenden Prozesse
[WMI_Beispiele.vb]*

Alternative

Um zu demonstrieren, dass es in WMI oft mehrere Wege gibt, sei hier noch gezeigt, dass man fast die gleichen Informationen auch über die Klasse `Win32_PerfRawData_PerfProc_Process` erhalten kann. Die Unterschiede sind folgende:

- ▶ Es gibt auch eine `Win32_PerfRawData_PerfProc_Process`-Instanz mit Namen `_Total`, die die aggregierten Daten aller Prozesse enthält.
- ▶ Das Attribut `WorkingSetSize` heißt hier einfach nur `WorkingSet`.
- ▶ Das Attribut `Name` enthält die Kurzform des Prozessnamens (ohne `.exe`).

WMI ermöglicht über den Performance Counter Provider Zugriff auf zahlreiche Leistungsdaten des Windows-Systems. Wenn Sie die `PerfRawData`-Klassen nicht finden, starten Sie den WMI-Dienst einmalig manuell an der Kommandozeile mit `Winmgmt /resyncperf`.

**Win32_PerfRaw
Data_PerfProc_
Process**



```
' =====
' Beispiel aus FCL-Buch.sln
' Ausgabe der Speichernutzung der laufenden Prozesse
' (C) Holger@Schwichtenberg.de
' =====

Sub WMI_Speichernutzung2()

    Dim P As ROOT.CIMV2.PerfRawData_PerfProc_Process
    Dim PMenge As ROOT.CIMV2.PerfRawData_PerfProc_Process.
        PerfRawData_PerfProc_ProcessCollection

    PMenge = P.GetInstances()

    out("Speichernutzung der laufenden Prozesse:")
    For Each P In PMenge
        out(P.Name & ": " & _
            Convert.ToDouble(P.WorkingSet) / 1024 & " KB")
    Next

End Sub
```

Listing 22.20: Speichernutzung der aktiven Prozesse [WMI_Beispiele.vb]

WMI_Speichernutzung1	WMI_Speichernutzung2
Liste via Win32_Process	Liste via Win32_PerfRawData_PerfProc_Process
Speichernutzung der laufenden Prozesse: System Idle Process: 20 KB notepad.exe: 2156 KB FCL-Buch.exe: 19052 KB ...	Speichernutzung der laufenden Prozesse: Idle: 20 KB notepad: 2156 KB FCL-Buch: 19092 KB ... _Total: 264304 KB

Tabelle 22.4: Gegenüberstellung der Ausgaben der obigen beiden Varianten zur Ausgabe der Speichernutzung

22.6.4 ASP.NET-Leistungsdaten

Die WMI-Klasse Win32_PerfFormattedData_AspNet_AspNetApplications enthält Leistungsdaten über ASP.NET.


```
' =====
' Beispiel aus FCL-Buch.sln
' Ausgabe der Performance-Daten von ASP.NET
' (C) Holger@Schwichtenberg.de
' =====

Sub WMI_ASPPerf()
    Dim pfad As New
ManagementPath("\\MARS\root\CIMV2:Win32_PerfFormattedData_ASP-
NET_AspNetApplications.Name="__Total__")
    Dim asp As New ROOT.CIMV2.PerfFormattedData_AspNet_
AspNetApplications(pfad)
    out("Anfragen Gesamt: " & asp.RequestsTotal.ToString)
    out("davon erfolgreich: " & _
        asp.RequestsSucceeded.ToString)
    out("Aktuelle Sitzungen: " & _
        asp.SessionsActive.ToString)
    out("Abgelaufene Sitzungen: " & _
        asp.SessionsTimedOut.ToString)
End Sub
```

Listing 22.21: Performance-Daten über Active Server Pages [WMI_Beispieler.vb]

```
1  Anfragen Gesamt: 7
2  davon erfolgreich: 7
3  Aktuelle Sitzungen: 1
4  Abgelaufene Sitzungen: 0
```

Ausgabe

WMI ermöglicht über den Performance Counter Provider Zugriff auf zahlreiche Leistungsdaten des Windows-Systems. Wenn Sie die PerfRaw Data-Klassen nicht finden, starten Sie den WMI-Dienst einmalig manuell an der Kommandozeile mit `Winmgmt /resyncperf`.



A .NET-Glossar

Dieses Glossar enthält die wichtigsten Begriffe rund um das .NET Framework. Es kann keine in die Grundprinzipien des .NET Frameworks einführende Lektüre ersetzen, wohl aber eine Hilfe zur Klärung ausgewählter Begriffe sein.

.NET

Der Begriff .NET (DOTNET) umfasst drei verschiedene Dinge:

DOTNET

1. Erstens ist .NET ein neues technisches Konzept für die Anwendungsentwicklung unter Windows (das .NET Framework).
2. Zum Zweiten ist .NET ein Marketing-Begriff für die neuesten Version von Betriebssystemen, Serverprodukten und Anwendungen aus dem Hause Microsoft.
3. Zum Dritten steht der Begriff für internetbasierte Dienstleistungen (Webservices) von Microsoft.



Abbildung A.1: Das Microsoft .NET-Logo

Microsoft spricht auch von der .NET-Initiative, die sich in offiziellen Präsentationen in der Regel aus folgenden sechs Teilbereichen zusammensetzt:

.NET-Initiative

1. .NET Framework
2. .NET Enterprise Server
3. .NET Device Software
4. .NET My Services (»Hailstorm«)
5. .NET Tools (Visual Studio .NET)
6. .NET User Experience

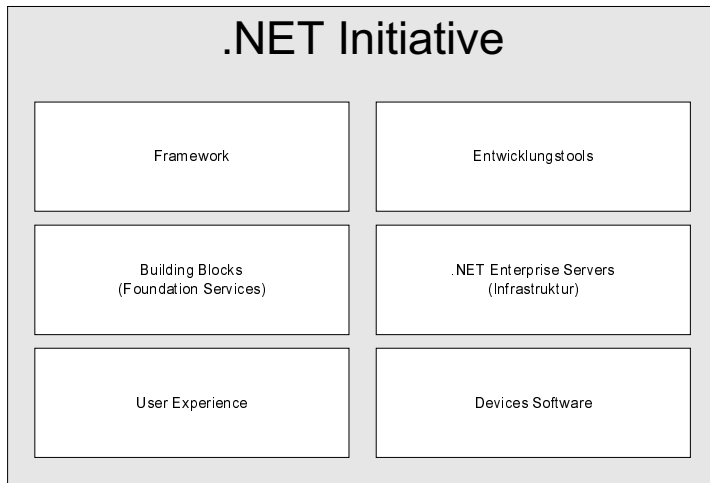


Abbildung A.2: Teilbereiche der .NET-Initiative



Geschäftsprozess- integration

Microsoft verkauft .NET in vielen Präsentationen als eine Architektur zur Geschäftsprozessintegration. Dies beruht sehr stark auf der Unterstützung von XML und SOAP im .NET Framework einerseits und dem Microsoft Biztalk Server andererseits, der eines der als .NET Enterprise Server bezeichneten Produkte ist.

CF .NET Compact Framework (CF)

siehe Compact Framework

.NET Data Provider

Managed Provider

Ein .NET Data Provider ist ein Datenquellen-Treiber für ADO.NET. Ein Data Provider entspricht von der Rolle her einem OLEDB-Provider in COM. Alias: Managed Provider

.NET Enterprise Services

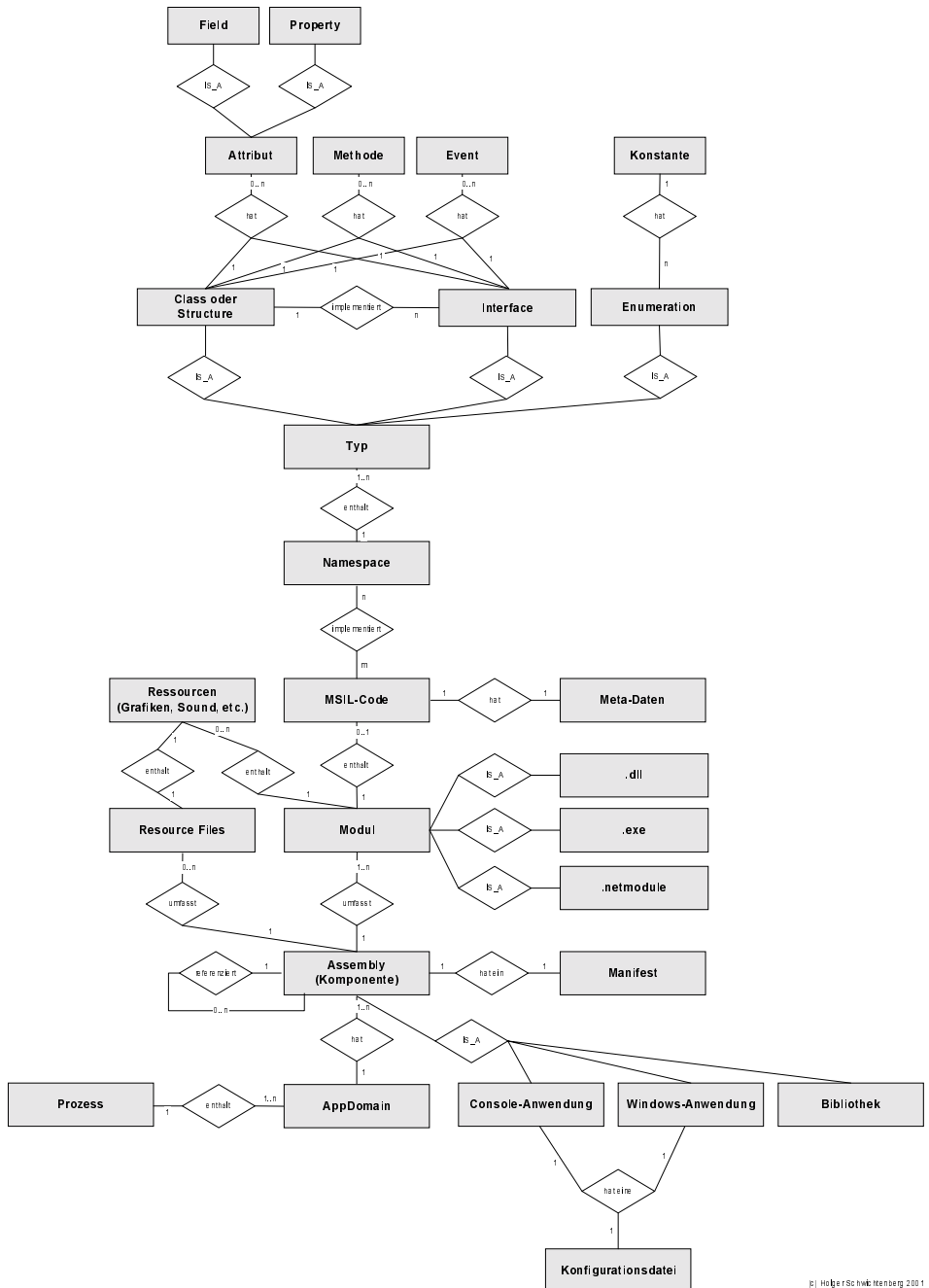
COM+ und .NET

Im Rahmen von .NET heißen die COM++-Dienste .NET Enterprise Services. Auch wenn das .NET Framework nicht mehr auf COM basiert, werden in Ermangelung äquivalenter Implementierungen in Managed Code noch die konventionellen COM++-Dienste für Transaktionen o. Ä. genutzt.

.NET Framework

Komponenten- basiertes Programmier- modell

Das .NET Framework ist ein plattformunabhängiges, komponenten-basiertes Programmiermodell. Microsoft bezeichnet das .NET Framework auch als internetoptimiertes Software-Architekturmodell.



Das .NET Framework ist ein neues technisches Konzept zur Anwendungsentwicklung unter Windows. Dabei ist – im Gegensatz zu anderen, von Microsoft zum Hype erklärten Neuerungen – das .NET Framework eine wirklich große Veränderung, wie Microsoft sie alle fünf bis sechs Jahre vollzieht. Vergleichbare Veränderungen waren der Schritt von DOS zu Windows und von 16-Bit-Windows zu 32-Bit-Windows.

Abbildung A.3 zeigt ein Entity-Relationship-Modell der wichtigsten Konzepte im .NET Framework-Komponentenmodell.

.NET Framework Class Library

FCL siehe Framework Class Library

.NET Runtime

siehe Common Language Runtime

.NET Framework Software Development Kit

SDK Das kostenlose .NET SDK enthält Dokumentationen und verschiedene Werkzeuge zur Arbeit mit dem .NET Framework.

Active Server Pages.NET (ASP.NET)

ASP.NET Webserver-Middleware zur Webserver-Programmierung; Nachfolger von ASP unter .NET

ActiveX Data Objects.NET (ADO.NET)

ADO.NET ADO.NET ist eine Programmierschnittstelle für den Zugriff auf Daten aller Art, insbesondere relationale Datenbanken. ADO.NET ist der Nachfolger von ADO und realisiert durch den Namespace `System.Data` in der FCL.

Application Context

Kontext Eine Application Domain kann in so genannte Kontexte aufgespalten werden. Ein Kontext ist eine Menge von Objekten innerhalb einer Application Domain, für die bestimmte Einstellungen gelten. Solche Einstellungen sind beispielsweise die Transaktionssteuerung. Kontexte werden mit Hilfe von Meta-Attributen definiert.

Application Domain (AppDomain)

Eine Application Domain ist die Unterteilung eines Prozesses in mehrere Fächer. Anwendungen, die in verschiedenen Application Domains laufen, sind voneinander genauso isoliert, als liefen sie in verschiedenen Prozessen.

AppDomain

Der Vorteil ist, dass der Aufwand zur Erzeugung einer Application Domain und für einen Programmcode-Aufruf zwischen zwei Application Domains geringer ist als für die Verwendung von Prozessen. Eine Application Domain kann mehrere Assemblies beinhalten.

Eine Application Domain wird auch als Pseudo-Prozess bezeichnet.



ASP.NET Server Control

siehe Server Control

Assembly

Eine Softwarekomponente heißt im .NET Framework Assembly (im Deutschen von einigen Autoren *Assemblierung* genannt). Jede .NET-Anwendung ist eine Assembly und jede Assembly ist eine Komponente im Sinne objektorientierter Softwarekomponenten-Architekturen (siehe auch Begriff *Komponente*). Eine .NET-Anwendung unterscheidet sich von einer normalen Komponente nur dadurch, dass sie einen eindeutigen Einsprungspunkt (*Entry Point*) besitzt, der von dem Runtime Host, der die Assembly geladen hat, aufgerufen werden kann. Wiederverwendung, Sicherheit, Versionierung und Deployment finden im .NET Framework auf der Ebene der Assemblies statt.

Komponente

Eine Assembly ist eine Funktionseinheit und besteht aus einem Modul (Single-File-Assembly) oder aus mindestens einem Modul und beliebigen weiteren Dateien (Multi-File-Assembly).

Assembly-Typen

Jede Assembly hat genau ein Manifest. Bei einer Multi-File-Assembly muss das Manifest gemeinsam mit dem Einsprungspunkt in einem der Module liegen.

Manifest

Eine Assembly wird im Standard anhand ihres Namens identifiziert. Eine Assembly kann darüber hinaus einen *Strong Name* besitzen und/oder mit dem Authenticode-Verfahren signiert werden.

Optionen



Auch unter COM ab Windows XP (NT 5.1) gibt es den Begriff Assembly. Dort ist eine Assembly ein Verbund von COM-Komponenten.

Assembly-Referenz (engl. Assembly Reference)

AssemblyRef

Eine Assembly kann Verweise auf andere Assemblies enthalten, wenn sie Typen aus diesen anderen Assemblies benötigt. Dazu enthält das Manifest einer Assembly eine Tabelle mit dem Namen *AssemblyRef*.

Attribut

Attribute sind Daten-Mitglieder einer Klasse. Es gibt in .NET zwei Arten von Attributen: Felder (engl. Fields) und Eigenschaften (engl. Properties).

Field *Fields* sind Attribute, bei denen der Nutzer des Objekts direkt einen bestimmten Speicherplatz innerhalb des Objekts beschreibt. Er kann alle dem Datentyp des Feldes entsprechenden Werte in das Feld schreiben, ohne dass eine weitere Prüfung erfolgt.

Property Im Gegensatz dazu wird bei einer *Property* bei jedem Schreib- und Lesezugriff Programmcode ausgeführt, bei dem z.B. Prüfungen durchgeführt werden können. So kann der Gültigkeitsbereich auf beliebige Werte beschränkt werden. Zu einer Property gehört eine *Get()*-Methode und/oder eine *Set()*-Methode. Wie und ob überhaupt die übergebenen Werte verarbeitet werden, ist in diesen Methoden zu implementieren.

Aus der Sicht des Nutzers der Klasse gibt es keinen Unterschied zwischen Fields und Properties.



Microsoft verwendet den Begriff Attribut im .NET Framework auch noch für einen anderen Fall, siehe Meta-Attribut.

MINFU

Aufzählungstyp

Ein Aufzählungstyp (Enumeration) ist eine Liste von Konstanten. Jede einzelne symbolische Konstante der Liste repräsentiert einen Ganzzahlwert. Zeichenketten oder andere Datentypen sind nicht erlaubt.

Base Class Library (BCL)

BCL siehe Framework Class Library

Boxing

Umwandlung eines Wert-Typs in einen Referenz-Typ (Gegenteil: Unboxing)

Classic Code

siehe Managed Code

Code Access Security (CAS)

Name für das Sicherheitssystem in .NET

CAS

Bei der CAS spielen im Gegensatz zu COM neben den Benutzerrechten, unter denen eine Komponente ausgeführt wird, auch Eigenschaften der Komponente selbst (z.B. Hersteller) und die Herkunftsquelle bzw. der Speicherort der Komponente eine Rolle. Dies basiert auf dem bereits aus den Sicherheitseinstellungen des Internet Explorers bekannten Zonenkonzept (MY COMPUTER, LOCAL INTRANET, TRUSTED SITES, INTERNET etc). Die Code Access Security (CAS) ist dabei eine starke Erweiterung der Internet Explorer-Sicherheit, da in .NET eigene Programmcodegruppen auf Basis eines HTTP-, FTP- oder File-URLs gebildet werden können.

COM Callable Wrapper

.NET-Komponenten werden von COM-Clients über einen COM Callable Wrapper (CCW) angesprochen. Normalerweise stellen die .NET-Entwicklungswerkzeuge einen CCW automatisch zur Verfügung. Ein CCW kann aber auch selbst entwickelt werden.

CCW

COM+

COM+ ist die Weiterentwicklung von COM, die in die Windows-Produktlinie ab Windows 2000 integriert ist. COM+ ist im Vergleich zu COM nichts völlig Neues (auch wenn das Microsoft-Marketing das gerne so verkauft), sondern nur die Ergänzung von COM um zusätzliche Dienste. Die COM+-Dienste sind wiederum eine Weiterentwicklung der Dienste, die unter Windows NT4 noch als *Microsoft Transaction Server (MTS)* bekannt waren. Zentraler Dienst von MTS und COM+ ist die komponentenbasierte Transaktionssteuerung.

Auch das .NET Framework verwendet noch die COM+-Dienste, da es in .NET noch keine eigenen äquivalenten Dienste gibt. .NET-Komponenten, die COM+-Dienste nutzen, heißen *Services Components*. Im Rahmen von .NET werden die COM+-Dienste *.NET Enterprise Services* genannt.

Common Intermediate Language (CIL)

- CIL** CIL ist der Name der Zwischensprache, die die Common Language Infrastructure (CLI) verwendet. Die MSIL ist eine Implementierung der CIL (siehe Stichwort *Intermediation*).



Siehe auch [CD:/Weitere Informationen/ECMA-Dokumente/TG3_CLI/PartitionIIICILOct01.doc].

Common Language Infrastructure (CLI)

- CLI** Die Common Language Infrastructure (CLI) ist der Teil des .NET Frameworks, den Microsoft bei der ECMA standardisiert hat (ECMA-Standard 335, Arbeitsgruppe TC39/TG3).

Weitere Informationen dazu finden Sie unter [CLI01a] und [CD:/Weitere Informationen/ECMA-Dokumente/TG3_CLI].

- Source Code** Es gibt eine Shared Source-Version der CLI und der Sprache C# für Windows XP und FreeBSD 4.5 [CD:/install/CLI/Source/].

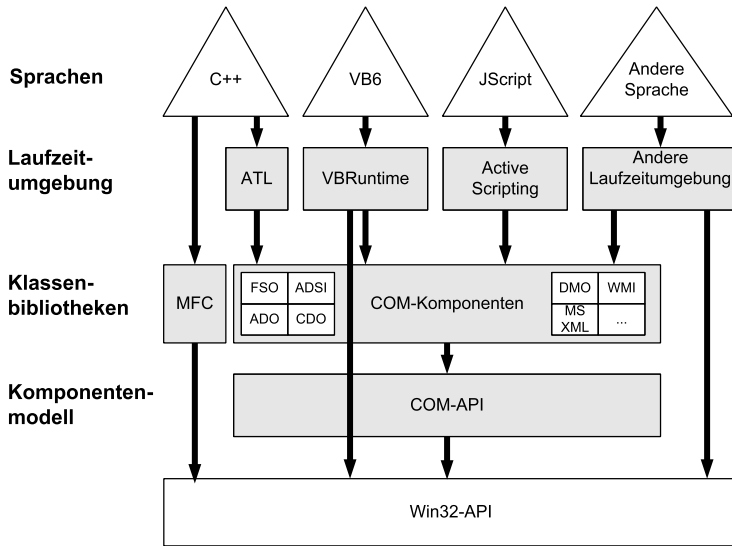


Da die CLI auch weite Teile der Framework Class Library umfasst, finden Sie in der Shared Source-Version der CLI auch den Quellcode (in C#) zu zahlreichen der in diesem Buch beschriebenen Klassen. Ein Einblick in diesen Quellcode kann sehr hilfreich für das Verständnis der FCL-Klassen sein!

Common Language Runtime (CLR)

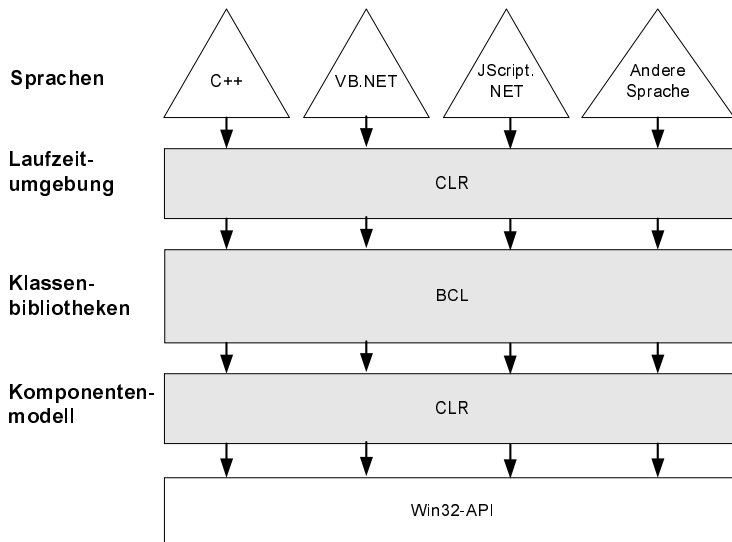
- CLR** Die Common Language Runtime (CLR) ist die Laufzeitumgebung des .NET Frameworks. Sie muss installiert sein, damit ein .NET-Programm (egal ob in Managed Code oder Native Code) ausgeführt werden kann. Die CLR ist die einheitliche Laufzeitumgebung aller .NET-Programmiersprachen. Sie ersetzt die in der COM-Welt vorhandenen unterschiedlichen Laufzeitumgebungen.

Die CLR stellt zahlreiche Basisdienste bereit. Dazu gehören der Just-in-Time-Compiler, die Code Access Security (CAS), der Garbage Collector und die Framework Class Library (FCL).



© Holger@Schwichtenberg.de 2001

Abbildung A.4: Laufzeitumgebung und Klassenbibliotheken vor .NET



© Holger@Schwichtenberg.de 2001

Abbildung A.5: Die Rolle der CLR in .NET

Die CLR ist implementiert in Form der Common Language Runtime Execution Engine in der Datei *mscorlib.dll*. Diese DLL wird nach `%Windows%/System32` installiert. Was überraschen mag: Diese DLL ist eine COM-Komponente. Man könnte auch sagen, diese Komponente ist die letzte COM-Komponente, die man schreiben musste. Alles danach könnte man als .NET-Komponente realisieren.



Die CLR hatte vor der offiziellen Vorstellung der .NET-Initiative im Juli 2000 die Namen NGWS Runtime und Universal Runtime.

NGWS

Common Language Specification (CLS)

CLS Die Common Language Specification (CLS) ist ein Regelwerk für Autoren von Compilern, das festlegt, wie die Umsetzung in die Microsoft Intermediation Language (MSIL) erfolgen muss. Die CLS legt auch fest, wie ein Application Programming Interface (API) aufgebaut sein muss, um im Rahmen des .NET Frameworks genutzt zu werden.

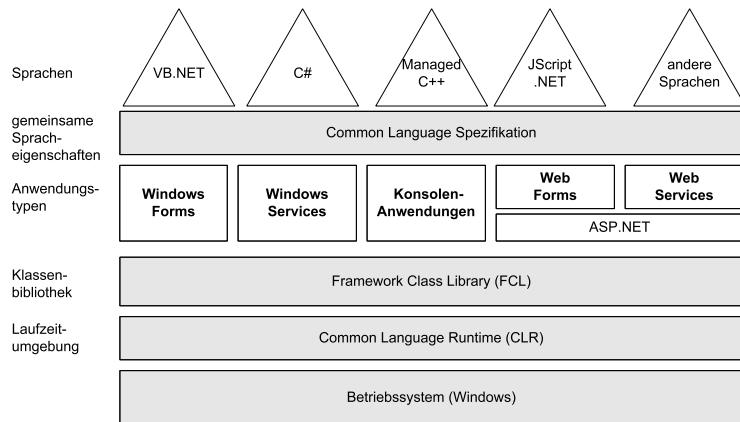


Abbildung A.6: Die Rolle der CLS

CTS .NET-Komponenten dürfen nach außen nur CLS-konforme Typen aus dem *Common Type System (CTS)* verwenden. Intern dürfen auch andere Typen verwendet werden.

Es gibt insgesamt 41 CLS-Regeln, von denen die meisten sich auf das CTS beziehen.

Weitere Informationen zur CLS finden Sie im CLI-Standard [CD:/Weitere Informationen/ECMA-Dokumente/TG2_CLI/PartitionI ArchOct01.doc]



Common Type System (CTS)

Zur CLS gehört auch ein gemeinsames Typkonzept, das *Common Type System (CTS)* genannt wird. Verschiedene Sprachen werden also nicht mehr wie bisher mit einem Datentyp verschiedene Konzepte verbinden (Ist ein Integer-Wert 16 oder 32 Bit lang? Ist ein String ein Array oder ein Zeiger auf einen mit 0 terminierten Speicherbereich?). Durch das CTS ist ein bestimmter Datentyp in allen CLS-Sprachen gleich. Dadurch werden die Sprachen interoperabel. Durch das CTS wird es auch möglich, dass in unterschiedlichen Sprachen implementierte Klassen voneinander erben (z.B. erbt eine Perl-Klasse von einer Visual Basic .NET-Klasse, die selbst wieder Unterklasse einer C#-Klasse ist).

CTS

Compact Framework (CF)

Das .NET Compact Framework ist eine spezielle, reduzierte und angepasste Version des .NET Frameworks für mobile Geräte wie Pocket PCs, Handheld PC und Hanfies, die ein Betriebssystem auf Basis von Windows CE verwenden.

CF

Das CF umfasst nur rund 10% der Größe des .NET Frameworks, deckt aber rund 25% der Funktionalität des großen Bruders ab. Nicht enthalten im CF ist zum Beispiel ASP.NET, die COM-Interoperabilität, ActiveX-Control-Unterstützung, verschiedene Windows Forms-Steuerelemente (z.B. FontDialog, ColorDialog, Splitter) die Nutzung beliebiger OLEDB-Provider, Dateisystemüberwachung, Remoting (mit Ausnahme des Einsatzes als Webservice-Client), GDI+, Drucken, Drag&Drop, Binäre Serialisierung, XPath, XML-Validation, die Laufzeitcodegenerierung und die Erzeugung eines Native Image (Install Time Compilation). Als Sprachen werden in der ersten Fassung des CF nur Visual Basic .NET und C# unterstützt. Die Klassenbibliothek des CF enthält aber auch zusätzliche Klassen, z. B. für IrDA und SQL Server CE. Microsoft behauptet aber, mit diesen 25% rund 75% der Funktionalitäten abzudecken, die beim mobile Computing praktisch gebraucht werden. Die restlichen Funktionen des OS kann eine Anwendung über den in .NET integrierten Plattform Invoke-Mechanismus (kurz P/Invoke) aufrufen.

Component Object Model (COM)

COM Das Component Object Model (COM) ist Microsofts Ansatz für objektorientierte Softwarekomponenten. COM ist das allgemeine Modell für den Aufbau und die Benutzung von Komponenten und der Vorgänger des .NET Frameworks. Das .NET Framework sollte ursprünglich COM 3.0 heißen. Jedoch hat Microsoft sich später für einen anderen Namen entschieden – was auch gerechtfertigt ist, weil das .NET Framework nicht mehr auf COM basiert.

Das .NET Framework verbessert zehn wesentliche Unzulänglichkeiten von COM:

Komplexität

1. Komplexität

Das neue Komponentenmodell ist einfacher und universeller. Es gibt keine verschiedenen Zugänge zu einem Objekt mehr wie in COM (`IUnknown` und `IDispatch`), sondern nur noch ein einheitliches Verfahren, das durch die .NET-Laufzeitumgebung für den Entwickler völlig transparent abläuft. Anders als in COM kann jeder .NET-Client jede .NET-Komponente nutzen. Nicht mehr Schnittstellen, sondern Klassen stehen im Mittelpunkt. .NET unterstützt aber ebenso wie COM Klassen mit Mehrfachschnittstellen. .NET-Klassen haben hierarchische Namen. Schnittstellen werden explizit definiert, anstatt abstrakte Klassen als Hilfskonstrukte zu nutzen.

Zudem sind alle .NET-Anwendungen und -Bibliotheken zugleich .NET-Komponenten. Es wird nicht mehr wie in COM zwischen einfachen DLLs/EXEs und Komponenten-DLLs/COM-EXEs unterschieden.

Meta-Daten

2. Selbstbeschreibung

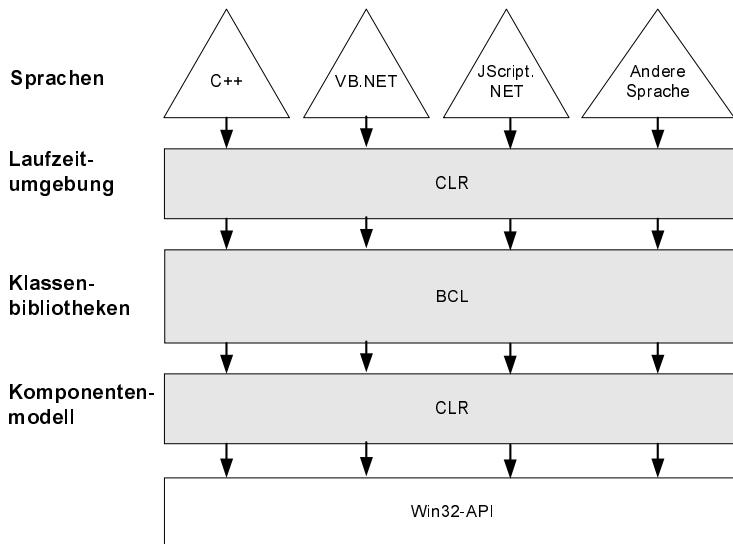
Im .NET Framework ist es Pflicht, dass alle Komponenten eine Selbstbeschreibung enthalten. Diese Meta-Daten liefern Informationen über die enthaltenen Klassen mit ihren Schnittstellen und über deren Attribute, Methoden und Ereignisse. Die Sprach-Compiler haben dafür zu sorgen, dass diese Meta-Daten erzeugt werden und konsistent zum wirklichen Inhalt der Komponenten sind.

Common Language Runtime

3. Einheitliche Laufzeitumgebung

COM stellt zwar ein Verfahren bereit, wie in unterschiedlichen Sprachen implementierte COM-Komponenten untereinander kommunizieren können, die Erstellung und Nutzung von COM-Komponenten ist jedoch in verschiedenen Sprachen

unterschiedlich gelöst gewesen. Um die komplexen COM-Mechanismen für den Programmierer einfacher nutzbar zu machen, enthalten die COM-fähigen Sprachen eine Laufzeitumgebung. Dies waren bisher für Visual Basic die VB-Runtime und für Visual C++ die Active Template Library (ATL). .NET besitzt eine einheitliche Laufzeitumgebung für alle Sprachen, die Common Language Runtime (CLR). Dies ist die Umsetzung eines Features, das Mitte der Neunzigerjahre bereits für das mit Windows 2000 ausgelieferte COM+ angekündigt war. COM+ enthielt jedoch diese Laufzeitumgebung noch nicht.



© Holger@Schwichtenberg.de 2001

Abbildung A.7: Einheitliche Laufzeitumgebung und einheitliche Klassenbibliothek in .NET

4. Verbesserte Versionierung

Eine Komponente kann in beliebig vielen Versionen auf einem einzigen System parallel installiert sein. Jede Anwendung kann so konfiguriert werden, dass sie eine bestimmte Version einer Komponente nutzt.

Versionierung

5. Einfachere Installation

Komponenten müssen nicht mehr in der Windows-Registry registriert werden. Um eine Komponente für eine einzelne Anwendung verfügbar zu machen, reicht es, sie auf die Fest-

**Keine
Registrierung**

platte des Systems zu kopieren (Schlagwort »XCOPY-Deployment«). Lediglich globale Komponenten erfordern eine spezielle Behandlung.

Remoting 6. Bessere Fernausführung

Die Verteilungsplattform von COM, das Distributed COM (DCOM), war eine proprietäre Entwicklung von Microsoft, die sich im Internet nicht durchsetzen konnte, da DCOM zahlreiche TCP-Ports verwendet, die durch Firewalls geblockt werden. Im .NET Framework erfolgt die Nutzung entfernter Komponenten auf Basis des Simple Object Access Protocols (SOAP).

Vererbung 7. Vererbung

Das .NET-Komponentenmodell unterstützt im Gegensatz zu COM die Vererbung, auch zwischen verschiedenen Sprachen.

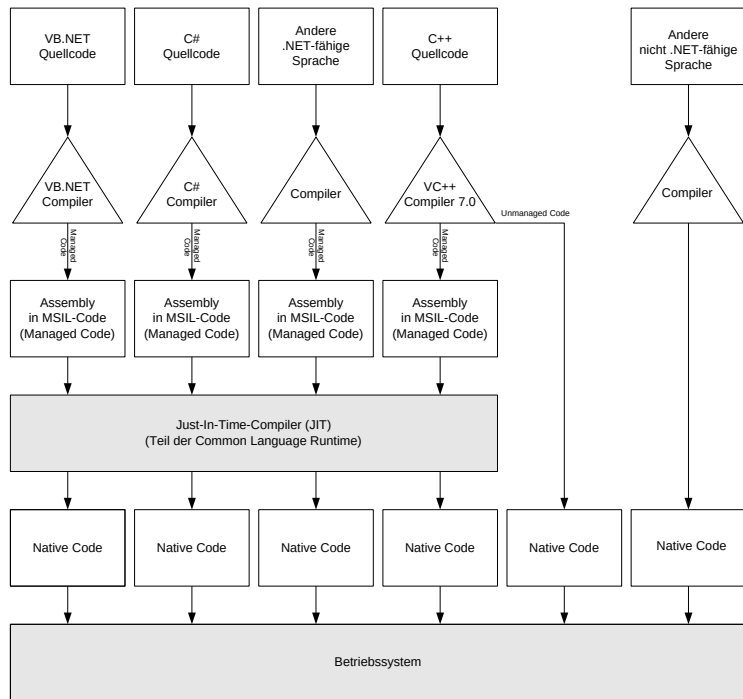


Abbildung A.8: NET-Intermediation

8. Plattformunabhängigkeit

Plattformunabhängigkeit

.NET-Komponenten sind durch ein Intermediation-Konzept plattformunabhängig. Compiler und Interpreter erzeugen in Zukunft keinen Maschinencode mehr, sondern einen prozessorunabhängigen Zwischencode. Dieser verwendet eine neue Sprache, die so genannte *Microsoft Intermediation Language (MSIL)*. Ein *Just-in-Time-Compiler* übersetzt diese MSIL zur Laufzeit in Maschinencode. Dabei optimiert der Just-in-Time-Compiler den Maschinencode für die jeweilige Plattform (insbesondere den Prozessortyp).

9. Bessere Garbage Collection

Speicher aufräumen

Die fehleranfällige Referenzzählung in COM mit `IUnknown::AddRef()` und `IUnknown::Release()` wird in .NET durch einen Garbage Collector ersetzt, der eigenständig in regelmäßigen Zyklen nicht mehr benutzte Objekte aufspürt und den Speicher freigibt. Der Garbage Collector erkennt auch zirkuläre Referenzen.

10. Code Access Security (CAS)

Sicherheit

Im .NET Framework kann sehr genau gesteuert werden, auf welche Systemressourcen eine .NET-Anwendung zugreifen kann.

CSharp

C# (gesprochen »C Sharp«) ist eine Weiterentwicklung von C++. Das # könnte man auch in ein vierfaches Pluszeichen aufsplitten (also C++++). Konzeptionell ist C# eine Mischung aus Visual C++ 7.0 und Visual Basic .NET. Gewisse Ähnlichkeiten gibt es zu Java. C# ist Ergebnis eines Projekts bei Microsoft, das gestartet wurde, nachdem die Firma Sun Microsoft die Veränderung der von Sun entwickelten Programmiersprache Java verboten hatte. Ursprünglich sollte die Sprache »Cool« heißen. Vater von C# ist Anders Heljsberg, der Erfinder von Turbo Pascal und Borland Delphi.

C#

C# ist bei der ECMA standardisiert (ECMA Standard 334, Arbeitsgruppe TC39/TG2). Weitere Informationen dazu finden Sie unter [CLI01a] und [CD:/Weitere Informationen/ECMA-Dokumente/TG2_C#].

ECMA 3334

Es gibt eine Shared Source-Version der Sprache C# für Windows XP und FreeBSD 4.5 [CD:/install/CLI/Source/ Source/sscli20020326/clr/src/csharp], die den kompletten Source-Code für den C#-Compiler enthält. Der C#-Compiler ist in C++ implementiert.

Shared Source-Version

**Vergleich zu
VB.NET**

Die Unterschiede zwischen Visual Basic .NET und C# sind eher syntaktischer Natur; hinsichtlich der Möglichkeiten gibt es nur marginale Vorteile für C#. Dies gilt auch für die Performance. Da beide Sprachen MSIL-Code erzeugen, sind große Unterschiede nicht vorhanden.



An manchen Stellen (z.B. im Startbildschirm von Visual Studio .NET) wird C# auch mit C#.NET bezeichnet.

Custom Control

siehe User Control

Distributed Component Object Model (DCOM)

DCOM DCOM erweitert COM um die Netzwerkfähigkeit, also die rechnerübergreifende Benutzung der COM-Komponenten in einem verteilten System.

Die Verteilungsplattform von COM, das Distributed COM (DCOM), ist eine proprietäre Entwicklung von Microsoft, die sich im Internet nicht durchsetzen konnte, da DCOM zahlreiche TCP-Ports verwendet, die durch Firewalls geblockt werden. Im .NET Framework erfolgt die Nutzung entfernter Komponenten auf Basis des Simple Object Access Protocols (SOAP).

DLL-Hölle

Die DLL-Hölle entsteht im klassischen COM durch zwei Faktoren:

1. Es kann nur eine Version einer bestimmten Komponente in der Registry registriert sein.

Dies bezieht sich insbesondere auf Unterversionen (Patches). Natürlich kann eine COM-Komponente grundsätzlich in zwei verschiedenen Versionen auf einem System existieren, wenn die beiden Versionen verschiedene GUIDs verwenden und andere Dateinamen haben. Dann handelt es sich nämlich aus der Sicht von COM um zwei verschiedene Komponenten.

Sobald aber von einer Komponente eine überarbeitete Version erscheint, die aus Kompatibilitätsgründen keine neuen GUIDs bekommen hat, kann nur eine der beiden Versionen auf dem System registriert sein.

2. Eine Komponente, die in Benutzung ist, kann nicht aktualisiert werden. Beim Austausch der Komponentendatei (DLL, EXE, etc.) kommt es zu einer Fehlermeldung. Komponenten sind teilweise auch noch lange nach ihrem tatsächlichen Nutzungs-ende vom System gesperrt.

DOTNET

siehe .NET

Eigenschaft

siehe Attribut

Enumeration

siehe Aufzählungstyp

Enterprise Architect

Variante von Visual Studio .NET, die auf Basis der Integration mit Microsoft Visio verschiedene Werkzeuge zur Analyse und Modellierung bereitstellt. Unterstützt werden die Unified Modelling Language (UML), das Object Role Modelling (ORM) und das Entity Relationship-Modell (ERM).

Enterprise Templates

Vorlagen zum Customizing von Visual Studio .NET, verwenden unter anderem die Template Definition Language (TDL). Enterprise Templates kann man nur mit den Enterprise-Versionen von Visual Studio .NET nutzen. Zum Erstellen von Enterprise Templates benötigt man die Enterprise Architect-Version.

European Computer Manufacturers Association (ECMA)

Standardisierungsgremium, bei dem die Sprache C# und Teile des .NET Frameworks unter dem Namen Common Language Infrastructure (CLI) standardisiert wurden. Ein anderer bekannter ECMA-Standard ist ECMAScript (alias JavaScript/JScript).

ECMA

An der Standardisierung von C# (ECMA-334) und der CLI (ECMA-335) haben folgende Unternehmen teilgenommen: Fujitsu Software, Hewlett-Packard, Intel Corporation, International Busi-

ness Machines, ISE, Microsoft Corporation, Monash University, Netscape, OpenWave, Plum Hall und Sun Microsystems.

Field

siehe Attribut

Framework Class Library (FCL)

FCL Zur Common Language Runtime (CLR) gehört eine sehr umfangreiche Klassenbibliothek, die so genannte .NET Framework Class Library. Die Framework Class Library (FCL) ist eine einheitliche Klassenbibliothek für alle .NET-fähigen Programmiersprachen (CLS-konformen Sprachen).

BCL/FCL-MINFU Zum Teil spricht Microsoft auch von der Base Class Library (BCL). In der Beta1-Version der Framework SDK-Dokumentation hieß das entsprechende Kapitel noch »Base Class Library«. In der Beta2 wurde es »Framework Class Library« genannt. In der Endfassung steht einfach »Class Library«. Letzter offizieller Stand vor Drucklegung dieses Buchs war, dass die BCL ein Teil der FCL ist, nämlich die Klassen der *mscorlib.dll* und *system.dll*. Diese Assemblies enthalten die Basisdatentypen (*System*), die IO-Funktionen (*System.IO*), die Netzwerkfunktionen (*System.Net*), das Reflection-API (*System.Reflection*), die COM-Interoperabilität und das Remoting (*System.Runtime*), Sicherheitskonfiguration (*System.Security*), Threading (*System.Threading*) und den Zugriff auf viele Windows-Bausteine (*System.Diagnostics*).

Garbage Collector (GC)

GC Im Gegensatz zu COM verfügt das .NET Framework über eine automatische Speicherverwaltung, die in der Common Language Runtime (CLR) implementiert ist.

Die CLR enthält einen Garbage Collector (GC), der im Hintergrund arbeitet und den Speicher aufräumt. Der Speicher wird allerdings nicht sofort nach dem Ende der Verwendung eines Objekts freigegeben, sondern zu einem nicht festgelegten Zeitpunkt bei Bedarf (Lazy Resource Recovery). Der Garbage Collector kann aber auch manuell gestartet werden.

Global Assembly Cache (GAC)

Der Global Assembly Cache (GAC) ist ein systemweites Verzeichnis für Shared Assemblies. Das Verzeichnis liegt im Dateisystem unter `%Windows%\Assembly`. Der GAC ist der Speicher für globale Komponenten. Das Besondere am GAC ist, dass er im Gegensatz zu einem normalen Dateiverzeichnis beliebig viele Versionen einer einzigen Datei (auch bei gleichem Namen) speichern kann. Dies wird möglich, weil der GAC in der Implementierung kein einzelnes Verzeichnis, sondern eine Verzeichnishierarchie ist.

GAC

Die Verwendung des GACs sollte wohl überlegt sein, da dieser Speicher genau wie das System32-Verzeichnis dazu neigen wird, mit längst nicht mehr benötigten Assemblies gefüllt zu sein.



Hailstorm

siehe .NET My Services

Intermediation

Das .NET-Intermediation-Konzept ist ein Programmierkonzept, das Java sehr ähnlich ist: Compiler und Interpreter erzeugen keinen Maschinencode mehr, sondern einen prozessorunabhängigen Zwischencode. Dieser verwendet eine neue Sprache, die so genannte Microsoft Intermediation Language (MSIL).

Ein *Just-in-Time-Compiler* übersetzt MSIL zur Laufzeit in Maschinencode. Dabei optimiert der Just-in-Time-Compiler den Maschinencode für die jeweilige Plattform (insbesondere den Prozessortyp). Das Konzept erinnert an den Java-Bytecode. Die MSIL hat 240 Operation Codes (OpCodes), ähnlich dem Java-Bytecode. Der Just-in-Time-Compiler ist Teil der .NET-Laufzeitumgebung, die Common Language Runtime (CLR) heißt.

Im Rahmen des Standards Common Language Infrastructure (CLI) heißt die Zwischensprache Common Intermediate Language (CIL) statt MSIL.

CIL

Intermediation Language (IL)

siehe Microsoft Intermediation Language

Java User Migration Path to Microsoft .NET

- JUMP** Strategie von Microsoft zur Integration von Java in das .NET Framework, siehe auch *JSharp*. JUMP basiert auf einem Vertrag zwischen Microsoft und Sun, der im Zuge des Konflikts über J++, die Microsoft-Implementierung von Java, entstanden ist.

JavaScript.NET

.NET-fähige Version von ECMAScript (JavaScript/JScript); ECMA-Standard 262, 290 und 327.

JSharp

- J#** Die Programmiersprache Java für das .NET Framework. Diese Implementierung erzeugt Code für die CLR in .NET, nicht für die Java Virtual Machine. J# ist Nachfolger von J++.

Just-in-Time-Compiler

Name des Übersetzers zwischen Managed Code (MSIL) und Native Code. Der Just-in-Time-Compiler ist Teil der Common Language Runtime (CLR).

Klasse

Eine .NET-Klasse umfasst Attribute, Methoden, Ereignisse, keinen, einen oder mehrere Konstruktoren, höchstens einen Destruktor, Konstanten und Aufzählungstypen. Eine Klasse ist in .NET ein Referenz-Typ.

Konfigurationsdateien in .NET

Ein wesentlicher Unterschied zwischen COM-Komponenten und .NET-Komponenten ist, dass letztere nicht mehr in der Windows-Registry registriert werden müssen. Alle notwendigen Konfigurationsinformationen für .NET-Komponenten werden in den Metadaten und Konfigurationsdateien im Dateisystem gespeichert. Die Konfigurationsdateien sind Textdateien im XML-Format.

Komponente

»Components repräsentieren speziell im Hinblick auf Wiederverwendung entworfene und implementierte Softwarebausteine. Ein Baustein stellt eine Menge von öffentlichen Diensten (engl. public services) zur Nutzung bereit.« ([EIC99], S. 363)

Gemäß dieser den üblichen Gepflogenheiten entsprechenden Definition ist jede .NET-Assembly eine Komponente, sofern sie öffentliche Klassen bereitstellt. Microsoft betreibt in .NET mit diesem Begriff wieder MINFU und definiert in einigen Fällen, das eine .NET-Komponente eine .NET-Klasse ist, die die Schnittstelle `System.ComponentModel.IComponent` implementiert. Diese Schnittstelle kann eine Klasse selbst implementieren oder von einer Oberklasse erben, z.B. von `System.ComponentModel.Component` oder `System.ComponentModel.MarshalByValueComponent`. Eine Klasse, die `IComponent` implementiert, hat den Vorteil, dass sie in einen visuellen Container aufgenommen werden kann. Ein Container kann ein Steuerelement sein (z.B. ein Webform oder ein Windows Form) oder auch nur zur Entwicklungszeit sichtbar sein (z.B. die Toolbox und die Designer in Visual Studio .NET).

**Komponenten-
MINFU**

Diese Verwendung des Begriffs Komponente ist auch deshalb ungünstig, weil in der Lehre der komponentenorientierten Softwareentwicklung und anderen Komponentenarchitekturen wie COM und JavaBeans üblicherweise eine Komponente aus mehreren Klassen besteht. Zudem verwendet Microsoft selbst an anderer Stelle in der .NET Framework SDK-Dokumentation den Begriff Komponente auch synonym zu Assembly. Auch dieses Buch hält sich an die allgemeine Definition, zumal der Namespace `System.ComponentModel` in diesem Buch nicht näher besprochen wird und es daher nicht zu Namenskonflikten kommt.



Managed Code

Programmcode, der im .NET Framework unter der .NET-Laufzeitumgebung abläuft, wird als Managed Code bezeichnet. Im Gegensatz dazu wird herkömmlicher Code als Unmanaged Code oder Classic Code bezeichnet.

Verwalteter Code

Das Kompilat von C#, Visual Basic .NET und JScript.NET ist immer Managed Code. Visual C++ 7.0 kann durch eine Compiler-Option Managed Code erzeugen. Visual FoxPro kann keinen Managed Code erzeugen. Es gibt zahlreiche weitere Sprachen, die Managed Code erzeugen können.

Managed Execution

Dieser Begriff bezeichnet die Ausführung von Managed Code durch die Common Language Runtime.

Managed Provider

siehe .NET Data Provider

Manifest

Jede Assembly enthält genau ein Manifest. Das Manifest einer Assembly übernimmt die Rolle einer Typbibliothek in COM, beinhaltet aber weitaus mehr Informationen:

- ▶ Name, Version und Ländercode einer Assembly
- ▶ die zur Assembly gehörenden .NET-Module (PE-Dateien)
- ▶ die zur Assembly gehörenden Ressourcen-Dateien (Dateien, die nicht PE-Dateien sind, z.B. *.bmp*, *.jpg*)
- ▶ Abhängigkeiten von anderen Assemblies (Referenzen)
- ▶ die von der Assembly exportierten Typen
- ▶ Rechte, die zur Ausführung der Assembly notwendig sind
- ▶ optional einen öffentlichen Schlüssel

Das Manifest kann Teil einer PE-Datei der Assembly sein oder in einem getrennten PE-File abgelegt werden. Bei Multi-File-Assemblies ist das Manifest das Bindeglied zwischen den einzelnen Dateien, da diese keine Kenntnis darüber haben, ob und zu welcher Assembly sie gehören.

Meta-Attribut

Deklaratives Programmieren

Assemblies, Klassen, Methoden, Ereignisse und Attribute, also alle Typen und alle Typmitglieder, können mit zusätzlichen Auszeichnungen/Merkmalen versehen werden, die zur Laufzeit ausgewertet werden. So kann zum Beispiel deklariert werden, dass ein Objekt serialisierbar sein soll oder ein bestimmtes Attribut in einer Klasse bei der Serialisierung unberücksichtigt bleiben soll. Attribute sind Metadaten, da sie den Programmcode näher beschreiben.

Leider hat Microsoft eine von der objektorientierten Lehre abweichende Begriffswelt: Microsoft nennt eine derartige Auszeichnung *Attribut* (engl. *Attribute*), was einen Namenskonflikt zu dem Begriff Attribut in der objektorientierten Lehre (vgl. für den deutschen Sprachraum [OES97], S. 157 und für den englischen Sprachraum [OXF97], S. 243) darstellt. Letztere Attribute nennt Microsoft *Felder* (engl. *Fields*) und *Eigenschaften* (engl. *Properties*). Ein klarer Fall von MINFU (vgl. Anhang D).

MINFU

Um diesen Namenskonflikt zu lösen, wird in diesem Buch der Begriff Meta-Attribute für die Auszeichnungen verwendet. Attribut steht für die normalen Daten-Mitglieder einer Klasse.

Konvention

Meta-Attribute werden in Visual Basic .NET in spitze Klammern `<AttributName>` und in C# in eckige Klammern `[Attributname]` gesetzt.

Meta-Daten

Informationen über Typen. In COM sind Meta-Daten in optionalen Typlibibliotheken abgelegt. Im .NET Framework sind Meta-Daten Pflichtbestandteile einer jeden Assembly.

Microsoft Intermediation Language (MSIL)

Die in .NET verwendete Zwischensprache (siehe *Intermediation*).

MSIL

Mobile Internet Toolkit (MMIT oder MIT)

Das Mobile Internet Toolkit ist eine Erweiterung für ASP.NET und Visual Studio .NET zur Entwicklung von Anwendungen für mobile Endgeräte. Das Mobile Internet Toolkit enthält u.a. Web-controls (`System.Web.UI.MobileControls`), die Wireless Markup Language (WML) erzeugen, und einen passenden Mobile Internet Designer für Visual Studio .NET.

MMIT

Das MIT ist kostenlos erhältlich und auf der CD zu diesem Buch enthalten [CD:/Install/DOTNET Add ons/Mobile Internet Toolkit].

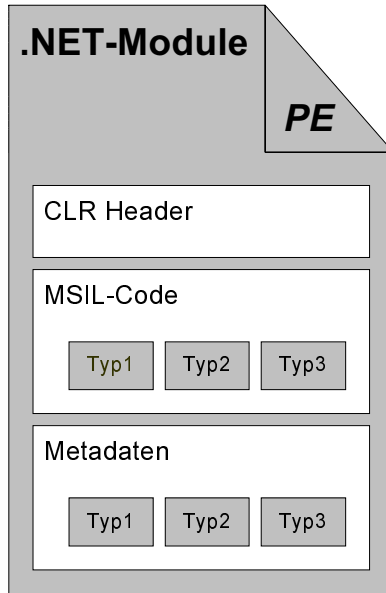


Modul

Ein .NET-Modul ist eine Einheit aus kompiliertem Programmcode im MSIL-Format und den zugehörigen Meta-Daten, die die im Code definierten Typen beschreiben. Ein .NET-Modul ist Teil einer

.NETMODULE

Assembly oder aber eine separate Datei mit der Dateierweiterung *.NETMODULE*. Die Datei liegt in beiden Fällen im Portable Executable (PE) File Format vor. Ein Modul unterscheidet sich von einer Assembly dadurch, dass es kein Manifest besitzt.



© Holger@Schwichtenberg.de 2001

Abbildung A.9: Aufbau eines Moduls in .NET

Mono

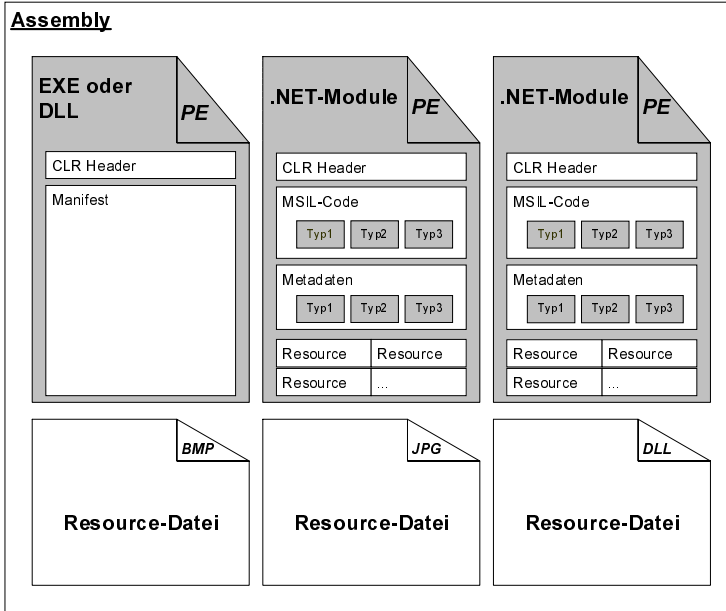
Open Source-Variante von .NET für Linux [MON02]

Multi-File-Assembly

MFA In .NET eine Assembly, die aus mehreren Dateien besteht.

My Services

Microsoft My Services (früherer Codename »Hailstorm«) sind SOAP-basierte Webservices von Microsoft.



© Holger@Schwartzklopp.de 2001

Abbildung A.10: Aufbau einer Multi-File-Assembly

Namespace

Typen werden in .NET nicht mehr durch GUIDs, sondern durch Zeichenketten eindeutig benannt. Diese Zeichenketten sind hierarchische Namen. Ein Name besteht aus dem relativen Namen des Typs und dem Namensraum (Namespace). Der relative Name muss nur innerhalb eines Namespaces eindeutig sein. Über alle Namespaces hinweg kann der Typname mehrfach vorkommen, denn der Namespace ermöglicht im Zweifel die Unterscheidung. Dies ist vergleichbar mit Dateien und Ordnern in einem Dateisystem.

Ein *Namespace* ist eine Gruppierung von Typen, die in Assemblies implementiert sind. Ein Namespace kann auch Unter-Namespaces enthalten, so dass eine Hierarchie entsteht.

Unter-Namespaces

Ein Namespace kann einen Alias haben. Die Gruppierung, also die Auswahl der Typen, die zu einem Namespace gehören, sollte nach logischen oder funktionellen Prinzipien erfolgen. Im Gegensatz dazu sollte die Zusammenfassung von Typen zu einer Assembly gemäß den Bedürfnissen der Verbreitung der Klassen (Deployment) erfolgen.

Alias

Native Code

Prozessor-spezifischer Code. Der Just-in-Time-Compiler erzeugt aus Managed Code zur Laufzeit Native Code.

Native Image

Vorkompilierung

Microsoft bietet auch eine Möglichkeit, MSIL-Code manuell in Maschinencode zu übersetzen und das Ergebnis dieser Übersetzung in einer Datei zu speichern. Eine derartige Datei heißt Native Image. Grundsätzlich kann zwar ein Native Image bereits während der Entwicklung erzeugt werden, aufgrund der plattform-spezifischen Übersetzung bietet es sich jedoch an, das Native Image erst bei der Installation einer .NET-Anwendung zu erzeugen (Install Time Compilation). Auch eine .NET-Anwendung, die in einem Native Image gespeichert ist, benötigt das .NET Framework, um ausgeführt werden zu können. Ein Native Image wird über das Werkzeug *ngen.exe* erzeugt, das Teil des .NET Framework Redistributable ist.

Native Image Cache

Microsoft spricht im Zusammenhang mit Native Images vom Native Image Cache. Der Native Image Cache ist aber nichts anderes als eine Teilmenge des GACs.

NGWS Runtime

siehe Common Language Runtime

Next Generation Windows Service

früherer Name des .NET Frameworks

Object Linking and Embedding Database

OLEDDB

Datenbankzugriffsschnittstelle, Nachfolger von ODBC

Operation Codes

Die MSIL hat 240 Operation Codes (OpCodes), ähnlich dem Java-Bytecode.

Platform Invoke

Mechanismus in .NET zur Nutzung von C-DLLs

Portable Executable

Das Portable Executable File Format ist die Microsoft-Implementierung des Common Object File Formats (COFF), das sowohl in Unix als auch vor .NET in Windows verwendet wurde.

PE

PreJITting

Kompilierung von MSIL nach OpCodes: Die Übersetzung erfolgt zur Laufzeit, allerdings nicht stückchenweise, sondern in einem Durchgang. Das Ergebnis dieser Kompilierung kann dann auch auf der Festplatte persistent gemacht werden. Plattformunabhängigkeit und plattformspezifische Optimierbarkeit bleiben gewahrt. Wenn die Laufzeitumgebung Änderungen an den MSIL-Dateien feststellt, wird die Kompilierung erneut ausgeführt.

Private Assembly

Private Assemblies sind Assemblies, die nur von einer einzigen Anwendung verwendet werden und die im Verzeichnis bzw. einem Unterverzeichnis der Anwendung gespeichert sind. Sie werden anhand ihres Dateinamens identifiziert und können einfach über das Kopieren per Dateisystem (Microsoft spricht in diesem Zusammenhang von XCOPY-Deployment) installiert werden.

Private Key (Privater Schlüssel)

Wird in einer Strongly Named Assembly dafür verwendet, eine digitale Signatur über den Hash-Wert des Inhalts der Assembly zu bilden.

Program Database

Datei mit Debug-Informationen, die während der Kompilierung einer Assembly erstellt wird; Dateiextension *.pdb*.

PDB

Project 7

An der Entwicklung von .NET waren auch verschiedene Universitäten unter dem Titel »Project 7« beteiligt.

Property

siehe Attribut

Public Key (Öffentlicher Schlüssel)

Wird in einer Strongly Named Assembly im Manifest gespeichert.

Public Key Token

Ein Public Key Token ist eine verkürzte Form eines öffentlichen Schlüssels (Public Key). Ein Public Key Token wird erzeugt, indem der Hash-Wert über einen Public Key gebildet wird. Von diesem Hash-Wert wird ein Ausschnitt (hier: 8 Bytes vom Ende) verwendet. Dies reicht in der Regel, um einen Public Key eindeutig zu identifizieren. Da es bei den Assembly-Verweisen nur darum geht, eine Assembly eindeutig zu identifizieren, reicht dieses Verfahren aus. Die Prüfung auf Integrität der Assembly, für die der komplette Public Key benötigt wird, findet dann nach dem Auffinden der Assembly statt. In der Assembly ist immer der komplette Public Key gespeichert.

Public Key-Verfahren

Verschlüsselungsverfahren, das mit einem asymmetrischen Schlüsselpaar (Public Key und Private Key) arbeitet. Wird in .NET zur Erzeugung von Strongly Named Assemblies eingesetzt.

Referenz-Typ

Ein Referenz-Typ wird auf dem Heap gespeichert, auf dem Stack gibt es einen Verweis auf die Speicheradresse im Heap. Eine Instanziierung ist erforderlich, um eine Instanz nutzen zu können. Eine Zuweisung erzeugt nur eine Kopie des Zeigers, bei Vergleichen wird auf die Identität der Zeiger geprüft. Ein Referenz-Typ hat im Gegensatz zu einem Wert-Typ die Klasse `System.ValueType` nicht in seiner Vererbungshierarchie (siehe Abbildung A.11).

Reflection

Das .NET Framework enthält eine Möglichkeit, per Programmcode auf die Meta-Daten zuzugreifen. So können Klassen und Objekte zur Laufzeit analysiert werden. Dieses Feature wird

»Reflection« genannt und über den CLR-Namespacesystem.
Reflection bereitgestellt.

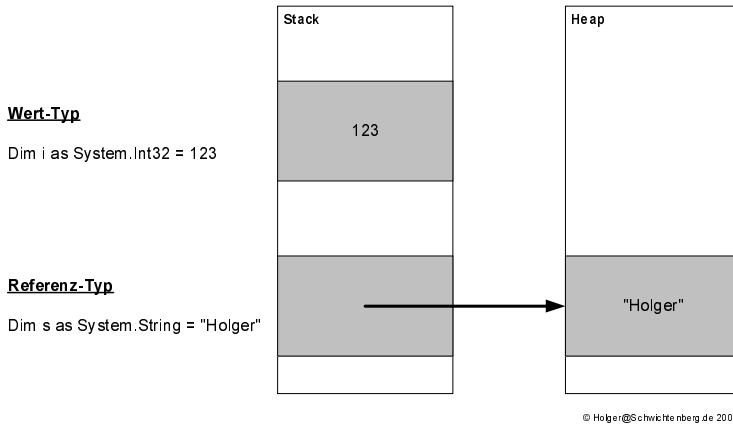


Abbildung A.11: Wert-Typ (z.B. Int32) vs. Referenz-Typ (z.B. String)

Reflection ist die verbesserte Variante dessen, was in COM mit Hilfe der TypLib-Informationskomponente (TLBINF32.DLL) möglich war.

Runtime Callable Wrapper

COM-Komponenten werden von .NET aus über einen so genannten Runtime Callable Wrapper (RCW) angesprochen. Der RCW setzt die COM-Schnittstellen in .NET-kompatible Schnittstellen um. Für COM-Automation unterstützende Schnittstellen (von IDispatch abgeleitete Schnittstellen) kann ein RCW mit dem .NET-SDK-Werkzeug TLBIMPEXE automatisch aus der Typbibliothek erzeugt werden. Für alle nicht automationsfähigen Schnittstellen muss manuell ein benutzerdefinierter RCW erstellt werden.

RCW

Runtime Host

Damit eine .NET-Anwendung ausgeführt werden kann, ist es notwendig, die CLR in den Speicher zu laden und die .NET-Anwendung zur Ausführung an die CLR zu übergeben. Dies ist die Aufgabe eines .NET Runtime Hosts (alias Application Domain Host). Für jeden Typ von .NET-Anwendungen muss es einen Runtime Host geben. Jeder Runtime Host besitzt am Anfang einen Teil Unmanaged Code, der Stub genannt wird.

Stub Dieser Stub kann nicht Managed Code sein, da die CLR ja noch nicht geladen ist. Es ist die Aufgabe des Stubs, die CLR zu laden. Dabei benutzt der Runtime Host das so genannte .NET Hosting API.

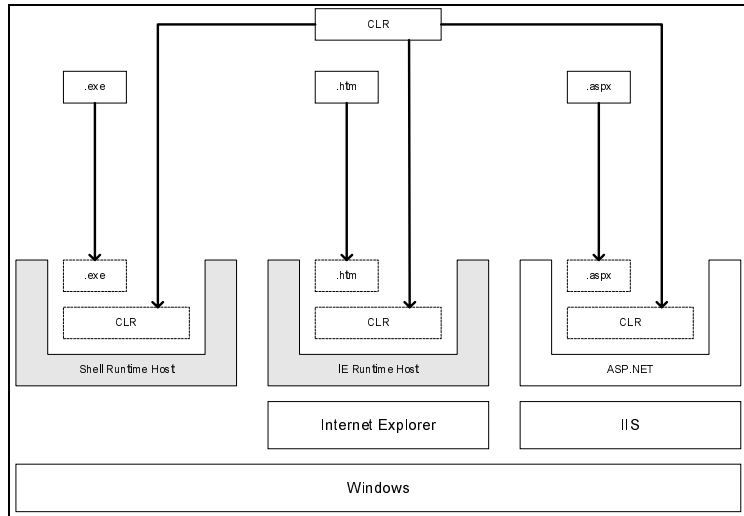


Abbildung A.12: Die drei im .NET Framework Version 1.0 verfügbaren Runtime Hosts

Schnittstelle

Schnittstelle vs. Klasse

Eine Schnittstelle (engl. Interface) ist eine Beschreibung von Attributen, Methoden und Ereignissen. Der Unterschied zur Klasse besteht darin, dass die Methoden jedoch keine Implementierung enthalten. Eine Schnittstelle entspricht einer abstrakten Klasse.

Eine Schnittstelle kann nicht instanziiert, sondern nur im Rahmen einer Klasse verwendet werden. Dort muss die Schnittstelle implementiert werden, genauer gesagt, dort müssen alle Attribute und Ereignisse deklariert und alle Methoden deklariert und implementiert werden.

Schnittstellenvererbung

In .NET kann eine Schnittstelle von genau einer anderen Schnittstelle erben. .NET-Schnittstellen können als Attribute nur Eigenschaften (Properties), nicht aber Felder (Fields) enthalten. .NET-Schnittstellen können keine Konstanten oder öffentlichen Aufzählungstypen enthalten.

Script for .NET

Scripting-Architektur für das .NET Framework. Nachfolger von Active Scripting.

Server Control

Ein serverseitiges Steuerelement in einem Webform in ASP.NET. Es gibt vordefinierte Steuerelemente und selbst definierte Steuerelemente (User Controls). Ein Server Control ist abgeleitet von der Klasse `System.Web.UI.Control`.

**ASP.NET-
Steuerelement**

Serviced Component

Auch .NET-Komponenten können Transaktionen unterstützen. Das .NET Framework enthält jedoch (noch) keinen eigenen Transaktionsmonitor, sondern greift auf die Transaktionsinfrastruktur von COM+ zurück. Auch die anderen Komponentendienste von COM+ (Object Pooling, Events, Queued Components, etc.) können verwendet werden. .NET-Komponenten, die in einer COM+-Anwendung laufen, heißen *Serviced Components*.

COM+

Die dazu notwendige Registrierung als COM+-Anwendung wird beim ersten Start durch die CLR vorgenommen (*First Use Lazy Registration*), wenn die Komponente durch so genannte Meta-Attribute entsprechend gekennzeichnet ist.

**First Use Lazy
Registration**

Shared Assembly

Shared Assemblies sind Assemblies, die im Global Assembly Cache (GAC) gespeichert und global für alle .NET-Anwendungen zugänglich sind. Shared Assemblies benötigen einen Strong Name.

GAC

Shared Name

siehe Strong Name

Side-by-Side-Execution

Im .NET Framework ist es möglich, mehrere verschiedene Versionen einer Komponente nicht nur auf der Festplatte zu speichern, sondern auch im Speicher zu halten. Dieses Verfahren heißt Side-by-Side-Execution und bekämpft die so genannte »DLL hell« (siehe DLL-Hölle).

DLL-Hölle

Simple Object Access Protocol

SOAP XML-basierter Aufruf von entfernten Methoden via HTTP. Das beim World Wide Web Consortium (W3C) entwickelte SOAP basiert auf der Extensible Markup Language (XML) und dem Hypertext Transfer Protocol (HTTP): Der Aufruf einer entfernten Routine wird in eine XML-Datenstruktur verpackt und via HTTP-Request (in der Regel auf dem Standardport 80) vom Client an den Server übertragen. Nach Abarbeitung der Routine verpackt der Server die Rückgabewerte ebenfalls in eine XML-Struktur und sendet diese per HTTP-Response zurück. Um auf einem System eine Routine per SOAP starten zu können, ist es also Voraussetzung, dass das aufgerufene System über einen HTTP-Listener verfügt. SOAP wird unter .NET eingesetzt für Remoting und Webservices.

Signed Assembly

Signierte Assembly

Der Begriff *Signed Assembly* ist doppeldeutig:

1. Eine Signed Assembly ist eine Assembly mit einem Strong Name.
2. Eine Signed Assembly ist eine Assembly mit einer Authenticode-Signatur.

Ein Strong Name garantiert einen eindeutigen Namen und die Integrität einer Assembly. Die Authenticode-Signatur garantiert die Identität des Herstellers. Beide Verfahren haben technisch nichts miteinander zu tun, können aber in einer Assembly ergänzend eingesetzt werden.

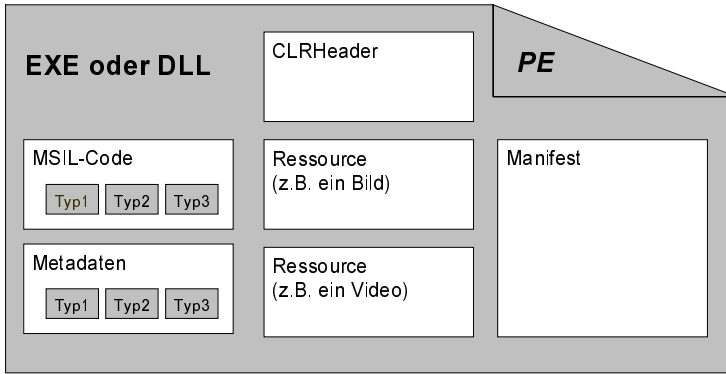
Dies ist wieder ein Beispiel für MINFU.

Single File Assembly

SFA In .NET eine Assembly, die aus nur einer Datei besteht. Diese Datei ist ein Modul mit Programmcode. In das Modul können Ressourcen hineinkompiliert sein.

Smart Device Extensions (SDE)

Die Smart Device Extensions sind eine Erweiterung für Visual Studio .NET für die Entwicklung von Anwendungen für das .NET Compact Framework.



© Holger@Schwichtenberg.de 2001

Abbildung A.13: Aufbau einer Single-File-Assembly

Die SDE umfasst eine auf normalen Windows-Versionen lauffähige Version des .NET Compact Framework, einen Designer für CF-Forms und einen Pocket PC-Emulator.

Software Development Kit

Name für Sammlungen von Dokumentationen, Beispielen und Tools zu Microsoft Entwickler-Produkten und Microsoft Servern. Es gibt verschiedene SDKs, z.B. zum Exchange Server, zu ADSI, zu .NET, etc. Das *Platform SDK* umfasst die meisten Einzel-SDKs.

SDK

Strong Name

Ein Strong Name (synonymer Begriff: Shared Name) garantiert einen eindeutigen Namen und die Integrität (Unversehrtheit) einer Assembly. Eine Shared Assembly benötigt zwingend einen Shared Name.

Shared Name

Ein Strong Name basiert auf einem Public Key-Verfahren, bei dem der endgültige Name der Assembly aus dem öffentlichen Schlüssel und dem Namen der Assembly gebildet wird. Zusätzlich wird die Assembly mit dem privaten Schlüssel signiert. Der öffentliche Schlüssel taucht als Eigenschaft *Originator* im Manifest einer Assembly auf. Durch die Umkehrung der Signatur mit Hilfe des öffentlichen Schlüssels kann der Empfänger einer Assembly sicher feststellen, ob die Datei auf ihrem Weg vom Autor zu ihm verändert wurde.

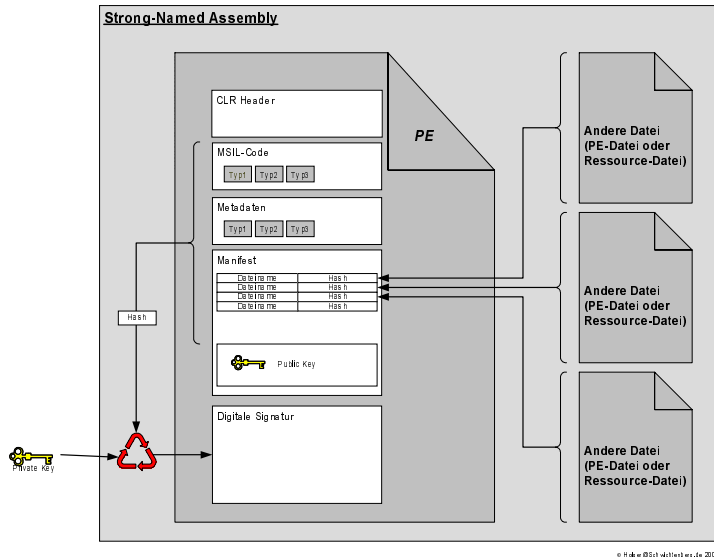


Abbildung A.14: Assembly mit Strong Name

Strong-Named Assembly

Ein Strong Name für eine Assembly (synonymer Begriff: Shared Name) sorgt dafür, dass keine Verwechslungsgefahr zwischen (gleichnamigen) Assemblies unterschiedlicher Hersteller besteht. Gleichzeitig garantiert ein Strong Name die Unversehrtheit einer Assembly. Der Strong Name garantiert aber nicht die Identität des Herstellers. Dazu ist eine Authenticode-Signatur erforderlich.

Struktur

Eine Struktur ist ein Wert-Typ. Sie unterstützt fast alle Elemente, die auch eine Klasse beinhalten kann, jedoch keine Vererbung.

Template Definition Language

TDL XML-Sprache zur Definition von Enterprise Templates in Visual Studio .NET

Transaktionen in .NET

siehe Serviced Components

Typbibliothek

Eine COM-Typbibliothek ist die formale Beschreibung der Fähigkeiten einer COM-Komponente. **TBL**

Unboxing

Umwandlung eines Referenz-Typs in einen Wert-Typ (Gegenteil: Boxing)

Universal Runtime

siehe Common Language Runtime

Unmanaged Code

siehe Managed Code

User Control

User Controls sind selbst entwickelte Steuerelemente für Windows Forms oder Webforms. Microsoft setzt zum Teil die Begriffe User Control und Custom Control gleich, zum Teil wird aber – gerade im Bereich ASP.NET – auch ein Unterschied gemacht. **.aspx**

In ASP.NET sind User Controls sind selbst entwickelte Webcontrols in Form einer HTML-Quellcode-Datei, optional mit Code-Behind-Datei. User Control haben die Dateierweiterung *.ascx* und liegen innerhalb einer Webanwendung. Ein User Control ist eine von `System.Web.UI.UserControl` bzw. `System.Web.UI.MobileControls.MobileUserControl` abgeleitete Klasse. User Controls wurden früher Pagelets genannt.

Custom Controls sind in ASP.NET ebenfalls selbstentwickelte Webcontrols, die aber in Form einer Assembly, die an jedem beliebigen Ort liegen kann. Ein Custom Control kann im Global Assembly Cache (GAC) liegen und so von jeder Webanwendung verwendet werden. Bei Custom Controls gibt es keine HTML-Quellcodedatei. Der auszugebende Code wird über eine `HtmlTextWriter`-Objekt an ASP.NET geliefert. Ein Custom Control erbt von `System.Web.UI.WebControls.WebControl` bzw. einer bestehenden Klasse eines Webcontrols. Custom Controls werden teilweise auch Advanced Controls genannt. Außerdem werden Custom Controls manchmal auch noch untergliedert in Inheritance Controls, Composite Controls und Direkt Controls. **Custom Control**

In der deutschen Dokumentation nennt Microsoft User Control »Webbenutzersteuerelemente« und die Custom Control »benutzerdefinierte Websteuerelemente«. Dies sind wenig glücklich gewählte Übersetzungen.

Validation Server Control

ASP.NET Server Control zur Überprüfung der Eingabe in Formularfeldern

Version Policy

Richtlinie in einer Konfigurationsdatei, die festlegt, welche Versionen von referenzierten Assemblies benutzt werden sollen

Versionierung in .NET

- a.b.c.d** Das .NET Framework bietet zusätzlich zur Side-by-Side-Execution eine konfigurierbare Versionskontrolle. Ein Verweis auf eine Shared Assembly erfolgt stets nicht nur über den Namen, sondern auch über die Versionsnummer. Jede Assembly hat eine viergliedrige Versionsnummer der Form a.b.c.d, z. B. 1.8.19.72. Die Versionskontrolle kann auch für Private Assemblies mit einer Konfigurationsdatei erzwungen werden.

Verwalteter Code

siehe Managed Code

View State

Repräsentation der Inhalte bzw. Zustände der Server Controls in einem ASP.NET-Webform

Visual Basic .NET (VB.NET)

- VB.NET** Die Version 7.0 der unter Windows am meisten verwendeten Programmiersprache.

VB.NET ist die Nachfolgeversion von Visual Basic 6.0 und trägt intern die Versionsnummer 7.0. VB.NET folgt der Common Language Specification (CLS) und ist daher eine mit dem .NET Framework kompatible Sprache. Der aktuellen Marketing-Strategie folgend heißt die neue Version also Visual Basic .NET.

Microsoft hat bei VB.NET ganz bewusst auf die Kompatibilität mit der Vorgängerversion VB 6.0 verzichtet. Inkompatibilitäten wurden aus zwei Gründen hingenommen:

Kompatibilität

1. Alle Features aus VB 6.0, die signifikante Verwirrung bei den Entwicklern stifteten, wurden bereinigt.
2. Alle Features aus VB 6.0, die nicht kompatibel mit dem .NET Framework waren, wurden geändert oder entfernt.

Dabei wurde zwar versucht, die Inkompatibilität zwischen VB 6.0 und VB.NET so zu gestalten, dass ein automatisiertes Upgrade möglich ist. Dies war jedoch keine Pflichtanforderung. Es gibt einen Upgrade-Wizard von VB 6.0 nach VB.NET, der jedoch nicht alle Änderungen vollziehen kann.

Im Zuge von VB.NET fusioniert Microsoft die drei Geschwister VB 6.0, Visual Basic for Applications (VBA) und VBScript zu einer Sprache. Leider wird es keine Fusion unter Gleichen, denn während Visual Basic 6.0 alle seine Funktionen in das neue Visual Basic .NET übernehmen darf, fallen die in VBScript 5.1 ergänzten Funktionen zur Interpretierung von zur Laufzeit generiertem Programmcode (`Eval` und `Execute`) der Fusion zum Opfer. Microsoft nährt jedoch die Hoffnung, dass diese Befehle in späteren Versionen von Visual Basic .NET wieder unterstützt werden.

Fusion

Visual Studio for Applications (VSA)

VSA ist eine Technologie für die Erweiterbarkeit (»Customizing«) von verteilten Anwendungen. VSA kann nicht nur in Windows-Anwendungen (als Nachfolger von Visual Basic for Applications), sondern auch in Web-Anwendungen für das Customizing eingesetzt werden.

VSA

VSA ist ein Teil von Script for .NET.



Web Server Control

siehe Server Control

Web Services Description Language (WDSL)

Beschreibungssprache für Webservices

WDSL

Webcontrol

Ein Steuerelement auf einem Webform in ASP.NET. Siehe auch Server Control.

Webform

Webforms sind die zentrale Technologie in .NET. Ein Webform besteht aus Webcontrols.

Webservice

SOAP Ein Webservice ist eine Klasse, die ihre Methoden auf einem Webserver zum Aufruf via SOAP (XML over HTTP) anbietet.

Screen Scraping Heutige Websites bieten eine interaktive, visuelle Benutzerschnittstelle in Form von HTML-Seiten mit Grafiken. Diese Art der Darstellung von Informationen richtet sich an Menschen. Für Maschinen (Anwendungen, Robots, Crawler, Agents) ist sie nicht adäquat. Maschinen müssen bislang Webseiten per Screen Scraping auslesen: Sie analysieren den HTML-Code, um Informationen herauszufiltern. Jede Änderung in der Darstellung kann dazu führen, dass die Maschine im Folgenden den Inhalt falsch interpretiert.

Ein Webservice ist ein WWW-Angebot mit einer Schnittstelle für Maschinen. Diese Schnittstelle bietet Funktionen mit definierten Parametern und Rückgabewerten. Der Aufruf dieser Funktionen erfolgt via Simple Object Access Protocol (SOAP) über HTTP.

Wert-Typ

System.ValueType Ein Wert-Typ wird auf dem Stack gespeichert. Eine Instanziierung ist für eine Instanz eines Wert-Typs nicht notwendig. Eine Zuweisung erzeugt eine Kopie des Wertes, bei Vergleichen wird auf Wertidentität geprüft. Ein Wert-Typ ist in .NET eine von `System.ValueType` abgeleitete Klasse (siehe auch *Referenz-Typ*).

B Active Directory Service Interface (ADSI)

Dieser Anhang, in dem die ADSI-Grundlagen und die ADSI-COM-Komponente vorgestellt werden, ist eine wichtige Ergänzung zu den Ausführungen in Kapitel 15 (`System.DirectoryServices`).

In Unternehmen werden heute üblicherweise mehrere Verzeichnisdienste (z.B. Windows NT-Domänen, Novell Directory Service, Microsoft Exchange Server) parallel betrieben. Jeder Verzeichnisdienst hat sein eigenes, proprietäres *Application Programming Interface (API)*, was die Entwicklung von Automatisierungslösungen erschwert.

Verzeichnisdienste

Microsoft stellt mit dem *Active Directory Service Interface (ADSI)* eine Programmierschnittstelle zur Verfügung, mit der der Zugriff auf unterschiedliche Verzeichnisdienste in einheitlicher Weise möglich ist. ADSI ist ein Meta-Objektmodell, da es mit wenigen universellen COM-Schnittstellen den Zugriff auf die Objektmodelle aller Verzeichnisdienste ermöglicht. Dieses Kapitel beschreibt das ADSI-Meta-Objektmodell sowie die grundsätzlichen Eigenschaften von Verzeichnisdiensten und von ADSI.

ADSI

Mit ADSI stellt Microsoft eine leistungsstarke Schnittstelle zur Automatisierung vieler administrativer Aufgaben zur Verfügung. Man merkt allerdings, dass ADSI eigentlich für Windows 2000/XP/.NET geschrieben wurde. Die Möglichkeiten unter NT4 sind eingeschränkt.

Die BCL-Klassen für den Zugriff auf Verzeichnisdienste (Namespace `System.DirectoryServices`) sind ein Wrapper für die COM-Komponente Active Directory Service Interface. Dies bedeutet, dass die Klassen im Namespace `System.DirectoryService` nicht funktionieren, wenn die ADSI-COM-Komponente nicht installiert ist. Die Klassen im Namespace `System.DirectoryService` bieten nur

ADSI unter .NET

sehr allgemeine Mechanismen für den Zugriff auf Verzeichnisdienste. Im Gegensatz zum COM-ADSI gibt es keine spezifischen Klassen mehr für einzelne Verzeichnisdienste. Bestimmte Operationen (z.B. Ändern des Kennworts in einem Benutzer-Objekt) müssen daher direkt oder indirekt über die ADSI-COM-Komponente aufgerufen werden.

Weitere Informationen

Weitere Informationen zu ADSI finden Sie in folgenden Büchern:

Holger Schwichtenberg
COM-Komponentenhandbuch: Systemprogrammierung und Scripting mit COM-Komponenten
ISBN 3-8273-1936-6
erschienen im Jahr 2001 bei Addison-Wesley

Holger Schwichtenberg
Windows Scripting: Automatisierte Systemadministration mit Visual Basic/VBScript und COM/.NET
2. Auflage
ISBN 3-8273-1843-2
erschienen im Jahr 2001 bei Addison-Wesley

B.1 Verzeichnisdienste

Was ist ein Verzeichnisdienst?

Ein Verzeichnis ist – sehr allgemein gesagt – ein hierarchischer Speicher für Informationen. Der Begriff *Verzeichnisdienst* (*Directory Service*) umfasst neben der Speicherung auch den Zugriff auf die Informationen.

Verzeichnisdienste sind in Unternehmensnetzwerken sehr populär, weil sie eine zentrale Verwaltung ermöglichen. Ein Verzeichnisdienst dient in Unternehmen der Verwaltung von Benutzern und Ressourcen (z.B. Computer, Drucker, Windows-Dienste) in einem Netzwerk. Sein Aufbau ist üblicherweise hierarchisch, so dass ein Verzeichnisbaum (*Directory Tree*) entsteht – nicht zu verwechseln mit dem Verzeichnisbaum des Dateisystems, obwohl er prinzipiell sehr ähnlich ist. Einem Verzeichnisdienst liegt eine Datenbank zugrunde, die aber nicht notwendigerweise eine relationale Datenbank ist.

Ein Verzeichnisdienst ist in der Regel ein verteiltes System. Rechner, die Leistungen im Rahmen eines Verzeichnisdienstes erbringen, werden *Directory System Agents (DAS)* genannt. Rechner, die Leistungen nutzen, heißen dagegen *Directory User Agents (DUA)*.

DAS und DUA

Die hier verwendete Begrifflichkeit orientiert sich an dem Standard ITU X.500 (ISO/IEC 9594), der heute Basis für viele Verzeichnisdienste ist. X.500 definiert den Aufbau eines Verzeichnisses, verschiedene Dienste für die Arbeit mit dem Verzeichnis und ein Zugriffsprotokoll. Eine vollständige Implementierung von X.500 ist jedoch aufgrund der Komplexität des Standards nur in wenigen Verzeichnisdiensten vorhanden. Weitere Informationen zu X.500 finden Sie in [CHA96].



Verzeichnisdienste sind heute in viele Systeme integriert, allen voran Betriebssysteme und Messaging-Systeme. Ob man die Verwaltung der Benutzer und Ressourcen in Windows NT4 einen Verzeichnisdienst nennen darf, ist aber oft eine Streitfrage, weil die Möglichkeit fehlt, Ressourcen hierarchisch zu organisieren. Verzeichnisdienste für Unternehmensnetzwerke bieten die Firmen Banyan (mit dem Produkt Banyan Vines) und Novell (mit dem Novell Directory Service NDS) schon lange an. Microsoft ist mit dem Active Directory eher spät dran.

Verzeichnisdienste in der Praxis

B.1.1 Aufbau eines Verzeichnisbaums

Ein Verzeichnisbaum enthält Verzeichniseinträge, wobei es drei Typen von Einträgen gibt: *Root (Wurzel)*, *Container (Knoten)* und *Leaves (Blätter)*. Die Wurzel eines Verzeichnisbaums ist darin das oberste Element und bildet den Einstiegspunkt zu den weiteren Einträgen. Container können andere Einträge enthalten, während Blätter das Ende eines Zweigs in einem Verzeichnisbaum darstellen.

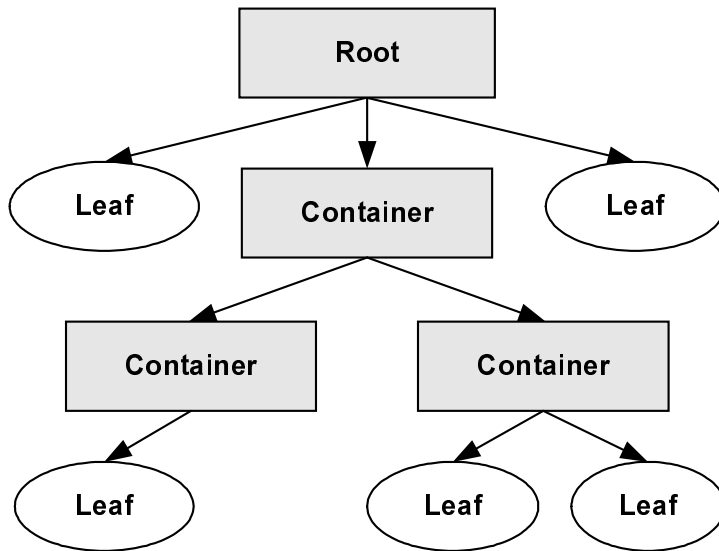
Verzeichnisbäume

Ein Namespace ist eine Zusammenfassung mehrerer Verzeichnisbäume eines Verzeichnisdienstes. Beispielsweise bildet Windows NT 4.0 einen Namespace, der aus den Domänen und Arbeitsgruppen besteht.



Jeder Eintrag in einem Verzeichnisbaum hat bestimmte Attribute. Jedes Attribut hat einen Wert, manche Verzeichnisdienste erlauben auch Mehrfachwerte für Attribute, z.B. die Ablage mehrerer Nummern für die Telefonnummer.

Attribute



(C) Holger@Schwichtenberg.de 2002

Abbildung B.1: Aufbau eines Verzeichnisdienstes

Objekte und Klassen

Objekte und Klassen

Einträge in Verzeichnisdiensten werden oft auch Objekte genannt. Im Vergleich zum Objektbegriff in der objektorientierten Programmierung verfügen Objekte in Verzeichnisdiensten nur über Eigenschaften in Form von Attributen, nicht aber über ein Verhalten in Form von Methoden. Operationen auf einem Verzeichnisdienst werden durch Zugriffsprotokolle wie das Lightweight Directory Access Protocol (LDAP) bzw. Verzeichnisdienstwerkzeuge definiert. Jeder Eintrag in einem Verzeichnisdienst wird auf Basis einer bestimmten Vorlage erzeugt. Die Vorlage wird üblicherweise Klasse genannt.



Die Verwendung der Begriffe Objekt und Klasse für Verzeichnisdiensteinträge bedeutet nicht, dass der Verzeichnisdienst mit objektorientierten Methoden implementiert wurde. Dies verhält sich genauso wie bei einer grafischen Benutzeroberfläche, die von einigen auch schon objektorientiert genannt wird, nur weil jedes Icon ein anderes Kontextmenü hat.

Semi-Strukturierung

Anders als in einer strukturierten Datenbank kann es in Verzeichnisdiensten optionale Attribute geben. Ein konkretes Objekt muss für ein optionales Attribut keinen Wert bereitstellen. Ein Verzeichnisdienst ist also vergleichbar mit einer semi-strukturierten Datenbank wie beispielsweise einem Message Store. Anders als eine Nachricht in einem Message Store kann ein Verzeichnisdienstobjekt jedoch nicht über die in der Verzeichnisdienstklasse definierten Attribute hinausgehen. Ein Verzeichnisobjekt hat die Möglichkeit, einige der vordefinierten Attribute wegzulassen; um neue hinzuzufügen, muss die Klasse geändert werden.

Semi-Strukturierung

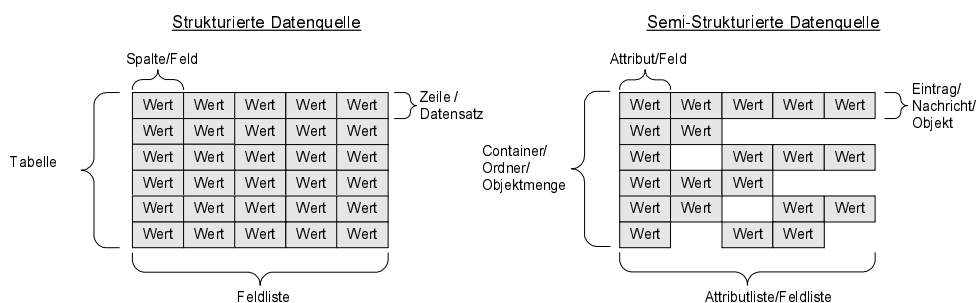


Abbildung B.2: Strukturierte vs. semi-strukturierte Daten

B.1.2 Verzeichnisdienstschemata

Verzeichnisdienste besitzen ein Schema, das eine abstrakte Beschreibung des Verzeichnisdienstes auf einer Meta-Ebene darstellt. Das Schema ähnelt einem Wörterbuch und enthält:

1. die Definition der Attribute und ihrer Syntax

Attribute

Die Syntax besteht aus den erlaubten Datentypen, dem erlaubten Wertebereich und der Unterscheidung zwischen einem Einfach- und einem Mehrfachattribut.

2. die Definition der Klassen

Klassen

Dabei wird auch definiert, ob eine Klasse ein Container oder ein Leaf ist, welche Attribute eine Klasse enthalten muss (Pflichtattribute) und welche enthalten sein können (optionale Attribute).

**Vererbungs-
hierarchie**

3. die Vererbungshierarchie der Klassen
Auch in Verzeichnisdiensten können Klassen von anderen Klassen abgeleitet sein. Meist ist auch Mehrfachvererbung möglich.

**Containment-
Hierarchie**

4. die Containment-Hierarchie
Es wird definiert, welche Instanzen einer anderen Klasse ein konkretes Objekt enthalten können (Beispiel: Ein Eintrag *Familie* kann Objekte der Klasse *Kinder* enthalten, nicht aber Objekte der Klasse *Garten*). Daraus ergibt sich zur Laufzeit die Objekthierarchie des Verzeichnisdienstes.

**Transparenz und
Schema-Modifikationen**

Das Schema ist in der Regel transparent, also selbst wiederum Teil des Verzeichnisdienstes und oft auch auslesbar. In vielen Verzeichnisdiensten ist darüber hinaus sogar der Aufbau des Schemas selbst wieder Teil des Schemas. Hier kann man also die bekannte Huhn-Ei-Frage stellen: Was war zuerst da? Einige Verzeichnisdienste ermöglichen es, das Schema zu verändern oder zu erweitern. Dies nennt man eine Schemamodifikation.

B.1.3 Objektidentifikation**DN und RDN**

Verzeichnisdienste sind in der Regel hierarchisch. Einzelne Einträge in einem Verzeichnisdienst werden daher über hierarchische Pfadangaben, so genannte *Fully Qualified Distinguished Names* (FQDNs) oder einfacher *Distinguished Names* (DNs) gebildet. Davon abzugrenzen ist der *Relative Distinguished Name* (RDN). Der RDN identifiziert (in der Regel) einen Eintrag innerhalb eines Containers eindeutig. In einem anderen Container darf der RDN sehr wohl erneut vorkommen. Der DN besteht aus dem RDN eines Eintrags und dem DN des übergeordneten Eintrags. Auf diese Weise gibt es zu jedem Eintrag einen eindeutigen Pfad. Die Bildung des DN ist abhängig vom Verzeichnisdienst. Der RDN ist vergleichbar mit einem Dateinamen, der DN mit einem kompletten Dateipfad.

**Objektidentifizierung bei NT4**

Es gibt Verzeichnisdienste, die keine Eindeutigkeit des RDNs innerhalb eines Containers fordern. Bei NT4 ist die Kombination aus RDN und Klassenname der »Schlüssel«. Innerhalb eines Containers darf es sehr wohl mehrfach den gleichen RDN geben, solange die gleichnamigen Objekte jeweils zu einer anderen Klasse gehören.

B.1.4 Meta-Directories

Ein Meta-Directory ist ein Dienst, der verschiedene Verzeichnisdienste zu einem zusammenfasst. Anstatt Konnektoren zwischen verschiedenen Verzeichnisdiensten zu schaffen, setzt der Meta-Directory-Ansatz auf eine übergeordnete Kontrollinstanz. Microsoft hat nach der Akquisition des Meta-Directory-Server-Herstellers Zoomit ein eigenes Produkt unter dem Namen *Microsoft Meta Directory Server* veröffentlicht.

Meta-Directories

B.1.5 Lightweight Directory Access Protocol (LDAP)

Über X.500 definiert ist auch ein Zugriffsprotokoll, mit dem DSAs und DSUs kommunizieren können. Dieses Protokoll heißt *Directory Access Protocol (DAP)*. Wie viele andere Protokolle der X-Serie konnte sich DAP aufgrund seiner Komplexität nicht durchsetzen. Von Yeong, Hoews und Kille wurde daher 1995 an der University of Michigan eine abgewandelte Form von DAP entwickelt, das *Lightweight Directory Access Protocol (LDAP)*. LDAP wurde von der Internet Engineering Task Force (IETF) standardisiert. Aktuell ist die Version 3.0 (zu LDAP 2.0 vgl. [RFC1777] bis [RFC1779] sowie [RFC1798], zu LDAP 3.0 vgl. [RFC2254] bis [RFC2256]). LDAP beruht auf TCP/IP und verwendet die Portnummer 389. LDAP in Verbindung mit dem Secure Socket Layer-Protokoll (LDAP over SSL) benutzt Port 636.

LDAP

Während LDAPv1 nur in der Lage war, nach Einträgen zu suchen und deren Attribute zurückzuliefern, unterstützt LDAPv3 folgende Verzeichnisoperationen: search, compare, add, delete, modify, modifyRDN.

Versionen

LDAP definiert im Gegensatz zu DAP nicht den Aufbau eines Verzeichnisdienstes, sondern nur ein Kommunikationsprotokoll. Mit LDAP kann sowohl auf X.500-kompatible als auch auf proprietäre Verzeichnisse zugegriffen werden. Voraussetzung ist lediglich die Unterstützung des LDAP-Protokolls.



Akzeptanz von LDAP

Die Akzeptanz von LDAP steigt ständig: Zum einen sind viele öffentliche Verzeichnisdienste im Internet durch LDAP ansprechbar (z.B. *ldap.bigfoot.com*, *ldap.whowhere.com*, *ldap.infospace.com*, *ldap.yahoo.com*). Zum anderen integrieren die Hersteller von Verzeichnisdiensten LDAP in ihre Systeme (z.B. Windows 2000 Active Directory, Microsoft Exchange seit Version 5.0, Microsoft Site Server, Lotus Notes, Netscape Directory Server).

Akzeptanz

LDAP-Clients

LDAP-Anwendungen

Ein einfacher LDAP-Client befindet sich inzwischen – von vielen bislang unbemerkt – auf fast jedem Windows-PC. Sowohl der Microsoft Internet Explorer (ab Version 4.0) als auch der Netscape Communicator (ab Version 4.0) enthalten einen LDAP-Client. Bei installiertem Internet Explorer ist der LDAP-Client auf drei Wegen erreichbar:

- ▶ durch Eingabe eines beliebigen LDAP-URLs an der URL-Zeile des Internet Explorers (z.B. *LDAP://kom.wi-inf.uni-essen.de*)
- ▶ durch den Menüpunkt BEARBEITEN/PERSONEN SUCHEN in Outlook Express
- ▶ mit dem Windows-Startmenü über SUCHEN/PERSONEN

In allen drei Fällen erscheint der LDAP-Client in Form eines PERSONEN SUCHEN-Fensters. Die zu durchsuchenden LDAP-Server können Sie in Outlook Express unter EXTRAS/KONTEN/VERZEICHNISDIENST einrichten.

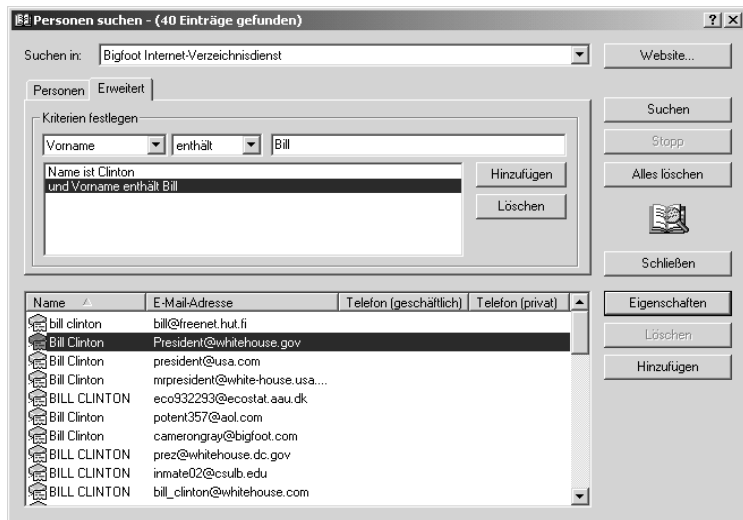


Abbildung B.3: Die Suche nach Bill Clinton liefert einige Ergebnisse im LDAP-Client des Internet Explorers.

ADB Der mit der ADSI-Installation mitgelieferte *Active Directory Browser (ADB)* ist dagegen ein vollwertiger LDAP-Client. Der ADB wird in Kapitel 1.3.1 vorgestellt.

LDAP-Programmierung

LDAP bietet eine eigene Programmierschnittstelle, die in RFC-1823 spezifiziert ist. Unter Windows findet sich das API in der C-basierten *wldap32.dll*. Im Zusammenhang mit der Microsoft-Strategie, von der Komplexität verschiedener APIs durch einfachere COM-Komponenten zu abstrahieren, die auch für den Einsatz in Scripting-Umgebungen geeignet sind, wird LDAP durch die COM-Komponente ADSI gekapselt.

WLDAP32.DLL

LDAP-Standardattributnamen

Der LDAP-Standard definiert Standardattribute für Benutzer auf Grundlage von X.500 [RFC2256]. Diese Attributnamen sind eine Empfehlung, aber keine Vorschrift. Einige dieser Namen sind sehr kurz, andere kurioserweise sehr lang.

Attribute

LDAP-Name	Erläuterung
C	Country
O	Organization
OU	Organizational Unit
DC	Domain Component
CN	Common Name
L	Locality Name
facsimileTelephoneNumber	(Ausdruck hat keine Abkürzung)
physicalDeliveryOfficeName	(Ausdruck hat keine Abkürzung)

Tabelle B.1: Ausgewählte LDAP-Standardattribute für Benutzer-Einträge

Die verschiedenen Verzeichnisdienste verwenden intern jedoch andere, proprietäre Namen.

Objektidentifikation bei LDAP

In LDAP sind alle Verzeichnisobjekte über einen URL ansprechbar. Ein kompletter LDAP-URL besteht aus der Namespace-ID LDAP, dem Namen des anzusprechenden LDAP-Servers sowie einem Distinguished Name (DN). Der LDAP-Server muss mit angegeben werden, da der DN keinen Aufschluss darüber gibt, welcher Server den Verzeichnisdienst bereitstellt. Ein Verzeichnisdienst kann auch von mehreren Servern bereitgestellt werden, so

LDAP-URLs

dass mit diesem Parameter zwischen den Servern gewählt werden kann. Der Aufbau des LDAP-URLs ist in [RFC1959] dokumentiert.

Attribut-Wert-Paare

Ein LDAP-DN wird durch Attribut-Wert-Paare gebildet. Ein Relative Distinguished Name (RDN) ist genau ein Attribut-Wert-Paar der Form `Attributname=Wert`. Ein Distinguished Name (DN) besteht aus der Aneinanderreihung solcher Attribut-Wert-Paare. Der DN unterscheidet eine *Top-Down*- und eine *Bottom-Up-Form*. Bei der Top-Down-Form wird das höchste Objekt im Verzeichnisbaum zuerst genannt, bei der Bottom-Up-Form zuletzt. Für die Bezeichnungen Top-Down-Syntax und Bottom-Up-Syntax gibt es in Anlehnung an die Frage in Gullivers Reisen, an welchem (dem großen oder dem kleinen) Ende das Ei aufzuschlagen sei, auch die Bezeichnungen *Big Endian* und *Little Endian*.

Ein Schlüsselattribut für jede Klasse

Der im DN zu verwendende Attributname wird in jeder Klasse einzeln als Schlüsselattribut festgelegt und ist keineswegs vorgegeben. Es ist üblich, die Attributnamen `O`, `OU` und `CN` als Schlüsselattribute zu verwenden. Der Wert eines Schlüsselattributs wird auch als *Common Name* eines Verzeichniseintrags bezeichnet. Auch der Attributname `CN` steht für Common Name, was ein wenig verwirrend ist. Letztlich bedeutet es aber, dass man üblicherweise dem Attribut, das den Common Name enthält, den Namen *Common Name* gibt. Die Beispiele `O` und `OU` zeigen aber, dass es auch Abweichungen davon geben kann.

Eintragsart	Beispiel Top-Down-Syntax (auch: Big Endian)	Beispiel Bottom-Up-Syntax (auch: Little Endian)
Root-Element eines LDAP-Servers	<code>LDAP://Sonne</code>	<code>LDAP://Sonne</code>
Recipients-Container eines Exchange Servers	<code>LDAP://Sonne/ o=UNIVERSUM/ ou=SONNENSYS- TEM/cn=Recipients/ cn=Mitarbeiter</code>	<code>LDAP://Sonne/ cn=Mitarbeiter, cn=Recipients, ou=SONNENSYS- SYSTEM, o=UNIVERSUM</code>

Tabelle B.2: Beispiele für LDAP-Pfade

Eintragsart	Beispiel Top-Down-Syntax (auch: Big Endian)	Beispiel Bottom-Up-Syntax (auch: Little Endian)
Benutzer in einem Active Directory	LDAP://sonne2000/ DC=de/DC=IT-Visi- ons/CN=Users/ CN=HS	LDAP://sonne2000/ CN=HS,CN=Users, DC=IT-Visions,DC=de
Konfiguration des POP3-Protokolls eines Exchange Servers	LDAP://Sonne/ o=UNIVERSUM/ ou=SONNENSYS TEM/cn=Configura- tion/ cn=Protocols/ cn=POP3	LDAP://Sonne/ cn=POP3, cn=Protocols, cn=Configuration, ou=SONNENSYS TEM,o=UNIVERSUM

Tabelle B.2: Beispiele für LDAP-Pfade (Fortsetzung)

B.2 ADSI-COM-Komponente

Das *Active Directory Service Interface* (ADSI) ist eine COM-Komponente (*activeds.dll*, Typbibliothek: *activeds.tlb*) für den Zugriff auf Verzeichnisdienste. Dabei ist ADSI nicht, wie der Name vermuten ließe, auf das Active Directory in Windows 2000 beschränkt. Die ADSI-Architektur ermöglicht es, auf unterschiedliche Arten von Verzeichnisdiensten zuzugreifen. Das allgemeine ADSI-Meta-Objektmodell bietet eine einheitliche Sichtweise auf verschiedene Verzeichnisdienste. Unter dem Namen JADSI gibt es auch eine Java-Implementierung von ADSI.

Die aktuelle ADSI-Version ist 2.5.

ADSI-Architektur



B.2.1 Architektur

ADSI bedient sich des Provider-Konzepts. Ein ADSI-Provider ist eine Stellvertreterkomponente für den Zugriff auf einen Verzeichnisdienst. Er implementiert den Zugriff auf einen bestimmten Verzeichnisdienst, indem er das ADSI-Objektmodell auf die API-Ebene des Verzeichnisdienstes umsetzt. Ein Verzeichnisdienst kann also durch ADSI ausgelesen und verändert werden, wenn ein ADSI-Provider für diesen Verzeichnisdienst existiert. ADSI-Provider werden in Form einer In-process-Komponente (COM-DLL) ausgeliefert. Im Gegensatz zum Begriff ADSI-Provider wird

ADSI-Provider sind Stellvertreter

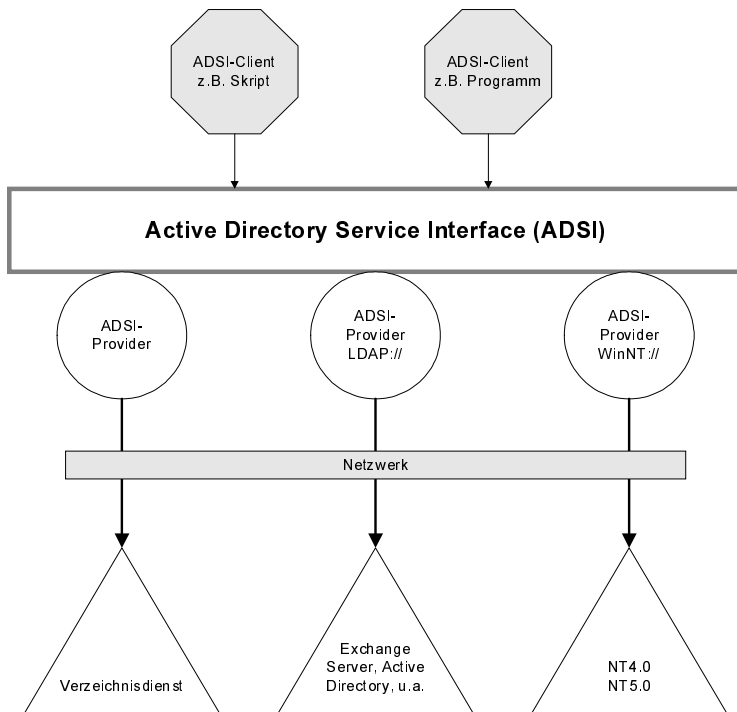
ein Programm, das ADSI nutzt, ADSI-Client genannt. Da ADSI in weiten Teilen COM-Automation unterstützt, können auch Skripte als ADSI-Clients eingesetzt werden.

Vergleich mit ADO

ADSI kann auf konzeptioneller Ebene verglichen werden mit ADO in der Universal Data Access-Architektur; die ADSI-Provider entsprechen dann den OLE DB-Providern. In ADSI definiert ein Provider einen Namespace. Jeder Namespace hat eine Name-space-ID.

ADSI-Extensions

Die in ADSI-Providern implementierten Klassen können durch so genannte ADSI Namespace Extensions um zusätzliche Schnittstellen erweitert werden. Beispiele für ADSI-Extensions sind die RAS-Komponente und CDO for Exchange Management. ADSI-Extensions können allerdings nicht mit Skriptsprachen oder Visual Basic entwickelt werden.



(c) Holger Schwichtenberg 2000

Abbildung B.4: ADSI-Architektur

B.2.2 Installation

Als Plattform für 32-Bit-ADSI-Clients unterstützt Microsoft natürlich nur die eigenen 32-Bit-Betriebssysteme. Es gibt verschiedene Add-ons für Windows 95/98/ME und für NT4. ADSI ist fester Bestandteil von Windows 2000, unabhängig davon, ob das Windows 2000 Active Directory installiert ist. Die entsprechenden Add-ons für die anderen Plattformen finden Sie auf der Buch-CD [CD:/install/adsi]. Kommende Updates erhalten Sie auch über die Microsoft ADSI-Website [ADS02]. Aktuell ist die ADSI-Version 2.5.

Verfügbarkeit für verschiedene Plattformen

Die Tabelle zeigt die Verfügbarkeit von ADSI.

ADSI gehört zum Installationsumfang des Betriebssystems	ADSI muss zusätzlich installiert werden
Windows 2000	Windows 95
Windows ME	Windows 98
Windows XP	Windows NT 4.0
Windows .NET	

Tabelle B.3: Verfügbarkeit von ADSI

ADSI muss nur auf dem Computer installiert sein, auf dem der ADSI-Client läuft. Eine Installation auf den anzusprechenden Verzeichnisdienst-Servern ist nicht notwendig. Zur Kommunikation verwendet ADSI das jeweilige Kommunikationsprotokoll des Verzeichnisdienstes.



B.2.3 Verfügbare ADSI-Provider

Für ADSI 2.5 liefert Microsoft folgende ADSI-Provider:

Provider	Namespace-ID	Provider-DLL
Windows NT	WinNT	<i>Adsnt.dll</i>
Lightweight Directory Access Protocol	LDAP und GC	<i>Adslap.dll</i>
Novell NetWare 3.x	NWCOMPAT	<i>Adsnw.dll</i>
Netware Directory Service 4.x	NDS	<i>Adsnds.dll</i>
Microsoft Internet Information Server Metabase	IIS	<i>Adsiis.dll</i>

Tabelle B.4: Verfügbare ADSI-Provider von Microsoft

Ausnahme IIS-Provider	Mit Ausnahme des IIS-Providers werden alle Provider durch das ADSI-Add-on bzw. durch das Windows 2000-Setup installiert. Der IIS-Provider wird mit dem Internet Information Server bzw. den entsprechenden Client-Tools (MMC-Snap-In) installiert.
Installierte Provider	Eine Liste der installierten ADSI-Provider befindet sich in der Registry unter <i>HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\ADs\Providers</i> . Die obige Tabelle enthält auch die Namespace-ID, die aus Sicht von COM die ProgID der Provider-Komponente ist und in Monikern verwendet wird, um den Namespace oder Objekte in dem Namespace anzusprechen. Zu der einen ADSI-Provider implementierenden DLL handelt man sich wie üblich über ProgID und CLSID durch.
LDAP und GC	Mit dem LDAP-Provider ist grundsätzlich der Zugriff auf alle LDAP-fähigen Verzeichnisdienste möglich, also insbesondere auf Windows 2000 Active Directory, Microsoft Exchange Server, Netscape Directory Server und Lotus Notes. Die Namespace-ID GC bezeichnet den Global Catalogue im Active Directory und ermöglicht einen domänenübergreifenden Zugriff.
WinNT	Der WinNT-Provider dient sowohl dem Zugriff auf ein Windows NT4-Verzeichnis als auch dem auf ein Windows 2000-System ohne installiertes Active Directory. Ein Windows 2000-Rechner ohne ADS ist nicht per LDAP ansprechbar. Der WinNT-Provider kann auch das Active Directory ansprechen, allerdings ist die Arbeit mit dem AD über diesen Provider großen Einschränkungen unterworfen und daher nicht empfehlenswert.

B.2.4 ADSI Software Development Kit

ADSI-SDK	Microsoft bietet zu ADSI ein Software Development Kit (SDK) an, in dem Include-Dateien, Beispiele und ein Resource Kit mit nützlichen zusätzlichen COM-Objekten zu finden sind (z.B. für Zugriff auf Verzeichnis- und Registry-Sicherheit, Benutzer-RAS-Konfiguration und ADSI-Fehlerinformationen). Das ADSI-SDK ist Teil des Plattform-SDKs im Microsoft Developer Network (MSDN), aber auch separat verfügbar [ADS02]. Mit dem SDK können ADSI-Provider für eigene Verzeichnisdienste entwickelt werden; das kann für Business-Anwendungen oder eigene Dienste durchaus auch sinnvoll sein, wenn deren Informationsraum eine bestimmte Komplexität erreicht. Allerdings müssen solche Provider mit C++ geschrieben werden.

B.2.5 Meta-Objektmodell

Einen Verzeichnisdienst in einer objektorientierten Komponente abzubilden ist ein nahe liegender Ansatz, da Syntax und Semantik von Verzeichnisdienst-Klassen und -Objekten den Klassen und Objekten im komponentenorientierten Sinne sehr ähnlich sind.

Die einzelnen Klassen eines Verzeichnisdienstes lassen sich als COM-Klassen mit Attributen und Methoden gut abbilden. Dabei werden die Methoden nicht durch das Verzeichnis selbst (Verzeichnisklassen haben kein Verhalten), sondern durch die COM-Komponente oder das API des Verzeichnisdienstes bereitgestellt.

Der ADSI-Kern definiert einen Satz von abstrakten ADSI-Standardschnittstellen, liefert aber keine Implementierung dazu. Die Aufgabe der Implementierung fällt den ADSI-Providern zu, die aus diesen abstrakten Schnittstellen Klassen für den jeweiligen Verzeichnisdienst zusammenstellen. ADSI arbeitet sehr intensiv mit Mehrfachschnittstellen. Die Klassen heißen in jedem Verzeichnisdienst anders. Der COM-Polymorphismus auf Schnittstellenbasis ermöglicht es dennoch, diese unterschiedlichen Klassen gleich zu behandeln.

Entsprechend dem Aufbau eines Verzeichnisdienstes kennt ADSI zwei Typen von Klassen:

- ▶ *Directory Container-Objekte*, die andere Objekte enthalten können
- ▶ *Directory Leaf-Objekte*, die die Blätter des Baums bilden und keine weiteren Objekte enthalten

Meta-Schnittstellen

Unter den ADSI-Schnittstellen gibt es einige, die Meta-Schnittstellen sind. Diese Schnittstellen ermöglichen es, auf die gesamte Funktionalität einer ADSI-Klasse zuzugreifen. Solche Meta-Schnittstellen sind

- ▶ IADs (für alle Objekte) und
- ▶ IADsContainer (für Container-Objekte).

Ein ADSI-Provider hat die Wahl, nur diese Meta-Schnittstellen zur Verfügung zu stellen oder aber auch andere konkrete Schnittstellen in den ADSI-Klassen zu implementieren.

ADSI und COM

Standardschnittstellen

Meta-Schnittstellen

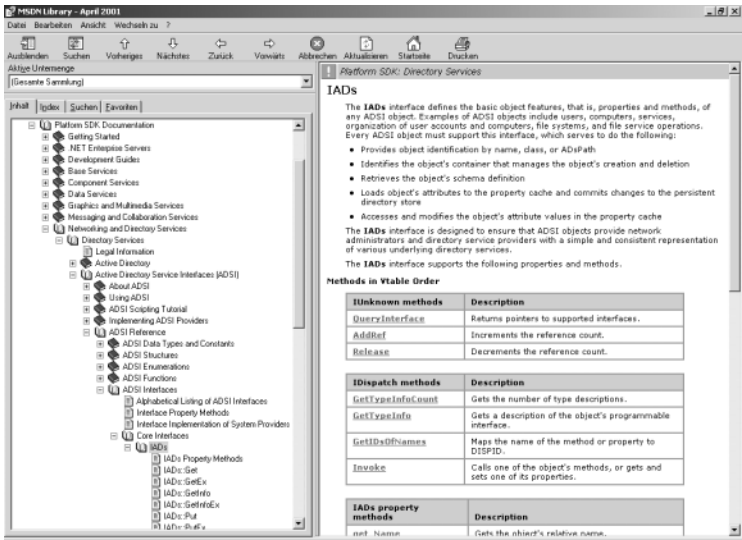


Abbildung B.5: Die MSDN Library dokumentiert zu ADSI Schnittstellen, keine Klassen.

Beispiel Die in der folgenden Abbildung dargestellte Klasse Computer aus dem WinNT-Provider besitzt neben den beiden Meta-Schnittstellen IADs und IADsContainer (mit weißen Bullets) auch noch zwei konkrete Schnittstellen IADsComputer und IADsComputerOperations (mit grauen Bullets).

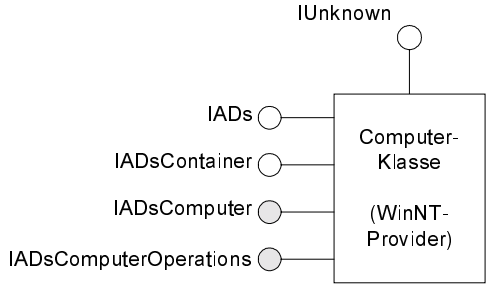


Abbildung B.6: Eine ADSI-Klasse mit Meta-Schnittstellen und konkreten Schnittstellen

Erweiterbarkeit

Die ADSI-Schnittstellen decken nur einen Standardfall ab. In der Praxis geht die Funktionalität eines konkreten Verzeichnisdienstes immer über die Möglichkeiten der ADSI-Standardschnittstellen hinaus. Um die darüber hinausgehenden spezifischen Eigenschaf-

ten eines Verzeichnisdienstes abzudecken, kann ein ADSI-Provider folgende Maßnahmen ergreifen:

- ▶ Der Provider kann zusätzliche (verzeichnisdienstspezifische) Attribute definieren, die dann nur über die Meta-Schnittstelle IADs verwendet werden können.
- ▶ Der Provider kann auch einzelne Attribute aus den ADSI-Standardschnittstellen nicht implementieren. Dies steht natürlich dem Vertragscharakter einer Schnittstelle entgegen, ist aber in diesem Fall akzeptabel, da sonst eine unübersichtlich große Anzahl von Schnittstellen notwendig wäre, um die Eigenarten aller Verzeichnisdienste abzudecken.
- ▶ *Schließlich kann der Provider auch eigene Schnittstellen definieren. Dies ist notwendig, wenn eine ADSI-Klasse Methoden bereitstellen will, die in keiner ADSI-Standardklasse existieren. Die Meta-Schnittstellen IADs und IADsContainer stellen nämlich keinen Mechanismus bereit, um verzeichnisdienstspezifische Methoden abzubilden!*

Mehr Attribute

Weniger Attribute



Weitere Schnittstellen

An dieser Stelle würde es dann Probleme für den Skriptentwickler geben, denn er könnte diese zusätzlichen Schnittstellen ja nicht nutzen. Die IDispatch-Methoden von IADs machen aber genau das, was von einer Standardschnittstelle im Rahmen der COM-Automation verlangt wird: Die IADs-Schnittstelle bildet die gesamte Funktionalität der Klasse ab, also auch die Methoden aus den zusätzlichen Schnittstellen. Fazit: Als Skriptentwickler müssen Sie sich um die Schnittstelle keine Sorgen machen. Beim frühen Binden in anderen Umgebungen können Sie über IADs zwar auf einige Attribute, aber nicht auf alle Methoden zugreifen.

Skriptentwickler haben es einfacher

Meta-Objektmodell

Die folgende Abbildung zeigt das ADSI-Meta-Objektmodell. Neben den konkreten Container-Klassen Namespaces und Namespace enthält das Meta-Objektmodell eine Container-Klasse und eine Blatt-Klasse. Beide haben keinen konkreten Namen, da diese Klasse ja erst durch den Provider realisiert wird. Dargestellt sind aber die Schnittstellen, die Container- bzw. Blatt-Klasse implementieren müssen.

Meta-Objektmodell

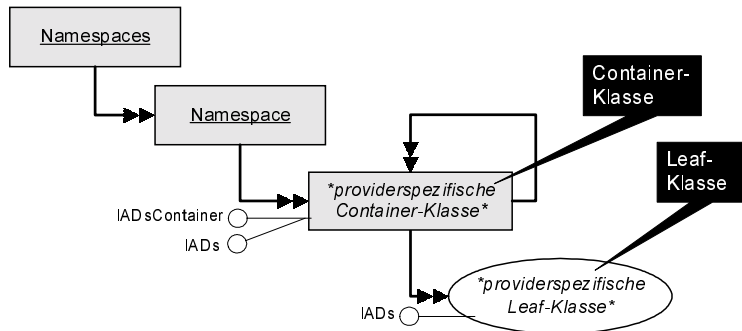


Abbildung B.7: ADSI-Meta-Objektmodell



Container versus Collections

ADSI durchbricht das Prinzip der üblichen COM-Objektmodelle, die aus der Abfolge Objekt-Collection-Objekt bestehen. Ein ADSI-Container unterscheidet sich von einer normalen COM-Collection insofern, als die einzelnen Unterobjekte selbst direkt wieder Unterobjekte enthalten können.

Klassen des ADSI-Kerns

Der ADSI-Kern implementiert selbst auch einige wenige Klassen (z.B. Namespace, Namespaces, SecurityDescriptor, AccessControlList und AccessControlEntry). Die drei letztgenannten Klassen sind extern instanziiierbar, wobei als ProgID nur der Klassenname und kein Komponentename verwendet wird.

B.2.6 Überblick über die ADSI-COM-Standardschnittstellen

ADSI-Schnittstellen

ADSI definiert neben den Meta-Schnittstellen eine Reihe weiterer konkreter Schnittstellen für viele übliche Anwendungsfälle in Verzeichnisdiensten wie die Benutzer- und Gruppenverwaltung, den Zugriff auf Computer, Drucker, Verzeichnisfreigaben, NT-Dienste und geöffnete Ressourcen. Darüber hinaus gibt es noch Schnittstellen für den Zugriff auf das Schema, auf komplexe Datentypen und providerspezifische Erweiterungen.

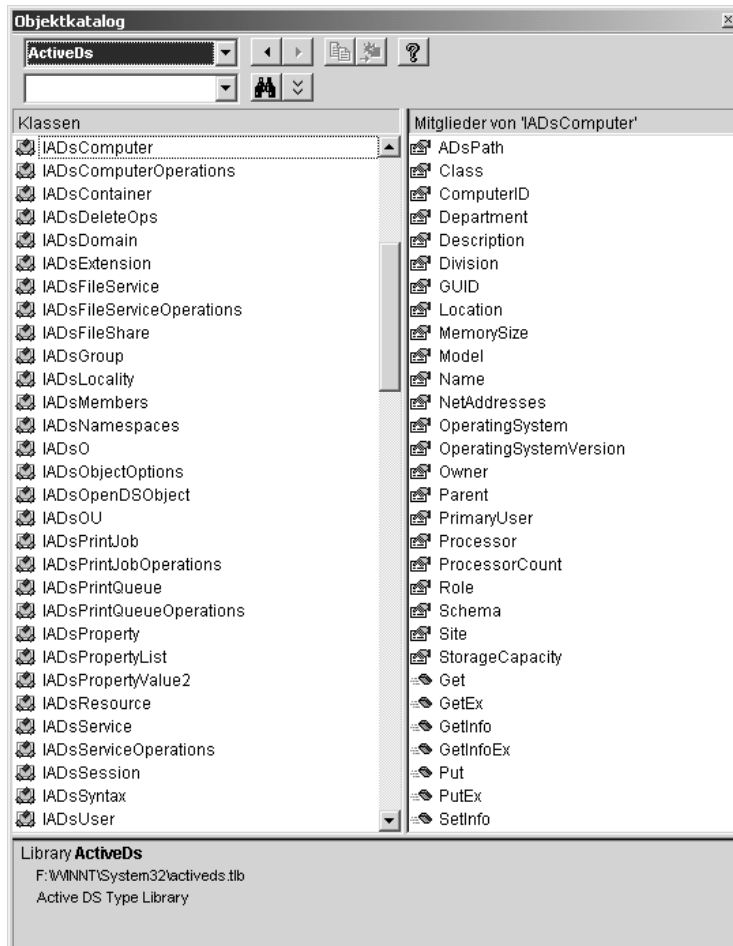


Abbildung B.8: Die wichtigsten ADSI-Standardschnittstellen im VB6-Objektkatalog

Die Standardschnittstellen lassen sich in fünf Gruppen einteilen:

- ▶ Schnittstellen, die Basisfunktionalitäten für ADSI bereitstellen und den Meta-Zugriff auf ADSI-Klassen ermöglichen
- ▶ Schnittstellen, die persistente Daten in Verzeichnisobjekten widerspiegeln. Zu einigen Verzeichnisobjekten gibt es dabei eine weitere Schnittstelle, die Aktionen auf diesen Verzeichnisobjekten bereitstellt.
- ▶ Schnittstellen, die keine persistenten Daten in Verzeichnisobjekten widerspiegeln, sondern dynamische Daten beinhalten und Funktionen auf Verzeichnisobjekten implementieren. In

**Schnittstellen-
typen**

der MSDN Library werden sie *Dynamic Object Interfaces* genannt. Objekte mit diesen Schnittstellen haben, da sie nicht persistent sind, keinen DN und können nicht direkt angesprochen werden.

- Schnittstellen zur Implementierung von Sicherheitsfunktionalitäten
- Schnittstellen zur Schemaverwaltung

Schnittstelle	Erläuterung
IADs	Diese Schnittstelle implementiert die absoluten Basisfunktionalitäten für ADSI-Objekte und muss bei der Implementierung des Providers daher in jede ADSI-Klasse (sowohl in Leaf- als auch in Container-Klassen) aufgenommen werden.
IADsContainer	Die Schnittstelle <i>IADsContainer</i> wird nur von Container-Klassen benutzt. <i>IADsContainer</i> realisiert Attribute und Methoden zur Verwaltung der enthaltenen Objekte.
IADsUser	Dient der Verwaltung von Benutzern
IADsMembers	Dient der Darstellung von Abhängigkeiten zwischen Verzeichnisobjekten (z.B. Benutzer <-> Gruppe)
IADsGroup	Dient der Verwaltung von Benutzergruppen
IADsCollection	Dient der Gruppierung von dynamischen Objekten zu Listen
IADsComputer	Zur Verwaltung einzelner Computer
IADsDomain	Zur Verwaltung einer Ansammlung von (Windows-)-Domänen
IADsService	Zur Verwaltung von Diensten auf einem bestimmten Computer
IADsFileservice	Spezieller Dateisystem-Dienst
IADsFileShare	Zur Verwaltung einer Verzeichnisfreigabe
IADsLocality	Dient der Verwaltung von Benutzern nach geographischen Gesichtspunkten
IADsO	Dient der Verwaltung von Organisationen
IADsOU	Dient der Verwaltung von Organisationseinheiten
IADsPrintQueue	Zur Verwaltung von Druckerwarteschlangen

Tabelle B.5: ADSI-Meta-Schnittstellen

Schnittstelle	Erläuterung
IADsPrintQueue Operations	Operationen auf Drucker-Warteschlangen: Statusinformationen, Anhalten eines Druckers, Weiterlaufenlassen, Löschen der Warteschlange
IADsComputer Operations	Abruf von Statusinformationen sowie das Herunterfahren und Neustarten von Computern
IADsService Operations	Operationen auf Diensten: Statusinformationen, Starten und Stoppen, Anhalten und Weiterlaufenlassen eines Dienstes
IADsFileservice Operations	Analog zu IADsServiceOperations, jedoch für den speziellen Fall eines Dateisystem-Dienstes; enthält Auflistungen der zugehörigen Sitzungen und Ressourcen
IADsPrintJob	Ein Druckauftrag in einer Druckerwarteschlange
IADsPrintJob Operations	Statusinformationen, Dauer, Seitenzahl, Position eines Druckauftrags, Anhalten und Weiterlaufenlassen eines Druckauftrags
IADsResource	Informationen über eine geöffnete Ressource auf einem Fileservice (z.B. Datei)
IADsSession	Informationen über eine geöffnete Sitzung (Session) auf einem Fileservice (z.B. verbundene Freigabe)
IADsOpenDSObject	Diese Schnittstelle dient der Impersonifizierung (d.h. dem Wechsel des Benutzerkontextes). IADsOpenDSObject stellt lediglich eine Methode zur Verfügung: OpenDSObject(). Bei OpenDSObject() können neben dem zu öffnenden Objekt auch ein Benutzername und ein Passwort eines Kontos angegeben werden, unter dem die Operationen auf dem zu öffnenden Objekt ausgeführt werden sollen.
IADsSecurity Descriptor	Informationen darüber, wer welche Zugriffsrechte auf ein Objekt hat; verwalten einen NT-Security Descriptor mit. Ein Security Descriptor besteht u.a. aus einer Access Control List (ACL).
IADsAccessControl Entry	Diese Schnittstelle dient der Verwaltung eines Access Control Entries (ACE) in einer Access Control List (ACL). Ein ACE legt fest, welche Rechte ein einzelner Benutzer oder eine einzelne Gruppe auf ein Objekt hat.
IADsAccessControl List	Liste aller Control Entries, die eine ACL bilden.
IADsClass	Repräsentiert eine Klassendefinition in einem Verzeichnisdienstschema

Tabelle B.5: ADSI-Meta-Schnittstellen (Fortsetzung)

Schnittstelle	Erläuterung
IADsProperty	Repräsentiert eine Attributdefinition in einem Verzeichnisdienstschema
IADsSyntax	Repräsentiert eine Syntaxbeschreibung, die einem Attribut zugeordnet werden kann
IADsPropertyList	Diese drei Schnittstellen sind die Möglichkeit einer ADSI-Klasse, Informationen über die unterstützten Attribute direkt über eine Instanz der Klasse bereitzustellen, ohne explizit auf das Verzeichnisschema zuzugreifen
IADsPropertyEntry	
IADsPropertyValue	

Tabelle B.5: ADSI-Meta-Schnittstellen (Fortsetzung)

Die Meta-Schnittstellen IADs und IADsContainer

IADs und IADsContainer

Zwei Schnittstellen bilden das Grundgerüst von ADSI: IADs und IADsContainer. Jede ADSI-Klasse muss IADs implementieren, um die Grundfunktionen für den Zugriff auf einen Verzeichniseintrag zur Verfügung zu stellen (siehe die nächsten beiden Tabellen). Alle Container-Objekte müssen zusätzlich IADsContainer implementieren (siehe dritte und vierte Tabelle).

Attribut	Erläuterung
Name	Name des ADSI-Objekts; entspricht dem Relative Distinguished Name (RDN)
Class	Name der Klasse, zu der das Verzeichnisdienstobjekt gehört; hier werden Namen von Backend-Klassen ausgegeben, nicht solche der Frontend-Klasse
GUID	Global Unique Identifier des Objekts
ADsPath	ADSI-Pfad des Verzeichnisobjekts
Parent	ADSI-Pfad des übergeordneten Container-Objekts
Schema	ADSI-Pfad zur Schemadefinition der Klasse, zu der das Objekt gehört

Tabelle B.6: Attribute der Standardschnittstelle IADs

Methode	Erläuterung
GetInfo()	Einlesen aller Attribute eines Verzeichnisobjekts in den Cache
GetInfoEx()	Einlesen einzelner Attribute eines Verzeichnisobjekts in den Cache

Tabelle B.7: Methoden der Standardschnittstelle IADs

Methode	Erläuterung
SetInfo()	Schreiben des Caches in das Verzeichnisobjekt
Get()	Auslesen eines Attributs aus dem Cache; nur mehrwertige Attribute werden als <i>Array of Variant</i> zurückgegeben
Put()	Schreiben eines einwertigen Attributs in den Cache
GetEx()	Auslesen eines Attributs aus dem Cache. Im Gegensatz zu <i>Get()</i> werden alle Attribute als <i>Array of Variant</i> zurückgegeben.
PutEx()	Verändern eines mehrwertigen Attributs im Cache

Tabelle B.7: Methoden der Standardschnittstelle IADs (Fortsetzung)

Attribut	Erläuterung
Count	Anzahl der untergeordneten Objekte
Filter	Möglichkeit, einen Filter auf einem Container zu definieren und so die Ergebnismenge auf die Instanzen bestimmter Klassen einzuschränken. <i>Filter</i> erwartet als Parameter ein <i>Array of String</i> mit den Klassennamen.

Tabelle B.8: Attribute der Standardschnittstelle IADsContainer

Methode	Erläuterung
GetObject()	Zugriff auf Zeiger eines untergeordneten Objekts anhand eines relativen Namens. Als erster Parameter muss der Name der Backend-Klasse übergeben werden, aus der das Objekt stammt, da einige Verzeichnisdienste nicht die Eindeutigkeit des RDNs verlangen.
Create()	Erzeugen eines untergeordneten Verzeichnisobjekts
Delete()	Löschen eines untergeordneten Verzeichnisobjekts
CopyHere()	Kopieren eines Verzeichnisobjekts in diesen Container
MoveHere()	Verschieben eines Verzeichnisobjekts in diesen Container

Tabelle B.9: Methoden der Standardschnittstelle IADsContainer

Einschränkungen mit *CopyHere()* und *MoveHere()*

CopyHere() und *MoveHere()* können nur innerhalb eines Verzeichnisses verwendet werden. Objekte können nicht von einem Verzeichnisdienst zu einem anderen kopiert oder verschoben werden. Nicht alle Verzeichnisdienste unterstützen diese Operationen. Zum Beispiel kann der Exchange Server 5.5 keine Empfänger zwi-

schen Containern verschieben. Dies kann aber das Active Directory und damit auch Exchange Server 2000. CopyHere() wird derzeit nur durch den Novell Directory Service (NDS) unterstützt.

B.3 ADSI-Werkzeuge

Dieses Kapitel stellt nützliche Werkzeuge für das Active Directory Service Interface (ADSI) vor.

B.3.1 Active Directory Service Browser (ADB)

Ein Browser für ADSI

Um die Objekthierarchie eines Verzeichnisdienstes zu erkunden, ist der *Microsoft Active Directory Service Browser (ADB)* ein zweckmäßiges Werkzeug (siehe nächste Abbildungen). Der ADB ist in den Installationspaketen zu ADSI enthalten und hat den Dateinamen *adswv.exe* (Größe 242 KB). Der ADB sollte nicht verwechselt werden mit einer anderen im Umlauf befindlichen Programmdatei namens *Active Directory VB Browser (dbbrowse.exe, 65 KB)*. Dabei handelt es sich um die kompilierte Version eines Microsoft-ADSI-Programmierbeispiels, die nicht die volle Funktionalität des ADBs umfasst.

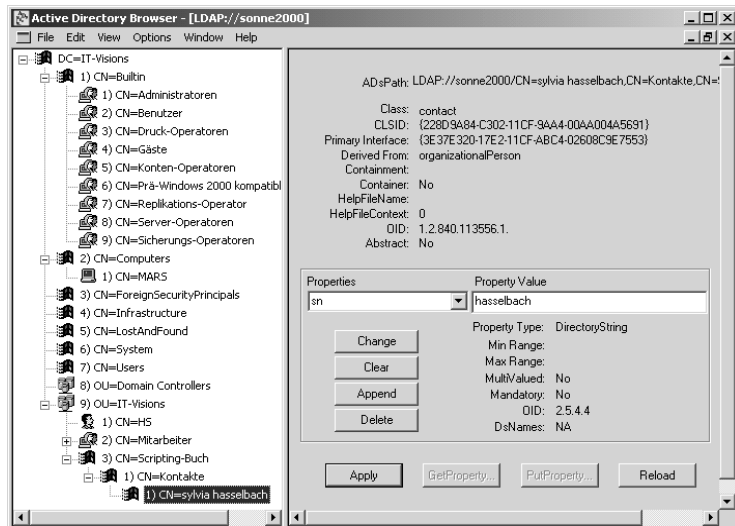


Abbildung B.9: Ansicht eines Kontakt-Eintrags in einem Active Directory. Verzichten muss man beim ADB auf schöne Icons. Dafür hat man einen mächtigen Thin-Client für alle Verzeichnisdienste.

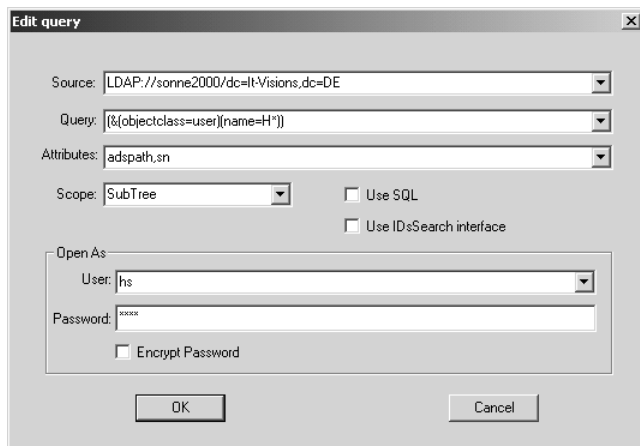


Abbildung B.10: Eingabe einer LDAP-Query

Der ADB in der Version 1.0.0.54 hat das Problem, dass er in seiner Baumdarstellung nicht mehr als 1006 Einträge darstellen kann. Das Windows 2000 Active Directory hat jedoch in seinem Schema mehr Einträge, so dass die Liste bei der Darstellung des Schemas abgeschnitten wird.



Bug

Der ADB ermöglicht sowohl das Browsing durch einen Verzeichnisbaum als auch die Ausführung von ADSI-Queries. Wählen Sie dazu FILE/NEW/QUERY.

B.3.2 ADSI Explorer

Das Betrachten eines beliebigen Verzeichnisdienstes ermöglicht auch das Freeware-Werkzeug *ADSI Explorer*. Anders als beim Active Directory Browser können Objektattribute nur gelesen, nicht aber geändert werden. Auch das Hinzufügen und Löschen von Verzeichnisobjekten ist nicht möglich.

Der *ADSI Explorer* ist ein kostenloses Werkzeug der Firma InTouchSoftware [ITS02]. Auf der Buch-CD finden Sie es unter [CD:/install/Werkzeuge/ADSI/ADSI Explorer].

Verzeichnisdienste erforschen

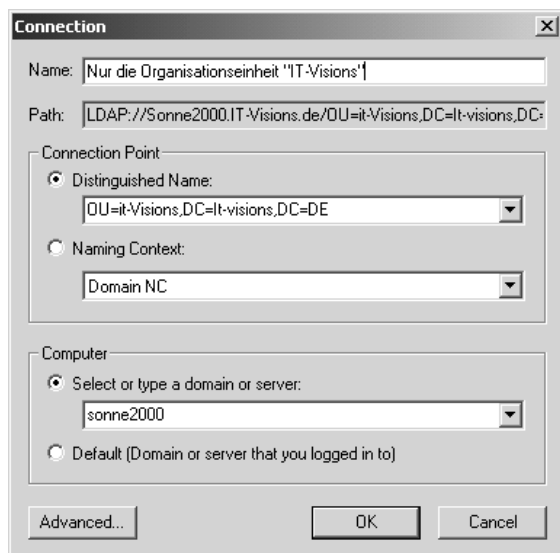


Abbildung B.13: Flexibles Mapping in ADSI Edit

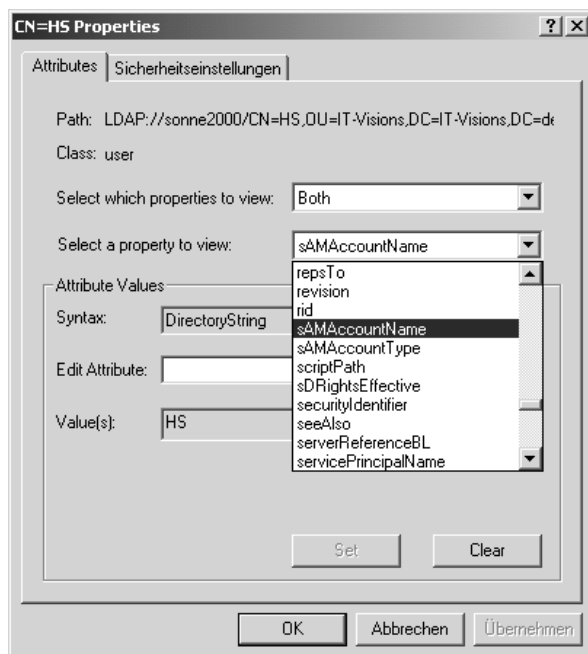


Abbildung B.14: Standardeigenschaftsfenster in ADSI Edit

ADSI Edit bietet zwei große Vorteile gegenüber den Standard-AD-Snap-Ins:

- ▶ Es kann jeder beliebige Container in den Baum gemappt werden.
- ▶ Da nur ein Standardeigenschaftsfenster verwendet wird, können auch durch eine Schemamodifikation entstandene Klassen und Attribute betrachtet und modifiziert werden. In den Standard-Snap-Ins wird dazu stets die Registrierung eines DisplaySpecifier benötigt.

B.3.4 ADSISchemaBrowser

Der *ADSI Schema Browser* (*SchemaBrowser.exe*) ist ein einfaches Tool zur Betrachtung von Schema und Objekthierarchie zur Laufzeit. Es ist in VB6 geschrieben und liegt der Buch-CD in kompilierter Form und im Quellcode bei [CD:/install/Werkzeuge/adsi/schemabrowser/].

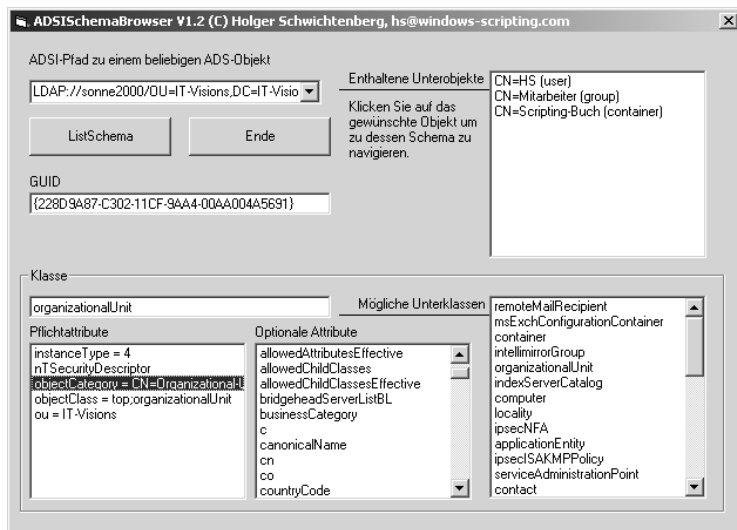


Abbildung B.15: ADSISchemaBrowser

C Windows Management Instrumentation (WMI)

Dieser Anhang enthält eine kurze Einführung in die mächtige Windows Management Instrumentation (WMI), die in .NET durch den BCL-Namespace `System.Management` (siehe Kapitel 22) verwendet werden kann.

WMI ist die Microsoft-Implementierung des Web Based Enterprise Managements (WBEM). WBEM ist ein Standard der Desktop Management Task Force (DMTF) für das Netz- und Systemmanagement, also zur Verwaltung von Netzwerk- und Systemressourcen (z.B. Hardware, Software, Benutzer). WBEM wurde ursprünglich von BMC Software, Cisco Systems, Compaq, Intel und Microsoft entwickelt und später an die DMTF übergeben. Aus historischen Gründen findet man in WMI-Tools häufig noch die Bezeichnung WBEM.

WBEM versus WMI

Kern von WBEM ist das *Common Information Model (CIM)*, das die durch WBEM zu verwaltenden Ressourcen durch objektorientierte Methoden modelliert. CIM ist ein Framework zur Beschreibung sowohl physischer als auch logischer Objekte. Die DMTF versteht CIM als eine Vereinigung bestehender Management-Architekturen wie dem OSI Management Framework X.700 (Common Management Information Protocol – CMIP) und dem Simple Network Management Protocol (zu CMIP und SNMP siehe [STA93]).

CIM

WBEM ist ein konsequent objektorientierter Einsatz. Alle Ressourcen werden durch Objekte repräsentiert, die in Klassen zusammengefasst sind. Ebenso wie bei dem ISO Management Framework nennt man auch in WBEM die Repräsentation einer (Hardware- oder Software-)Ressource ein *Managed Object (MO)*.

Managed Objects



Der Name Web Based Enterprise Management ist irreführend, weil er nahe legt, dass es sich bei WBEM um eine grafische Benutzerschnittstelle auf Webbasis für das Management von Systeminformationen handelt. WBEM ist jedoch lediglich eine Architektur mit Programmierschnittstelle, also weder Tool noch Anwendung.

Weitere
Informationen

Weitere Informationen zu WBEM/WMI finden Sie in folgenden Büchern:

Holger Schwichtenberg
COM-Komponenten-Handbuch: Systemprogrammierung und Scripting mit COM-Komponenten
ISBN 3-8273-1936-6
erschieden im Jahr 2001 bei Addison-Wesley

Holger Schwichtenberg
Windows Scripting: Automatisierte Systemadministration mit Visual Basic/VBScript und COM/.NET
2. Auflage
ISBN 3-8273-1843-2
erschieden im Jahr 2001 bei Addison-Wesley

C.I Überblick über die Managed Objects

Features

Die Informationsfülle, die WMI bereits heute liefert, ist riesig. Beispiele für Bereiche, aus denen WMI verschiedene Managed Objects bereitstellt, zeigen die folgenden Tabellen.

Installierte Software	Laufende Prozesse
BIOS	Programmgruppen im Startmenü
Boot-Konfiguration	NT-Dienste (Win32_SystemServices)
Installierte Hardware (z. B. Netzwerkkarten, Grafikkarten) einschließlich Treiber und deren Zuordnung zu LoadOrderGroups, belegte Ressourcen (IRQ, Port, DMA), Konfiguration (z. B. Druckereinstellungen)	NT-Ereignisprotokoll
Ordner und Dateien des Dateisystems	Eingerichtete Zeitzonen
	Benutzerkonten (inkl. deren Gruppenzuordnung, Desktop-Einstellungen und Ereignisprotokolleinträge)
	Registry

Tabelle C.1: Überblick über WMI-Informationen (WMI 1.5)

Netzlaufwerksverbindungen	Dateisicherheit, Freigabesicherheit
Installiertes Betriebssystem (z. B. Betriebssystemname, Build-Version, Installationsdatum, Datum und Uhrzeit des letzten Boot-Vorgangs)	Active Directory
Installierte Updates und Hotfixes	Installierte COM-Komponenten einschließlich Zuordnung zu Komponentenkategorien und DCOM-Einstellungen
Umgebungsvariablen	Druckaufträge
Performance Monitor-Daten	Geplante Vorgänge (Taskscheduler)
SNMP-Daten	ODBC-Einstellungen
	Auslagerungsdateien

Tabelle C.1: Überblick über WMI-Informationen (WMI 1.5) (Fortsetzung)

Distributed File System (DFS)	Terminal Services
IP-Routing	Datum und Uhrzeit
Disk Quotas	Active Directory-Replikation
Ausführung eines Ping	Clustering
Ausführung von CHKDSK	DNS Server
Internet Information Services (IIS)	Network Load Balancing (NLB)
Netzwerkverbindungen und Sitzungen	Security Configuration Editor (SCE)
Drucker und Druckerwarteschlangen	Prozessüberwachung

Tabelle C.2: Zusätzliche Möglichkeiten in WMI 5.1 (Windows XP/.NET)

Welche Managed Object es konkret gibt, können Sie mit dem WMI Object Browser (Kapitel 1.6.2), dem WMI CIM Studio (Kapitel 1.6.3) und dem WMI-Add-in für den Visual Studio .NET Server Explorer (Kapitel 22.5) erforschen.

C.2 Installation und Konfiguration

WMI ist in Windows 2000, Windows ME, Windows XP und Windows .NET bereits integriert. Für die anderen Windows-Versionen gibt es ein kostenloses Add-on. Im Gegensatz zum Active Directory Service Interface (ADSI) muss WMI bei entfernten Operationen sowohl auf dem Client als auch auf dem Server installiert sein.

Ob WMI installiert ist oder nicht, erkennt man daran, ob der Registry-Schlüssel `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\WBEM` existiert oder nicht.

Die WMI-Version in Windows XP und Windows .NET ist gegenüber dem WMI in Windows 2000 und Windows ME erweitert. Zum Zeitpunkt des Redaktionsschlusses dieses Buchs bot Microsoft noch kein Update für Windows 2000 und Windows ME an.

WMI hat zwischen 2000 und XP einen Versionssprung gemacht. Bei Windows 2000 wurde Version 1.50.1085 mitgeliefert. Bei XP heißt die Version 5.1.2600; sie wurde damit an die Betriebssystemversion angeglichen.

WMI Software Development Kit

SDK Microsoft bietet zu WMI ein Software Development Kit (SDK) an, das Dokumentation, Beispielcode, C++-Include-Dateien und einen Teil der in Kapitel 1.6 beschriebenen Werkzeuge enthält. Das SDK ist auf der Buch-CD enthalten [CD:/install/SDK/WMI SDK].

C.3 Architektur

WinMgmt.exe WMI wird durch die ausführbare Datei *WinMgmt.exe* implementiert. *WinMgmt.exe* läuft unter der NT-Produktfamilie als Dienst unter dem Namen »WinMgmt (Windows-Verwaltungsinstrumentation)«. Auf Windows 95/98/ME wird *WinMgmt.exe* beim ersten WMI-Aufruf als normaler Prozess gestartet. Der manuelle Start von *WinMgmt.exe* ist auch in der NT-Produktfamilie möglich.

WMI wird unterhalb des Systemverzeichnisses in einem Unterverzeichnis mit dem Namen *\WBEM* installiert. Dort finden sich in dem Unterverzeichnis *\Log* zahlreiche Logfiles. Die WMI-Registry-Einstellungen finden Sie unter *HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\WBEM*.

Provider

WMI-Provider WMI basiert ebenso wie das Active Directory Service Interface (ADSI) und OLE DB auf einem Provider-Konzept, das eine Erweiterbarkeit ermöglicht. Provider sind COM-Komponenten (DLL oder EXE), die beim Aufruf einer Ressource, die durch den Provider bereitgestellt wird, in den Speicher geladen werden. Die WMI-Provider spielen aber im Vergleich zu den ADSI-Providern eine untergeordnete Rolle bei der praktischen Anwendung von WMI,

da alle Informationen in ein globales Schema aufgenommen werden, so dass es beim konkreten Zugriff auf eine Klasse kaum Unterschiede zwischen den Providern gibt.

C.4 Programmierung

Sowohl unter COM als auch unter .NET ist die Programmierschnittstelle von WMI als Meta-Objektmodell realisiert: Eine kleine Menge von Meta-Klassen dient dazu, eine große Menge von Managed Objects (WMI-Klassen und -Objekte) zu verwalten. Dabei wird ein Meta-Objekt (unter .NET z.B. eine Instanz der Klasse `System.Management.ManagementObject`) an ein Managed Object gebunden und bietet dann Zugriff auf Attribute und Methoden des gebundenen Objekts. Das Meta-Objektmodell ermöglicht die einfache Erweiterbarkeit von WMI.

Meta-Objektmodell

Die Implementierung des Meta-Objektmodells ist in COM und .NET sehr verschieden.

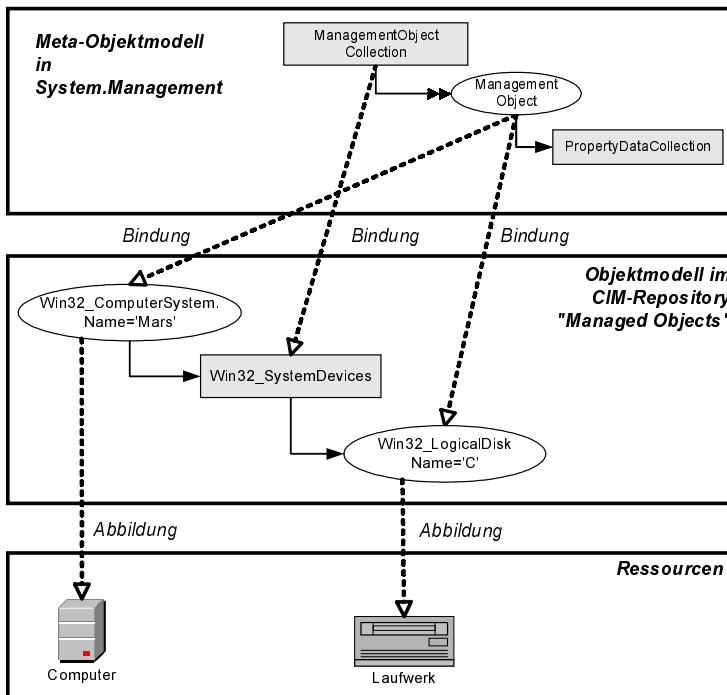


Abbildung C.1: WMI-Programmierung via Meta-Objektmodell

Meta-Klasse für Zugriff auf...	COM-Komponente (wbemdisp.dll)	.NET-Komponente (System.Management.dll)
WMI-Objekt	WbemScripting.SWbemObject	System.Management.ManagementObject
WMI-Klasse	WbemScripting.SWbemObject	System.Management.ManagementClass
Menge von WMI-Objekten oder -Klassen	WbemScripting.SWbemObjectSet	System.Management.ManagementObjectCollection

Tabelle C.3: Die wichtigsten Meta-Klassen

Programmierung

WMI unterstützt sowohl den Zugriff auf einzelne Managed Objects als auch den Zugriff auf Objektmengen. Weiterhin unterstützt WMI folgende Vorgehensweisen:

- ▶ Suchen von Managed Objects durch Abfragen im SQL-Stil (WMI Query Language)
- ▶ transparentes Schema zur Erforschung der Managed Objects per Programmcode
- ▶ Möglichkeit zur Definition eigener Ereignisse auf jede beliebige Veränderung in einem Managed Object
- ▶ asynchrone Befehlsausführungen, die ihre Ergebnisse durch Ereignisse zurückliefern

WMI 5.1

Neue Funktionen

Windows XP und Windows .NET enthalten eine erweiterte Version von WMI. Neu sind folgende Funktionen:

1. Event Correlation (Zusammenfassung von Ereignissen)
2. Garantierte Zustellung von Ereignissen durch optionale Integration mit dem Microsoft Message Queue Service (MSMQ)
3. Zahlreiche neue Managed Objects
4. Erweiterungen im WMI-Meta-Objektmodell

C.5 Konzepte

In diesem Unterkapitel erhalten Sie eine Einführung in die wichtigsten Konzepte von WMI.



Auch der Aufbau des Schemas ist Teil des Schemas. Dieser Teil des Schemas wird als Metamodell bezeichnet

C.5.2 Namespaces

Namensräume

Ein WMI-Namespace ist ein Instrument zur Gruppierung von WMI-Klassen und deren Instanzen in logische Einheiten. Ein Namespace ist der Startpunkt sowohl für eine Vererbungshierarchie von WMI-Klassen als auch für eine Containment-Hierarchie von WMI-Objekten. Jeder Namespace hat also sein eigenes Schema. Klassennamen in zwei Namespaces dürfen theoretisch gleich sein, sollten es aber nicht, da in zukünftigen WMI-Versionen geplant ist, Namespace-übergreifende Operationen zu ermöglichen. In WMI 1.5 sind keine Objektassoziationen zwischen verschiedenen Namespaces möglich. Um gleiche Klassennamen zu vermeiden, gibt es die Konvention, dass dem Klassennamen stets der Namespace-Name vorangestellt werden soll. Ein Namespace ist selbst eine Klasse, die direkt oder indirekt von der Systemklasse `__Namespace` geerbt hat.



Namespaces können hierarchisch angeordnet werden, so dass eine Namespace-Hierarchie entsteht. Diese Hierarchie dient aber nur der Übersichtlichkeit; sie impliziert keine Vererbung von Klassen. Ein bestimmter Namespace wird über einen hierarchischen Pfad der Form `RootNamespace\Namespace1\Namespace2\... usw.` angesprochen.

Namespace versus
Provider

Ein Namespace ist unabhängig von einem WMI-Provider: Ein Provider kann mehrere Namespaces realisieren, ebenso wie ein Namespace Klassen aus mehreren Providern enthalten kann.

WMI-Namespace	Anzahl Klassen	Davon Systemklassen (Name beginnt mit doppeltem Unterstrich)
root\SECURITY	56	56
root\RSOP	53	51
root\Cli	61	51
root\WMI	654	51
root\CIMV2	919	51
root\MSAPPS10	171	51

Tabelle C.4: Anzahl der Klassen in WMI 5.1 (Windows XP mit installiertem Visual Studio .NET und Office XP Professional)

WMI-Namespace	Anzahl Klassen	Davon Systemklassen (Name beginnt mit doppeltem Unterstrich)
root\Policy	55	51
root\Microsoft	51	51
root\DEFAULT	61	53
root\directory	51	51
root\subscription	85	51
root\NetFrameworkv1	583	51
SUMME	2800	619

Tabelle C.4: Anzahl der Klassen in WMI 5.1 (Windows XP mit installiertem Visual Studio .NET und Office XP Professional) (Fortsetzung)

C.5.3 Lokalisierung

WMI erlaubt Lokalisierung (d.h. landesspezifische Anpassung) von Schemainformationen und die Speicherung mehrerer lokalisierter Versionen eines Namespaces innerhalb des WMI-Repository. WMI speichert dazu die sprachneutralen Teile der Klassendefinition getrennt von den landesspezifischen Teilen. Landesspezifische Informationen sind insbesondere die Hilfe-Informationen zu den Klassen und Eigenschaften.

Die landesspezifischen Informationen werden in Unter-Namespaces gespeichert. Jedes Land hat eine *LocaleID*.

LocaleID

1. *ms_407* steht für Deutsch.
2. *ms_409* steht für amerikanisches Englisch.

Der Namespace `\root\CIMV2\ms_407` ist also der `\root\CIMV2`-Namespace mit deutschen Hilfe-Informationen, `\root\CIMV2\ms_409` der gleiche mit englischen Texten.

Beim Zugriff auf den übergeordneten Namespace `root\CIMV2\` ist die Sprache abhängig von Ihren Computereinstellungen.



In einigen Tools werden diese Unter-Namespaces durch eine Weltkugel angezeigt, in anderen Tools gibt es keinen Unterschied zu den normalen Namespaces.

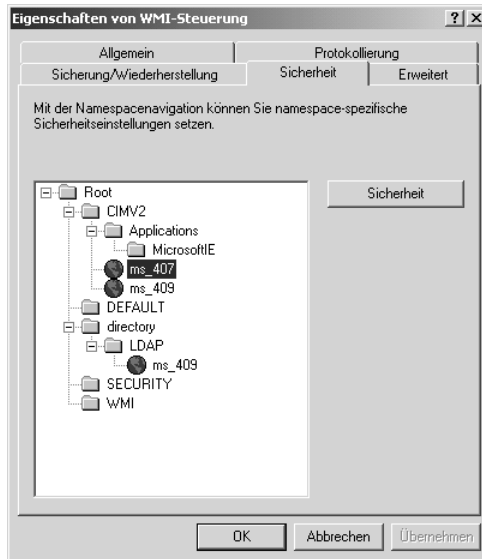


Abbildung C.3: Anzeige der Namespace-Hierarchie im WMI-Snap-In in der MMC (Eigenschaftsfenster des Eintrags WMI-Steuerung)

C.5.4 Klassen und Objekte

Klassen *CIM-Klassen* sind eine sehr allgemeine, betriebssystemunabhängige Beschreibung von Ressourcen. *WMI-Klassen* sind eine konkrete, in der Windows-Umgebung implementierte Repräsentation von Ressourcen. Die meisten WMI-Klassen sind von CIM-Klassen abgeleitet und erweitern den Standard. Einige Ressourcen in einem Windows-System können auch direkt durch CIM-Klassen abgebildet werden.

Namenskonvention Es ist eine Konvention, dass ein kompletter WMI-Klassenname aus dem Namespace-Namen und dem eigentlichen Klassennamen besteht, getrennt durch einen Unterstrich.

NamespaceName_ClassName

Statische Methoden WMI-Klassen können so genannte statische Methoden implementieren, die direkt auf einer Klasse ausgeführt werden können, ohne dass eine Instanz der Klasse benötigt würde. Statische Methoden sind z.B. Konstruktormethoden wie die Methode `Create()` auf der Klasse `Win32_Process`. Es gibt auch abstrakte Klassen in WMI, von denen keine Instanzen erzeugt werden können.

Schlüsselattribute

Schlüsselattribute (*Keys*) sind Attribute, die der eindeutigen Identifizierung einer Instanz innerhalb einer Klasse dienen. Ein Key entspricht dem Primärschlüssel einer relationalen Tabelle. Ebenso wie ein Primärschlüssel aus mehreren Spalten einer Datenbanktabelle bestehen kann, kann auch ein Key in WMI aus mehreren Attributen bestehen. Einen Schlüsselwert darf es innerhalb aller Instanzen einer Klasse nur einmal geben. Wenn der Key aus mehreren Attributen besteht, müssen nur alle Attributwerte zusammen eindeutig sein. Welche Attribute Schlüsselattribute sind, wird in der Klassendefinition festgelegt, damit alle Instanzen einer Klasse die gleichen Schlüsselattribute besitzen.

Keys

Qualifier

Qualifier sind Informationen, die eine Klasse, ein Objekt, ein Attribut, eine Methode oder einen Parameter beschreiben. Qualifier dienen im derzeitigen WMI nur der Informationsversorgung des Nutzers. Sie ermöglichen keine verbindlichen Einstellungen für die WMI-Provider (Beispiel: ein Attribut, das einen Qualifier `read-only` besitzt, muss nicht notwendigerweise wirklich schreibgeschützt sein). Durch den Qualifier `Key` wird festgelegt, ob ein Attribut ein Schlüsselattribut ist.

Qualifier

Objektassoziationen

Instanzen von Managed Objects können durch Assoziationen miteinander verbunden sein. Ein Beispiel für eine Assoziation ist `CIM_DirectoryContainsFile`. Diese Klasse ist eine Assoziation zwischen `CIM_Directory` und `CIM_DataFile`.

Assoziationen

Systemklassen

WMI definiert eine Reihe von Systemklassen, die der Verwaltung von WMI selbst und insbesondere dem Ereignisdienst dienen. Die Systemklassen sind in jedem Provider implementiert; sie sind daran erkennbar, dass der Name mit einem doppelten Unterstrich beginnt. Beispiele für Systemklassen sind:

Systemklassen

- ▶ `__EventConsumer`
- ▶ `__Namespace`
- ▶ `__Event`
- ▶ `__InstanceDeletionEvent`

Datentypen

Datentypen CIM definiert 16 Standarddatentypen.

Symbolische Konstante	Wert
wbemCimtypeSint8	16
wbemCimtypeUint8	17
wbemCimtypeSint16	2
wbemCimtypeUint16	18
wbemCimtypeSint32	3
wbemCimtypeUint32	19
wbemCimtypeSint64	20
wbemCimtypeUint64	21
wbemCimtypeReal32	4
wbemCimtypeReal64	5
wbemCimtypeBoolean	11
wbemCimtypeString	8
wbemCimtypeDatetime	101
wbemCimtypeReference	102
wbemCimtypeChar16	103
wbemCimtypeObject	13

Tabelle C.5: CIM-Standarddatentypen (definiert in der Konstantenliste WbemCimtype Enum in wbemdisp.tlb)

Datum und
Uhrzeit

Datum und Uhrzeit werden als String der Form `yyyymmddHHMMSS.mmmmmmsUUU` gespeichert, wobei neben dem selbsterklärenden Kürzel anzumerken ist, dass `mmmmmm` die Anzahl der Millisekunden ist und `UUU` die Anzahl der Minuten, die die lokale Zeit von der *Universal Coordinated Time* (UTC) abweicht. Das `s` ist das Vorzeichen. In Deutschland steht daher für `UUU` der Wert `+060`.

WMI kennt ein eigenes Format für Zeitintervalle: `ddddddd dHHMMSS.mmmmm:000`. Auch ein Zeitintervall wird als String abgelegt. Dabei repräsentiert `ddddddd` die Anzahl der Tage. Der String endet immer auf `:000`.

Systemattribute

Systemattribute

Alle WMI-Klassen und alle Instanzen besitzen eine Reihe von Systemeigenschaften. Sie beginnen mit einem doppelten Unterstrich und können über die WMI-Komponente nicht direkt abgefragt

werden. Die wichtigsten dieser Eigenschaften werden aber durch das Unterobjekt `Path_` der `SWbemObject`-Klasse bereitgestellt.

Attribut	Erläuterung
<code>__Class</code>	Name der Klasse. Dieser Wert kann für Klassen geändert werden.
<code>__Derivation</code>	Ein Array of String, das die Vererbungshierarchie wiedergibt. Der erste Eintrag ist die direkte Oberklasse.
<code>__Dynasty</code>	Name der obersten Klasse der Vererbungshierarchie. Bei der obersten Klasse steht hier kein Leerstring, sondern der gleiche String wie bei <code>__Class</code> .
<code>__Genus</code>	1 = <code>SWbemObject</code> ist eine Klasse. 2 = <code>SWbemObject</code> ist eine Instanz.
<code>__Namespace</code>	Namespace, in dem die Klasse oder die Instanz existiert
<code>__Path</code>	Vollständiger WMI-Pfad einschließlich Server und Namespace
<code>__Property_Count</code>	Anzahl der Attribute der Klasse. Dabei werden diese Systemattribute nicht mitgezählt.
<code>__Relpath</code>	WMI-Pfad ohne Server und Namespace
<code>__Server</code>	Name des Servers
<code>__Superclass</code>	Name der direkten Oberklasse

Tabelle C.6: WMI-Systemeigenschaften

C.5.5 Managed Object Format (MOF)

Das *Managed Object Format (MOF)* ist eine Sprache zur Definition von Managed Objects. MOF basiert auf der *Interface Definition Language (IDL)* und ist ein Textformat. MOF-Dateien können mit Hilfe des MOF-Compilers (*mofcomp.exe*) in das CIM-Repository übernommen werden.

Das nachfolgende Listing zeigt Ausschnitte aus der Datei *msioff9.mof*, die die MOF-Beschreibung für Informationen über Microsoft Office liefert. Die dort definierten Klassen entsprechen den im MS Info anzeigbaren Daten. Das MOF-File definiert zunächst einen neuen Namespace *MSAPPS* und dann über eine CLSID den Provider, der die Funktionalität der im Folgenden definierten Klassen implementiert.

**MO-Beschreibung
durch MOF**

**MOF für Micro-
soft Office**

```

/*****
/** File: MSIOff9.mof - Office Extension MOF File for MSInfo
5.0
*****/
/****Creates namespace for MSAPPS
#pragma namespace ("\\\\".\\Root")
instance of __Namespace
{
    Name = "MSAPPS";
};
/** Declare an instance of the __Win32Provider so as to
"register" the
/** Office provider.
instance of __Win32Provider as $P
{
    Name = "OffProv";
    CLSID = "{D2BD7935-05FC-11D2-9059-00C04FD7A1BD}";
};

/** Class: Win32_WordDocument
/** Derived from:
[dynamic: ToInstance, provider("OffProv")]
class Win32_WordDocument
{
    [key, read: ToInstance ToSubClass] string Name;
    [read: ToInstance ToSubClass] string Path;
    [read: ToInstance ToSubClass] real32 Size;
    [read: ToInstance ToSubClass] datetime CreateDate;
};
/** Class: Win32_AccessDatabase
/** Derived from:
[dynamic: ToInstance, provider("OffProv"), Singleton:
DisableOverride ToInstance ToSubClass]
class Win32_AccessDatabase
{
    [read: ToInstance ToSubClass] string Name;
    [read: ToInstance ToSubClass] string Path;
    [read: ToInstance ToSubClass] real32 Size;
    [read: ToInstance ToSubClass] datetime CreateDate;
    [read: ToInstance ToSubClass] string User;
    [read: ToInstance ToSubClass] string JetVersion;
};

```

Listing C.1: Ein kleiner Ausschnitt aus dem MOF-File MSIOff9.mof

Der WMI-Provider für Microsoft Office 2000 gehört zu den WMI-Providern (implementiert durch `offprov.exe`), die nicht automatisch in WMI eingebunden werden. Sie können dies jedoch selbst vornehmen, indem Sie das mit Office 2000 mitgelieferte MOF-File `msioff9.mof` kompilieren:

mofcomp.exe MSIOFF9.mof

Danach stehen Ihnen eine Reihe interessanter Informationen über MS Office zur Verfügung.

C.5.6 Sicherheitsfunktionen

WMI basiert auf COM und verwendet die COM-Sicherheitsfunktionen und die entsprechend verfügbaren Security Provider. Sicherheitseinstellungen können auf der Ebene eines jeden Namespaces festgelegt werden. Diese Einstellung erfolgt im WMI-Snap-In in der MMC. Ein COM-Client, der auf ein WMI-Objekt zugreifen will, wird zunächst gegen die Sicherheitseinstellung des Namespaces geprüft, zu dem das Objekt gehört. Die Vergabe von Zugriffsrechten auf Objekt- oder Klassenebene unterstützt WMI 1.5 nicht.

Sicherheit

WMI unterstützt *Impersonifizierung* für den Zugriff auf entfernte Rechner. Es ist also möglich, beim Aufruf von WMI-Objekten auf einem entfernten Rechnersystem den Benutzerkontext zu wechseln und als ein anderer Benutzer aufzutreten als der, unter dem der COM-Client läuft. Dies ist allerdings beim Zugriff auf das lokale WMI nicht möglich. Ein Versuch, den Benutzerkontext vor dem Zugriff auf lokale WMI-Objekte zu wechseln, wird von WMI mit dem Fehler 80041064 quittiert: »Benutzeranmeldeinformationen können für lokale Verbindungen nicht verwendet werden«.

Impersonifizierung

Im Gegensatz zu anderen Komponenten erlaubt WMI Vorgaben für die COM-Sicherheit durch den Client. Sowohl Impersonifizierungs- als auch Authentifizierungsmodus können beim Verbindungsaufbau eingestellt werden. Der Client kann unter Windows 2000 auch den Security Service Provider (SSP) zwischen der NT4-Lanmanager (NTLM)- und der Kerberos-Authentifizierung wählen. Wird kein SSPI explizit angegeben, verhandelt WMI den SSPI beim Verbindungsaufbau. Es wird zunächst versucht, Kerberos zu verwenden. Kerberos kann allerdings nie für lokale Aufrufe verwendet werden. Für lokale Aufrufe kann eine Vielzahl von Einzelrechten, so genannte *Privileges*, gesetzt werden.

Programmatische Sicherheit

C.5.7 Ereignisse

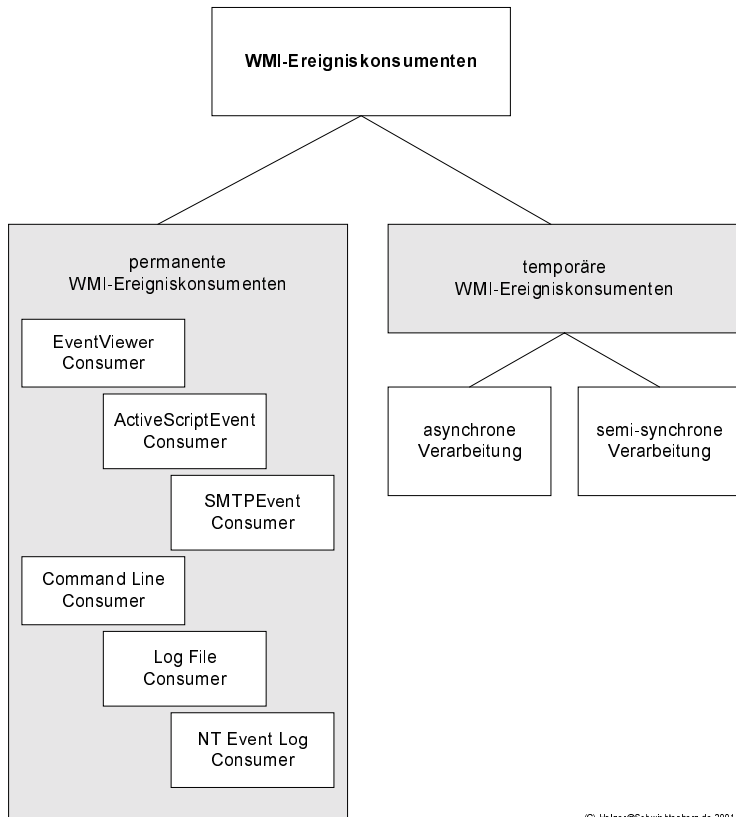
WMI bietet ein komplexes System für Ereignisse in Managed Objects. Dabei registrieren sich so genannte *Ereigniskonsumenten* (*Event Consumer*) bei WMI für bestimmte Ereignisse. Der Ereignis-

Konsumenten

**Permanente
versus temporäre
Konsumenten**

konsument führt beim Eintritt eines Ereignisses eine bestimmte Aktion aus.

WMI unterscheidet zwei Arten von Ereigniskonsumenten: *temporäre* und *permanente Ereigniskonsumenten*. Der Unterschied zwischen den beiden Typen ist, dass ein temporärer Ereigniskonsument nur Ereignisbenachrichtigungen erhält, wenn er aktiv ist. Ein temporärer Ereigniskonsument wird durch ein Script oder ein Programm implementiert. Nach Beendigung des Scripts/Programmes ist der Konsument nicht mehr vorhanden. Dagegen ist ein permanenter Konsument in Form eines Managed Objects im Repository gespeichert und kann zu jedem Zeitpunkt Ereignisbenachrichtigungen empfangen, da WMI den Consumer bei Bedarf selbst startet und dann das Ereignis übermittelt.



(C) Holger@Schwiebenberg.de 2001

Abbildung C.4: Überblick über die Ereigniskonsumenten

Permanente Ereigniskonsumenten sind z. B.:

1. der *EventViewerConsumer* (im Namespace */root/cimv2*), der die aufgetretenen Ereignisse in einem Bildschirmfenster, dem WMI Event Viewer, darstellt. Der Event Viewer wird unter den Werkzeugen in Kapitel 1.6 vorgestellt. **EventViewer-Consumer**
2. der *ActiveScriptEventConsumer* (im Namespace */root/default*), der beim Auftreten eines Ereignisses ein Active Script ausführt **ActiveScriptEvent-Consumer**
SMTPEventConsumer
3. Mit Hilfe des *SMTPEventConsumers* können bei Ereignissen E-Mails über den Microsoft SMTP-Server (der Bestandteil des Internet Information Servers ab Version 4.0 ist) versendet werden. Der *SMTPEventConsumer* ist standardmäßig nicht registriert. Um ihn benutzen zu können, muss die Datei *smtpcons.mof*, die sich im Verzeichnis *%SystemRoot%\Winnt\wbem* befindet, mit Hilfe des MOF-Compilers (*mofcomp.exe*) kompiliert und ins WMI-Repository aufgenommen werden. Der Provider wird im Namespace *root\default* registriert.

Windows XP und Windows .NET enthalten drei weitere permanente Ereigniskonsumenten:

Neu in XP/.NET

1. Command Line-Ereigniskonsumenten: Start einer Anwendung
2. NT Event Log-Ereigniskonsumenten: Eintrag in das NT-Ereignisprotokoll
3. Log File-Ereigniskonsumenten: Eintrag in eine Protokolldatei

Der Ereigniskonsument definiert sein Interesse gegenüber WMI in Form eines WMI-Filters. Ein WMI-Filter ist im Wesentlichen eine *WQL Event Query* (vgl. nächster Abschnitt).

Filter

Ein Ereigniskonsument ist eine Instanz einer von der Systemklasse *__EventConsumer* erben Klasse. Ein WMI-Filter ist eine Instanz der Systemklasse *__EventFilter*. Die Bindung zwischen einem Consumer und einem Filter ist als Instanz von *__FilterToConsumer Binding* gespeichert.

Administration

Es gibt zwei Möglichkeiten, Consumer und Filter zu definieren und aneinander zu binden:

- über das Werkzeug *WMI Event Registration*
- über Programmcode. Dadurch, dass Consumer, Filter und Bindungen selbst wieder als WMI-Objekte gespeichert werden,

können diese leicht auch per WMI automatisiert verwaltet werden.

Provider Kern des Ereignissystems ist der *Event Provider*. Er informiert WMI über Veränderungen in der Managementumgebung oder im Repository. WMI leitet die Ereignisse an die für dieses Ereignis registrierten Konsumenten weiter.

C.5.8 WMI Query Language (WQL)

WQL WMI erlaubt es, Suchanfragen in einer Syntax zu stellen, die auf der ANSI Standard Structured Query Language (SQL) basiert. Der SQL-Dialekt heißt WMI Query Language, kurz: WQL. Es wird allerdings nur Lesezugriff mit dem SQL-Befehl `SELECT` unterstützt. Weder DDL (Data Definition Language) noch DML (Data Manipulation Language) werden unterstützt.

Schlüsselwörter Das WQL-`SELECT` unterstützt neben den Standardschlüsselwörtern `FROM`, `WHERE`, `GROUP BY`, `HAVING` und `WITHIN` auch die nicht in ADSI-SQL definierten Schlüsselwörter `ASSOCIATORS OF` und `REFERENCES OF`.

Operatoren Unterstützte Operatoren sind `=`, `<`, `>`, `<=`, `>=`, `!=` (alternativ: `<>`) sowie `IS NULL`, `IS NOT NULL` und `ISA`. Der `ISA`-Operator ermöglicht die Abfrage nach Unterklassen einer bestimmten Klasse. Wenn Dorf eine Unterklasse von Stadt ist, dann erfüllt ein Objekt Gallisches-Dorf die Bedingung `ISA Stadt`.

Beispiele Beispiele für Abfragen mit WQL sind:

```
Select * from Win32_Service where state='running' and
startmode='manual'
Select * from __InstanceModificationEvent within 5 where
Target Instance isa "Win32_Service" AND
TargetInstance.State="Stopped"
```



Der Microsoft Systems Management Server (SMS) enthält eine erweiterte Fassung von WQL unter dem Namen Extended WQL.

C.6 Werkzeuge

Microsoft liefert für die *Windows Management Instrumentation* (WMI) einige Tools, die mit dem WMI-SDK installiert werden. Dieses ist auf der CD enthalten [CD:/install/SDK/WMI SDK]. *WBEMTest* und *MofCom* sind allerdings Bestandteile des WMI-Kerns.

C.6.1 WMI-Steuerung

Zur Konfiguration von WMI liefert Microsoft unter Windows 2000/XP/.NET ein MMC-Snap-In und für alle anderen Windows-Versionen eine eigenständige *WMI Control Application* (*wbemcntl.exe*).

wbemcntl.exe

Das MMC-Snap-In heißt *WMI-Steuerung*. Das Snap-In implementiert bislang nur einen einzelnen Ast ohne Untereinträge. Einstellungen sind lediglich über das Eigenschaftsfenster dieses Astes möglich. Der Ast WMI-STEUERUNG ist auch Teil der vordefinierten Konsole »Computerverwaltung«.

MMC Snap-In

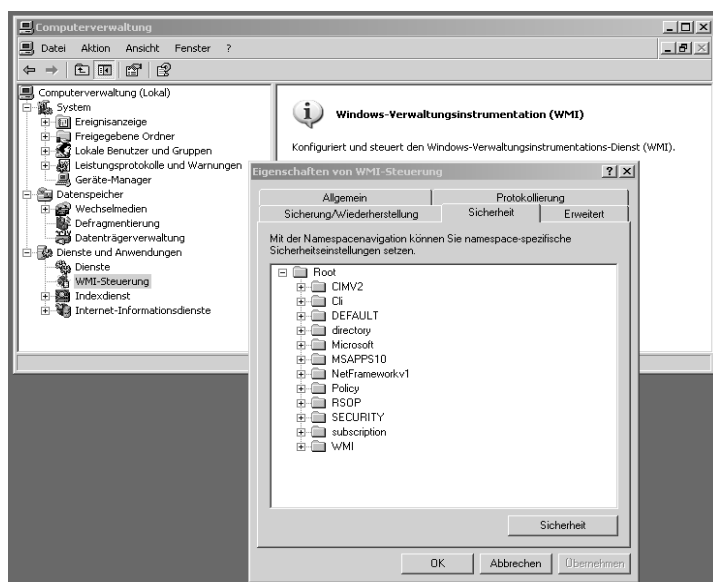


Abbildung C.5: MMC-Snap-In »WMI-Steuerung« (hier unter Windows XP)

C.6.2 WMI Object Browser

Der *WMI Object Browser* ist eine clientseitige Web-Anwendung für die Darstellung des aktuellen Objektmodells. Die HTML-Datei heißt *browser.htm* und wird bei der Installation des WMI-SDKs in das Unterverzeichnis */Applications* installiert. In einer Baumdarstellung (genannt *Object Explorer*) wird die aktuelle *Containment-Hierarchie* abgebildet. In der rechten Fensterhälfte (dem *Object Viewer*) werden zu einer ausgewählten Instanz die Methoden, Attribute und Beziehungen dargestellt (vgl. folgende Abbildung).

Containment-Hierarchie in browser.htm



WMI Object Browser versus Object Browser

Implementierung

Verwechseln Sie den WMI Object Browser nicht mit dem Objektkatalog, der in der englischen Version einiger Entwicklungsumgebungen Object Browser heißt. Während der WMI Object Browser ein spezielles Werkzeug zur Darstellung von WMI-Objekten ist, dient der allgemeine Object Browser der Anzeige von Typbibliotheken.

Der WMI Object Browser besteht aus HTML-Seiten mit DHTML-Scripten und ActiveX-Steuerelementen, wobei die Hauptfunktionalität durch letztere erbracht wird. Sie starten den WMI Object Browser über die Datei *browser.htm* bzw. über eine bei der Installation des WMI-SDKs angelegte Verknüpfung in Ihrem Startmenü.

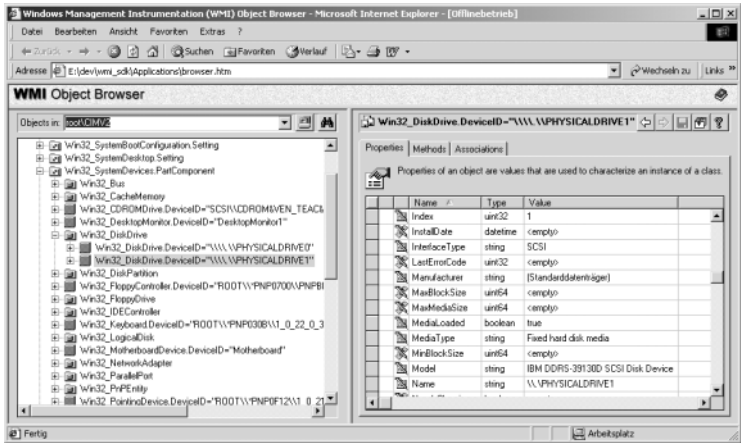


Abbildung C.6: Der WMI Object Browser zeigt an, dass der Computer zwei Festplatten besitzt, wobei die angewählte zweite Festplatte eine IBM DDRS-Festplatte mit SCSI-Schnittstelle ist.

Symbol	Erläuterung
	Markiert ein Systemattribut
	Markiert ein Schlüsselattribut. Eine Instanz ist anhand dieses Attributs eindeutig identifizierbar.
	Der Pfeil markiert ein von einer Oberklasse geerbtes Attribut. Der rote Strich soll einen Stift darstellen und bedeutet, dass der Attributwert verändert werden kann.

Abbildung C.7: Symbole im Object Viewer

Symbol	Erläuterung
	Dieses Attribut bezeichnet ein lokales (nicht geerbtes) Attribut. In der Fassung mit dem roten Strich ist das Attribut beschreibbar.
	Bedeutet, dass es sich um ein mehrwertiges Attribut handelt. Die Werte werden in einem separaten Fenster nach einem Klick auf dieses Symbol dargestellt.

Abbildung C.7: Symbole im Object Viewer (Fortsetzung)

Funktionsüberblick

Der Object Viewer bietet folgende Funktionen:

- ▶ Hilfe zu jeder WMI-Klasse (Fragezeichen-Icon). Die Sprache ist abhängig vom gewählten Namespace. *root\cimv2\ms_407* ist Deutsch, *root\cimv2\ms_409* ist Englisch. Beim Zugriff auf den übergeordneten *root\cimv2* ist die Sprache abhängig von Ihren Computereinstellungen. **Hilfe**
- ▶ Anzeige und Veränderung der Attribute. Veränderungen können direkt in der Attributtabelle vorgenommen werden. Änderungen müssen explizit persistent gemacht werden (Diskettensymbol). **Attribute**
- ▶ Anzeige der Methoden (Registerkarte METHODS) und ihrer Parameter (nach einem Doppelklick auf den Methodennamen). Über den Kontextmenüeintrag EXECUTE können Methoden ausgeführt werden. Zur Eingabe der notwendigen Parameter erscheint ein Dialogfenster. **Methoden**
- ▶ Darstellung der Beziehungen der Instanz zu anderen Instanzen in Form eines Graphen (Registerkarte ASSOCIATIONS, vgl. Abbildung). **Objektbeziehungen**

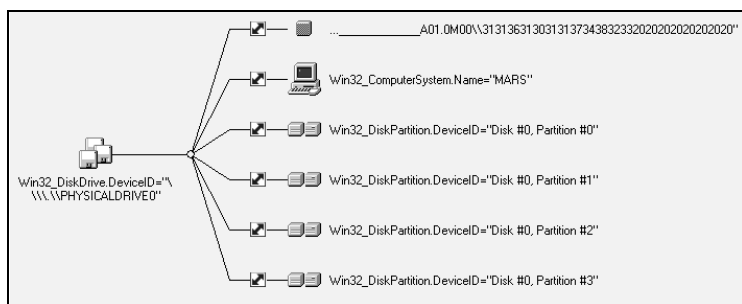


Abbildung C.8: Die Registerkarte »Associations« im Object Viewer des WMI Object Browsers zeigt die WMI-Instanz einer Festplatte, die vier Partitionen enthält.

C.6.3 WMI CIM Studio

Vererbungshierarchie in studio.htm

Das WMI CIM Studio setzt im Gegensatz zum WMI Object Browser auf der Klassenebene an und stellt die Vererbungshierarchie der CIM-Klassen dar. Diese wird in der linken Fensterhälfte, dem Class Explorer, gezeigt. Die rechte Fensterhälfte (genannt Class Viewer) ähnelt der Ansicht des WMI Object Browsers: Hier werden die Attribute, Methoden und Beziehungen angezeigt.



Das CIM Studio hat ebenso wie der WMI Object Browser den Dateinamen studio.htm innerhalb des WMI-SDKs.

Vergleich zum WMI Object Browser

Das CIM Studio umfasst auch einen Großteil der Funktionalität des WMI Object Browsers, da zu jeder Klasse eine Liste der vorhandenen Instanzen angezeigt werden kann. Zu jeder Instanz ist dann wiederum der Object Viewer mit der Anzeige der Attribute, Methoden und Beziehungen verfügbar. Nicht erreichbar im CIM Studio ist jedoch der Object Explorer (die linke Fensterhälfte aus dem WMI Object Browser).

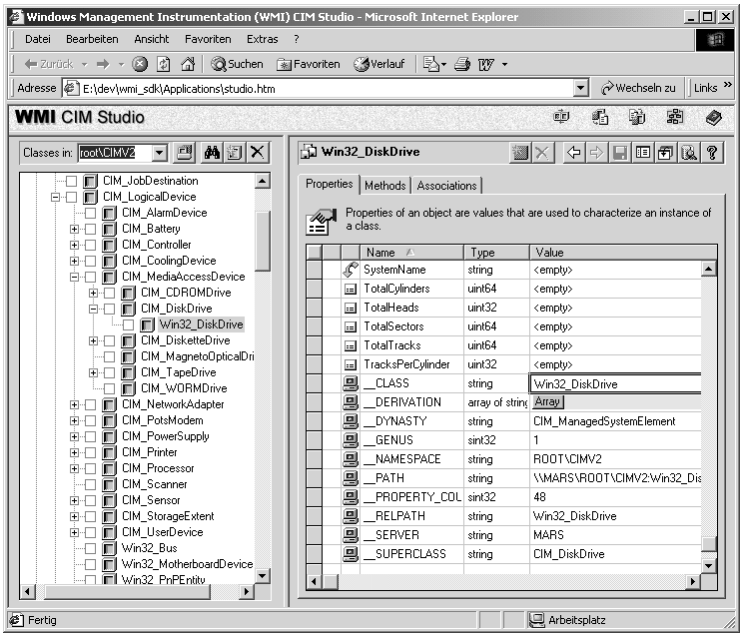


Abbildung C.9: Die Klasse Win32_DiskDrive im CIM Studio. Die Klasse ist eine Unterklasse von CIM_DiskDrive, die wiederum Unterklasse von CIM_MediaAccessDrive ist.

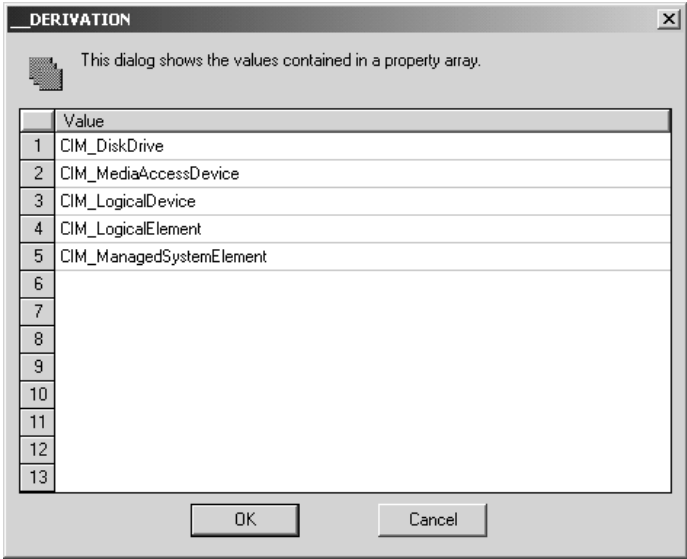


Abbildung C.10: Vererbungshierarchie der Klasse Win32_DiskDrive

Funktionsüberblick

Die Registerkarte PROPERTIES (vgl. folgende Abbildung) zeigt natürlich nur die im Repository gespeicherten globalen Klasseneigenschaften an, die bei allen Instanzen der Klasse gleich sind (z. B. Klassenname, Namespace, Oberklasse). Diese Eigenschaften sind an den beiden führenden Unterstrichen (»__«) erkennbar; für alle anderen Eigenschaften, die ja nur in einer konkreten Instanz belegt sind, wird »<EMPTY>« angezeigt.

Attribute

Der Reiter METHODS zeigt analog zum WMI Object Browser die durch die Klasse implementierten Methoden an. Dabei sind die statischen Methoden ausführbar, da hier ja keine Instanz vorhanden ist.

Methoden

Unter ASSOCIATIONS erscheint ein Graph, der anzeigt, welche Unterklassen die Klasse enthalten kann.

Assoziationen

Symbol	Erläuterung
	Erzeugen neuer Instanzen dieser Klasse
	Zeigt die Instanzen dieser Klasse (Tabellensymbol)

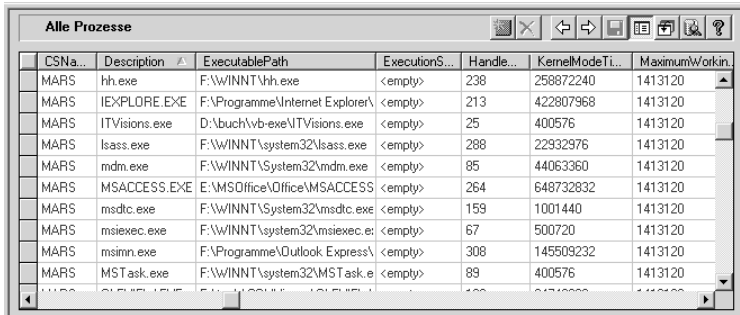
Abbildung C.11: Icons im Class Viewer

Symbol	Erläuterung
	Erstellen, Testen und Speichern von WQL-Abfragen
	Zeigt die Hilfeinformationen zu dieser Klasse an

Abbildung C.11: Icons im Class Viewer (Fortsetzung)

Instanzenansicht

Über das Tabellensymbol in der Symbolleiste des Class Viewers gelangen Sie zur Darstellung der Instanzen dieser Klasse in Form einer Tabelle (vgl. folgende Abbildung). Durch einen Doppelklick auf einen Eintrag der Tabelle erscheint der Object Viewer für diese Instanz mit den schon beim WMI Object Browser beschriebenen Fähigkeiten.



CSNa...	Description	ExecutablePath	ExecutionS...	Handle...	KernelModeTI...	MaximumWorkin...
MARS	hh.exe	F:\WINNT\hh.exe	<empty>	238	258872240	1413120
MARS	IEXPLORE.EXE	F:\Programme\Internet Explorer\	<empty>	213	422807968	1413120
MARS	ITVisions.exe	D:\buch\vb-exe\ITVisions.exe	<empty>	25	400576	1413120
MARS	lsass.exe	F:\WINNT\system32\lsass.exe	<empty>	288	22932976	1413120
MARS	mdm.exe	F:\WINNT\system32\mdm.exe	<empty>	85	44063360	1413120
MARS	MSACCESS.EXE	E:\MSOffice\Office\MSACCESS	<empty>	264	648732832	1413120
MARS	msdtc.exe	F:\WINNT\system32\msdtc.exe	<empty>	159	1001440	1413120
MARS	msiexec.exe	F:\WINNT\system32\msiexec.e	<empty>	67	500720	1413120
MARS	msimn.exe	F:\Programme\Outlook Express\	<empty>	308	145509232	1413120
MARS	MSTask.exe	F:\WINNT\system32\MSTask.e	<empty>	89	400576	1413120

Abbildung C.12: Listen der Instanzen der Klasse WIN32_Process in der Instanzenansicht des CIM Studios

Weitere Funktionen

Weitere Features
des Class Viewers

- Weiterhin bietet der Class Viewer folgende Möglichkeiten:
- Ausführen von WQL-Abfragen. Abfragen können zur späteren Verwendung auch gespeichert werden.
 - Erzeugen neuer Instanzen einer Klasse (blauer Kasten mit Stern in der Symbolleiste). Das Symbol ist nur aktiv, wenn eine Klasse ausgewählt wurde, die Instanzen haben kann. Das CIM Studio zeigt Ihnen daraufhin einen Object Viewer mit leeren Feldwerten. In vielen Fällen (z.B. bei WIN32_Process) sollten jedoch die entsprechenden Konstruktormethoden verwendet werden, anstatt mühsam zu versuchen, passende Werte in die neue Instanz einzutragen.

- ▶ Löschen bestehender Instanzen
- ▶ Hinzufügen von neuen Klassen
- ▶ Ändern bestehender Klassen
- ▶ Löschen von Klassen
- ▶ Verschiedene Wizards zur Code-Generierung (z.B. Managed Object Format-Dateien; Framework-Code für WMI-Provider)

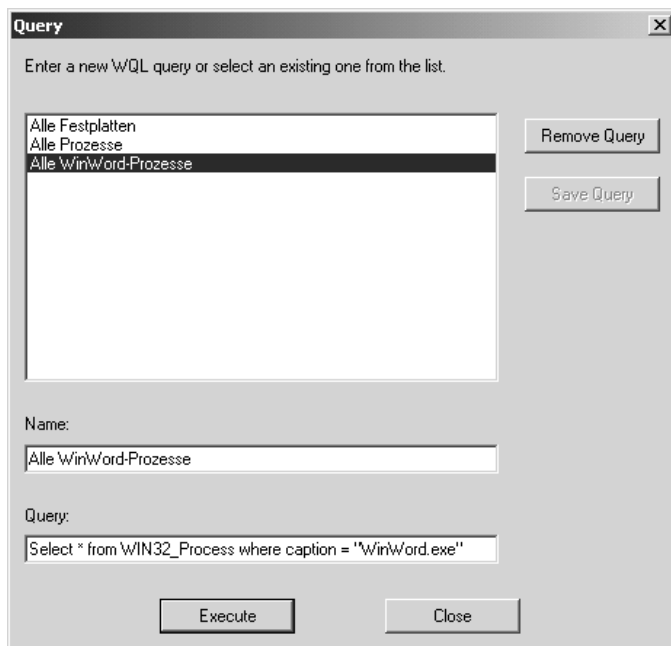


Abbildung C.13: WQL-Abfrage nach den aktiven Winword-Prozessen im Abfragefenster des CIM Studios

C.6.4 WMI Event Registration Tool

Das *WMI Event Registration Tool* (*eventreg.htm*) ist ein GUI zur Konfiguration von Ereigniskonsumenten. Um das Werkzeug einsetzen zu können, müssen Sie WQL Event Queries beherrschen.

eventreg.htm

Die Arbeit mit dem Registration Tool ist leider wenig intuitiv. Im Folgenden ist der Ablauf der Konfiguration der Ereignisüberwachung für den WMI Event Viewer dargestellt. Die zugehörige Ereigniskonsumentenklasse *EventViewerConsumer* ist im Namespace */root/cimv2* registriert.

Beispiel

Konsument definieren

- Definieren Sie zunächst einen Ereigniskonsumenten:
 - Wählen Sie erst in dem Drop-Down-Menü in der Symbolleiste CONSUMERS.
 - Klicken Sie bei der Klasse `__EventConsumer` auf das »+«-Zeichen, bis Sie die Unterklasse `EventViewerConsumer` sehen. Markieren Sie dann `EventViewerConsumer` und wählen Sie im Kontextmenü NEW INSTANCE.
 - Geben Sie in dem Dialogfenster einen beliebigen Namen und eine DESCRIPTION ein. Wenn das Ereignis auf einem entfernten System abgefangen werden soll, geben Sie den Namen oder die IP-Adresse des Rechners bei MACHINE NAME an.
 - Tragen Sie bei SEVERITY einen Wert größer/gleich 0 ein; dabei ist der Fehler umso schwerer, je kleiner der Wert ist. Der Event Viewer betrachtet »0« als einen Fehler, den Wert »1« als eine Warnung und alles größer »1« als eine Information. Die Microsoft WMI-Dokumentation ist in diesem Punkt fehlerhaft.

Filter definieren

- Definieren Sie dann einen Filter:
 - Wählen Sie erst in dem Drop-Down-Menü in der Symbolleiste FILTERS.
 - Klicken Sie auf die Klasse `__EventFilter` und wählen Sie im Kontextmenü NEW INSTANCE.
 - Geben Sie in dem Dialogfenster einen beliebigen Namen und als QUERYLANGUAGE »WQL« ein. Geben Sie unter QUERY eine gültige Event Query ein. Eine gültige Event Query ist z.B. `SELECT * FROM __InstanceModificationEvent WITHIN 5 WHERE TargetInstance isa "Win32_Service" AND TargetInstance.State="Stopped"`, um ein Ereignis auszulösen, wenn ein NT-Dienst stoppt.

Bindung des Konsumenten an einen Filter

- Bindung des Consumers an einen Filter:
 - Ebenfalls in der FILTERS-Ansicht werden bei Auswahl eines zuvor angelegten Filters in der rechten Bildschirmhälfte die verfügbaren Consumer angezeigt. Wählen Sie den im ersten Schritt angelegten Consumer und dort im Kontextmenü REGISTER. Das gleiche Ergebnis erreichen Sie auch aus der Ansicht CONSUMER. Dort stehen Ihnen die zuvor definierten Filter zur Auswahl.

Damit ist die Konfiguration abgeschlossen. Alle auftretenden Ereignisse werden dann im WMI Event Viewer angezeigt.

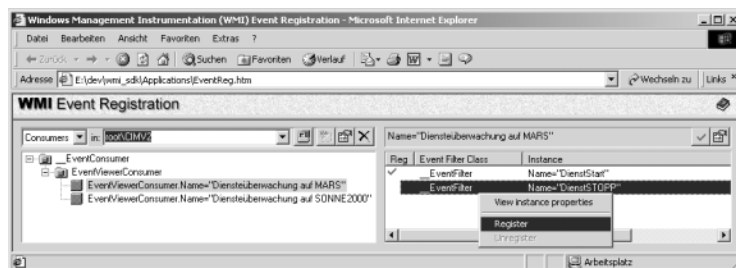


Abbildung C.14: Bindung einer Instanz von `__EventFilter` an eine Instanz von `EventViewerConsumer`

C.6.5 WMI Event Viewer

Der *WMI Event Viewer* ist das einzige unter den WMI SDK-Tools, das keine HTML-Anwendung, sondern eine ausführbare Datei (*wbemeventviewer.exe*) ist. Der Event Viewer ist ein permanenter WMI-Ereigniskonsumter, der durch eine `__EventFilter`-Instanz definierte Ereignisse auf dem Bildschirm darstellt. Der Event Viewer wird über die WMI-Klasse `EventViewerConsumer` konfiguriert. Diese Konfiguration können Sie über das WMI Event Registration Tool oder über den Event Viewer selbst durchführen.

**wbemevent-
viewer.exe**

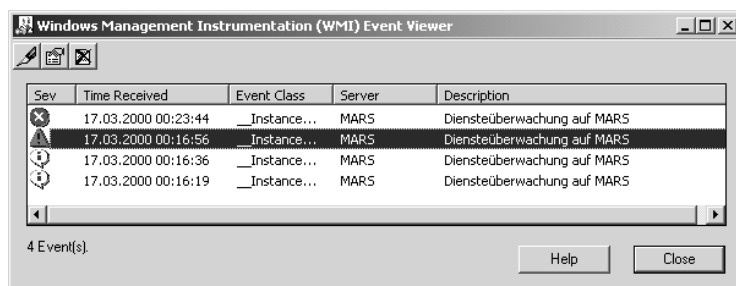


Abbildung C.15: Anzeige eines Ereignisses im Event Viewer: Durch einen Doppelklick erhalten Sie Details zu dem Ereignis. Dabei sehen Sie auch alle Attributwerte des auslösenden MOs.

C.6.6 VBInstance

VBInstance.vbp Microsoft liefert im WMI-SDK eine Visual Basic-Beispielanwendung, die sich als Tool eignet. Das Beispiel *VBInstance* im WMI-SDK liefert eine grafische Darstellung für ein beliebiges numerisches Attribut für die Instanzen einer beliebigen WMI-Klasse (siehe folgende Abbildung). Da im SDK leider keine kompilierte Version mitgeliefert wurde, finden Sie eine solche auf der CD zu diesem Buch. Sie können damit auf einfache Art Grafiken wie z. B. den Vergleich der Größe aller verfügbaren Laufwerke erstellen.

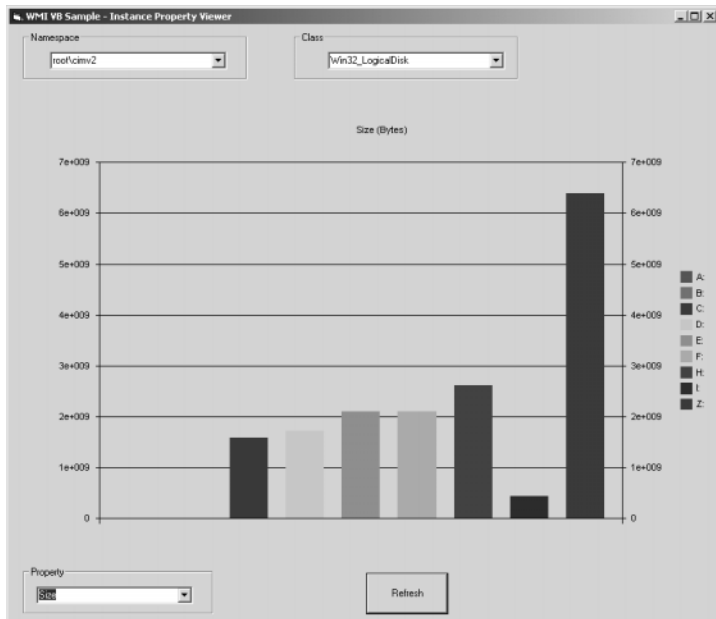


Abbildung C.16: VBInstance zeigt die Größe der verfügbaren Laufwerke an. Die Größe der Disketten in den Laufwerken A: und B: ist zu klein, als dass sie im Graph sichtbar würde.

C.6.7 WMI-Testprogramm

WBEMTest.exe *WBEMTest.exe* gehört zum Standardinstallationsumfang (Verzeichnis %System%\WBEM) und bietet einen einfachen, wenig komfortablen GUI zur Ausführung von Operationen auf dem Repository (siehe folgende Abbildung). Das Werkzeug eignet sich, wie der Name schon sagt, zum Testen von WMI. Alle Operationen von *WBEMTest* können auch mit dem CIM Studio ausgeführt werden.

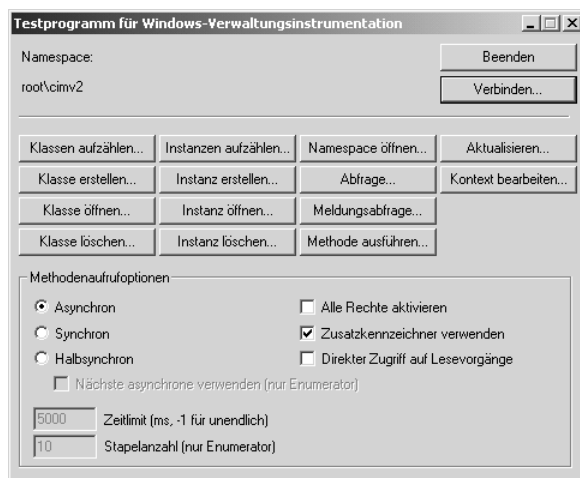


Abbildung C.17: WBEMTest.exe

C.6.8 MOF Compiler

Der *MOF Compiler* (*mofcomp.exe*) ist ein DOS-Programm und dient der Aufnahme von Managed Object Format-Dateien in das Repository (vgl. Ausführungen zu MOF in Kapitel 1.5.5). Der MOF Compiler besitzt eine Reihe von Kommandozeilenoptionen, die Sie sehen, wenn Sie *mofcomp* ohne weitere Parameter an der Kommandozeile starten. Der Befehl *MofComp Datei.mof* übernimmt die angegebene MOF-Datei in das Repository.

**MOF-Dateien
kompilieren**

D Hinweise zum Buch

D.1 Sprachliche Konventionen

Die gleiche Sprache zu sprechen ist in der immer komplizierter werdenden IT-Welt nicht einfach. Dieses Kapitel dokumentiert einige sprachliche Konventionen, die in diesem Buch zur Anwendung kommen.

- ▶ Die Verwendung des Begriffs »Windows NT« oder einfach »NT« umfasst die gesamte NT-Produktfamilie, also Windows NT 4.0, Windows 2000, Windows XP und Windows .NET. **NT**
Windows 2000, Windows XP und Windows .NET sind nur Marketing-Begriffe; das Betriebssystem heißt intern »Windows NT« (vgl. Registry-Schlüssel `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\WindowsNT\CurrentVersion`).
Der Begriff »Windows NT 4.0« (oder kurz »NT4«) umfasst dagegen Windows 2000 und die Folgeversionen nicht.
- ▶ Windows 9x steht für Windows 95 und Windows 98. **Windows 9x**
- ▶ Windows ME steht für die Windows Millennium Edition. **ME**
- ▶ Visual Basic (oder kurz »VB«) wird als Oberbegriff über die Visual Basic-Sprachfamilie, einschließlich Visual Basic .NET (VB.NET) verwendet. **VB**
- ▶ Dieses Buch verwendet absichtlich viele englische Fachbegriffe bzw. Anglizismen, da deutsche Übersetzungen oft mehrdeutig bzw. kompliziert sind. Außerdem haben sich die englischen Fachbegriffe unter Experten so sehr etabliert, dass es manchmal nur zu Verwunderung führt, wenn man deutsche Übersetzungen verwendet. **Englische Begriffe**
- ▶ Methodennamen werden stets – unabhängig von der Parameteranzahl – durch ein Klammersymbol »()« kenntlich gemacht und damit von Attributnamen abgegrenzt. **Methodenname**

- Verkürzung** ▶ Um den Text besser lesbar zu machen, wird nach der Einführung eines Klassen- oder Mitgliedsnamens häufig auf die Erwähnung der Begriffe Klasse, Objekt, Methode, Attribut und Ereignis verzichtet. Zum Beispiel: »Ein `DataSet`-Objekt kann nicht nur eines, sondern mehrere `DataTable`-Objekte aufnehmen. Ein `DataSet` ist daher mächtiger als ein `RecordSet`.«
- ▶ Ebenso wird eine Gruppe von Klassen bei Bedarf zu einem Begriff zusammengefasst. Beispiel: `CommandBuilder` steht sowohl für `OleDbCommandBuilder` als auch für `SqlCommandBuilder` (und alle anderen gleichbedeutenden Klassen in anderen ADO.NET Data Providern).
- Attribut und Meta-Attribut** ▶ In Abgrenzung zum Daten-Mitglied einer Klasse, das in der objektorientierten Lehre *Attribut* genannt wird (vgl. für den deutschen Sprachraum [OES97], S. 157 und für den englischen Sprachraum [OXF97], S. 243) und zu den zusätzlichen Auszeichnungen, die Microsoft *Attribut* nennt, wird in diesem Buch für letzteren Fall der Begriff *Meta-Attribut* verwendet. Meta-Attribut hebt dabei hervor, dass es sich nicht um eine Programmanweisung, sondern um eine zusätzliche Beschreibung handelt.
- MINFU** ▶ Auf Basis der Erkenntnis, dass Microsoft regelmäßig Probleme mit der Bezeichnung der eigenen Produkte und Konzepte hat, schuf der amerikanische Autor David S. Platt ein neues Wort: MINFU. Dies ist eine Abkürzung für **M**icrosoft **N**omenclature **F**oul-Up (siehe [PLA00] und [PLA01], Seite IX).

D.2 Hinweise zur Formatierung des Textes

Damit es zu keinen Missverständnissen bei der Eingabe von *Programcode*, *URLs* u.ä. kommt, wurde beim Umbruch auf den Trennstrich verzichtet.

Quellcode sowie die Namen von Klassen, Schnittstellen, Attributen, Methoden und Ereignissen erkennen Sie an der nicht-proportionalen Schrift.

In *kursiver proportionaler Schrift* finden Sie Textausgaben von Programmen.

Kursiv gesetzt sind Hyperlinks und andere Pfadangaben. Außerdem sind Begriffe kursiv gesetzt, wenn diese erstmalig eingeführt werden oder der Begriff aus mehreren Wörtern besteht und es zu

Missverständnissen kommen könnte, welche Wörter zu dem Begriff gehören.

Fett geschrieben sind Wörter, die besonders betont werden sollen.

In KAPITÄLCHEN finden Sie alle Namen, die Bilschirmelementen entsprechen (z. B. Registerkarten, Menüeinträge, Schaltflächen).

Manche Abschnitte in diesem Buch sind durch Symbole und *kur-sive* Textauszeichnung besonders gekennzeichnet. Diese Symbole haben folgende Bedeutung:

Wichtige *Hinweise* und *Tipps* finden Sie in Abschnitten, die mit diesem Symbol gekennzeichnet sind.



Technische Hinweise sind mit diesem Symbol hervorgehoben.



Abschnitte mit diesem Symbol sprechen eine Warnung aus.



D.3 Hinweise zu den Code-Beispielen

Alle Listings sind auf der dem Buch beiliegenden CD-ROM im Verzeichnis `/code` enthalten. In der Regel finden Sie unter dem Listing eine eckige Klammer, die den Namen der Datei auf der CD enthält. Um Platz zu sparen, ist der komplette Pfadname in den meisten Fällen nicht angegeben. Sie werden jedoch jede Datei leicht finden, wenn Sie die folgenden Hinweise beachten.

`/code`

Ablaufumgebung »FCL-Buch«

Alle Beispiele der Kapitel 4, 6, 10, 11, 14, 15 und 22 sind zu einem VB.NET-Projekt mit dem Namen »FCL-Buch« zusammengefasst. »FCL-Buch« ist Teil einer gleichnamigen Projektmappe (Solution). »FCL-Buch« bildet eine einfache Ablaufumgebung für alle Beispiele aus den o.g. Kapiteln.

»FCL-Buch«

Das Projekt besteht aus zahlreichen Unterverzeichnissen, die den Kapiteln dieses Buchs entsprechen. Jedes Unterverzeichnis enthält verschiedene Code-Dateien. Unter den einzelnen Listings finden Sie immer in eckigen Klammern den Namen der Code-Datei, in der sich das Listing befindet, z. B.

```

' =====
' Beispiel aus FCL-Buch.sln
' Ändern von Attributwerten in einem WMI-Objekt
' (C) Holger@Schwichtenberg.de
' =====

Sub ADONET_DoCmd()
...
End Sub

```

Listing D.1: ... [ADONET_Command.vb]

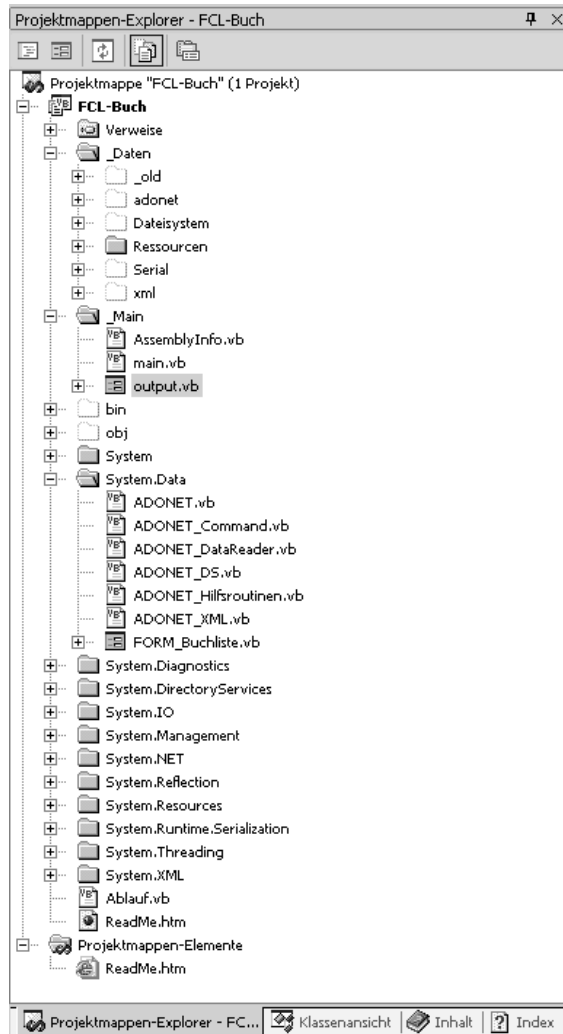


Abbildung D.1: Inhalt der Projektmappe »FCL-Buch«

Das Verzeichnis *_Daten* enthält alle Datendateien, mit denen die Beispiele in diesem Projekt arbeiten. Das Verzeichnis *_Main* enthält die Implementierung eines Ausgabefensters für die Code-Beispiele.

Das Ausgabefenster (siehe folgende Abbildung) erscheint nach dem Start des Projekts und bietet folgende Möglichkeiten:

Ausgabefenster

- ▶ Neustart des Beispiels (mit oder ohne Ausgabe von Zeilennummern vor den Ausgabezeilen)
- ▶ Löschen des Ausgabefensters
- ▶ Beenden der Anwendung

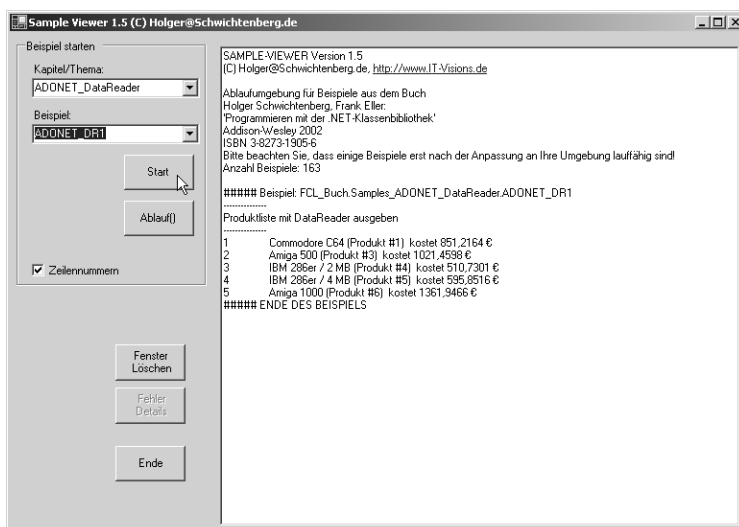


Abbildung D.2: Ausgabefenster des Projekts »FCL-Buch«

Welche(s) Beispiel(e) Sie starten wollen, legen Sie in dem Modul *Ablauf.vb* in der Unteroutine *Ablauf()* fest. Rufen Sie dort einfach die entsprechende(n) Unteroutine(n) auf, z.B.

Ablauf()

```
Public Sub ablaufr()
    ADONET_DoCmd()
End Sub
```

In fast allen Listings in dem Projekt »FCL-Buch« wird als Ausgabefunktion *out()* verwendet. *out()* ist in dem Modul *main.vb* implementiert und erzeugt gleichzeitig eine Ausgabe in das Ausgabefenster der Ablaufumgebung und an der Standardausgabe (*Console.WriteLine()*).

out()

Andere Beispiele

Bei allen anderen Beispielen finden Sie den Namen des Projekts bzw. der Projektmappe in der Kopfzeile des Listings und den Namen der Code-Datei, die den Listingcode enthält, in der Listing-Unterschrift.

Variablennamen

Variablennamen Aus Platzgründen wurde in den Listings in diesem Buch oft auf lange Variablennamen verzichtet. Die Listings sind in der Regel so kurz, dass dies nicht zu Nachteilen bei der Lesbarkeit führt. Für Laufvariablen wurden einbuchstabige Namen wie `a`, `b`, `x` und `y` verwendet. Namen für Objektvariablen sind meist Abkürzungen der Klassennamen, wobei das Präfix `obj` oder `o` vorangestellt wurde. Sofern der Klassenname nicht eindeutig ist, werden auch `o` und `obj` als einzelne Variablen verwendet. Namen für Collections beginnen häufig mit dem Präfix `col`.

Kommentare

Für die Kommentarzeilen gibt es eine bestimmte Notation, die in der folgenden Tabelle dokumentiert ist.

Kommentarzeichen	Bedeutung
<code>^ ===</code>	Kommentar zu einem eigenständigen Beispiel
<code>^ ###</code>	Kommentar zu einer Hilfsroutine, die von einem eigenständigen Beispiel oder einer anderen Hilfsroutine aufgerufen wird
<code>^ ---</code>	Zentraler Kommentar innerhalb einer Routine
<code>^ %q% %i% %o%</code>	Beginn einer Liste von Konstantendefinitionen
<code>^</code>	Sonstige Kommentare innerhalb einer Routine

Tabelle D.1: Kommentarzeichen

Zeilenumbrüche

Die Zeilenlänge in einem Buch ist begrenzt. Daher ist es oftmals notwendig, Zeilenumbrüche einzufügen, wo man normalerweise keine Zeilenumbrüche machen würde. In Fällen, wo durch Anführungszeichen klar ist, dass Zeilen zusammengehören, wurde auf das Einfügen eines manuellen Zeilenwechsels (`»_«` in Visual Basic) verzichtet.

D.4 Grafische Notation in den Objektdiagrammen

Eine wichtige Leistung dieses Buchs ist es, Ihnen zu den Komponenten grafische Darstellungen der Objektmodelle in Form von Objektdiagrammen an die Hand zu geben, die es ermöglichen, die grundsätzliche Navigation im Objektmodell einer Komponente wesentlich schneller zu erfassen, als dies mit einer textlichen Beschreibung möglich wäre.

Da die Objektmodelle oft sehr umfangreich und die Navigationspfade komplex sind, beschränkt sich die Darstellung in der Regel auf die wichtigsten Klassen und deren Zusammenhänge.

Die Objektmodelle stellen die Containment-Hierarchie der Objekte zur Laufzeit dar, nicht die Vererbungshierarchie der Klassen. Allerdings müssen nicht alle in den Objektmodellen dargestellten Beziehungen zu jedem Zeitpunkt existieren. Die Objektdiagramme sind die **abstrakte** Darstellung möglicher Beziehungen.

Als Notation wurde bewusst nicht die Unified Modelling Language (UML) gewählt, sondern eine eigene Notation, die sich an der üblichen Microsoft-Dokumentation orientiert und in diesem speziellen Fall wesentlich einfacher zu lesen ist als UML.

Eigene Notation

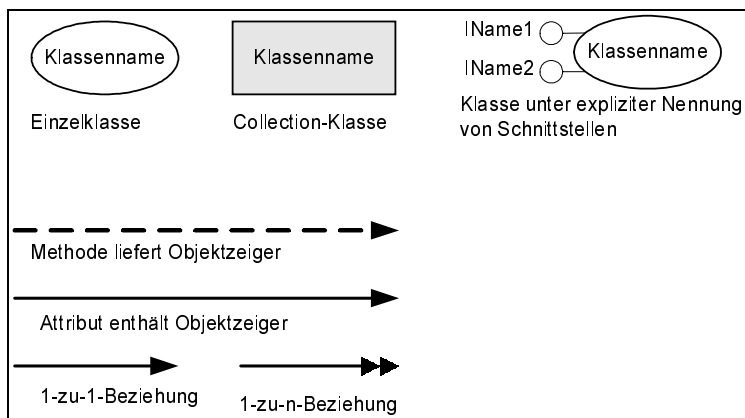


Abbildung D.3: Grafische Elemente zur Darstellung von Objektdiagrammen

Grundsätzlicher Aufbau

Ein Objektdiagramm ist ein Graph mit Knoten und Kanten. Die Knoten sind Klassen oder elementare Datentypen, die hier als geometrische Formen dargestellt werden. Die Kanten drücken Beziehungen zwischen den Klassen/Datentypen aus; sie sind als Verbindungslinien in Form von Pfeilen dargestellt. Bemerkungen sind als Callouts (Sprechblasen) mit schwarzem Hintergrund dargestellt.

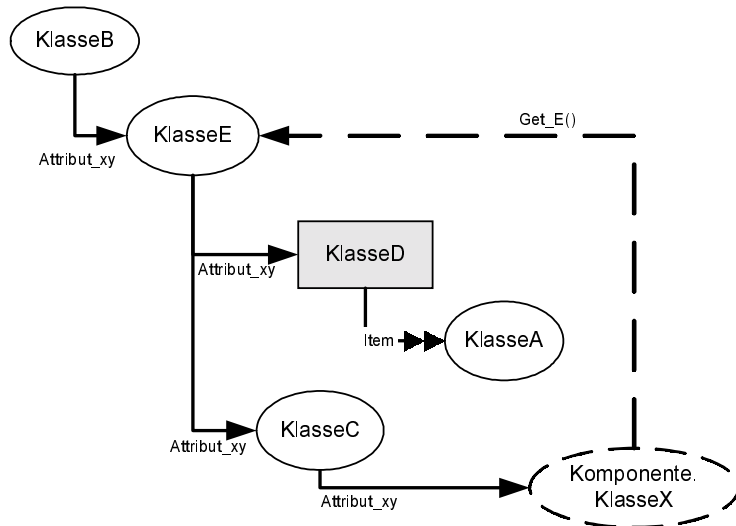


Abbildung D.4: Beispiel für die grafische Darstellung eines Objektmodells in diesem Buch

Die Objektdiagramme von Verzeichnisdienst-Objektmodellen sind mit zusätzlichen Icons versehen, die nur der Illustration dienen und keine inhaltliche Bedeutung haben.

D.4.1 Knoten (geometrische Formen)

Knoten In den Objektdiagrammen werden folgende Bausteine als Knoten verwendet:

- Einfache Klassen sind als Ovale dargestellt.
- Objektmengen in Form von Collection-/Container-Klassen sind dagegen in Form von Rechtecken dargestellt.

Sofern eine Klasse in einem Objektmodell mehrfach eingezeichnet ist, bedeutet dies, dass Instanzen dieser Klasse in verschiedenen Zusammenhängen vorkommen können. Allerdings bedeutet das im Umkehrschluss nicht, dass mehrere Verweise auf ein und den gleichen grafischen Baustein immer ein und die gleiche Instanz repräsentieren.

Sofern ein Oval oder ein Rechteck eine gestrichelte Umrandung hat, bedeutet dies, dass die Klasse nicht in dieser, sondern in einer anderen Komponente implementiert ist. Es gibt also eine Verbindung zu einem anderen Objektmodell. Der Name ist dann in der voll qualifizierten Form `Namespace.UnterNamespace.Typname` angegeben.

D.4.2 Kanten (Verbindungslinien)

Die Kanten stellen Nutzungsbeziehungen auf Basis von Attributen oder Methoden dar. Die Kanten sind als gerichtete Pfeile dargestellt, wobei der Pfeil immer bei der Klasse beginnt, in der das Attribut bzw. die Methode implementiert ist. Die Pfeilspitze weist auf die Klassen, deren Instanz von dem Attribut oder der Methode als Ergebnis geliefert wird.

Kanten

Die Notation unterscheidet nicht zwischen Assoziationen und Aggregationen, da letztere zum einen sehr selten vorkommen und sich zum anderen von der Nutzung her nicht von den Assoziationen unterscheiden. Verbindungslinien stellen Nutzungsbeziehungen dar. Die Notation unterscheidet allerdings zwischen dem Zugriff über ein Attribut und dem über eine Methode.

Nutzungsbeziehungen auf Basis von Attributen

Die durchgezogenen Pfeile sind Nutzungsbeziehungen auf Basis von Attributen:

Durchgezogene Linien

- ▶ Ein durchgezogener Pfeil von einem Oval zu einem Rechteck bedeutet, dass eine Klasse eine Collection zur Speicherung von Objektmengen enthält. Sofern diese Linie nicht beschriftet ist, ist die Collection über ein Attribut der Klasse erreichbar, das genauso heißt wie die Collection. Heißt das Attribut anders, steht der Attributname als Beschriftung an der Linie.
- ▶ Ein durchgezogener Pfeil von einem Rechteck zu einem Oval bedeutet, dass eine Collection/ein Container aus Objekten der Klasse besteht. Hier erfolgt üblicherweise der Zugriff auf die

untergeordneten Objekte über das Attribut `Item`; nur wenn dies nicht zutrifft, ist das Attribut explizit angegeben. Auch in den Fällen, in denen `Item` als Methode implementiert ist, wird konsistent eine durchgezogene Linie verwendet.

- Ein durchgezogener Pfeil von einer Klasse zu einer anderen Klasse bedeutet, dass eine Instanz der übergeordneten Klasse genau ein Unterobjekt der untergeordneten Klasse erhalten wird. Auch hier erfolgt die Angabe des Attributs nur, wenn dies nicht gleich dem Klassennamen ist.

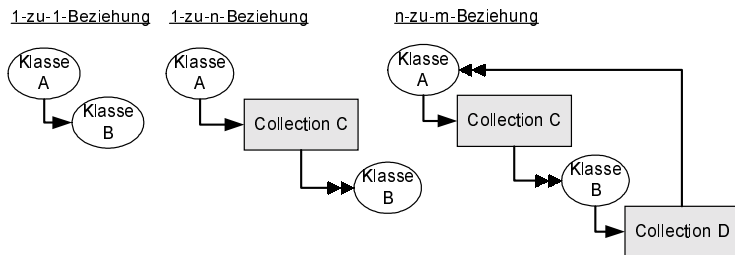


Abbildung D.5: Modellierung der Nutzungsbeziehungen mit verschiedenen Kardinalitäten

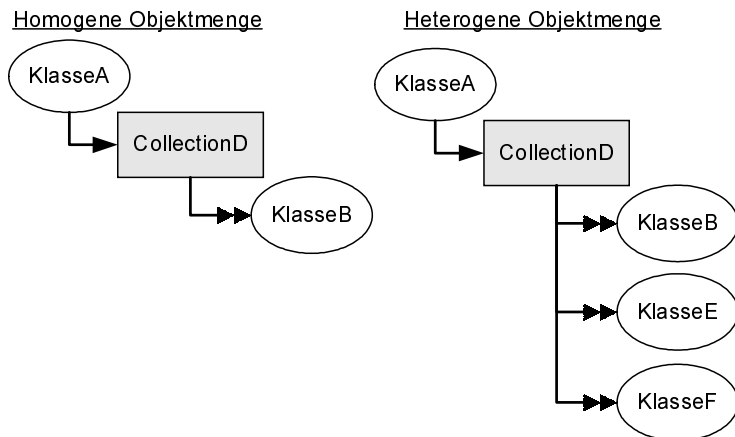


Abbildung D.6: Unterscheidung zwischen homogener und heterogener Collection

Von Klassen, die Objektmengen repräsentieren, können Pfeile mit einer doppelten Pfeilspitze ausgehen. Dies bedeutet, dass die Objektmenge n Unterobjekte dieser Klasse enthalten kann (1-zu- n -

Beziehung). Eine einfache Pfeilspitze ist die Beschränkung auf eine Instanz (1-zu-1-Beziehung). Sind an dem Pfeil allerdings mehrere Attribut- bzw. Methodennamen genannt, so kann jedes der genannten Attribute eine andere Instanz liefern.

Kann-Beziehungen werden von Muss-Beziehungen in der grafischen Notation nicht unterschieden.



Nutzungsbeziehungen auf Basis von Methoden

Gestrichelte Pfeile sind Zugriffspfade auf Basis von Methodenaufrufen. Eine gestrichelte Linie mit der Beschriftung `mName()` bedeutet, dass ein Aufruf der Methode `mName()` auf einer Instanz der Ausgangsklasse eine Instanz der Klasse liefert, auf die die Spitze des Pfeils zeigt. Der Methodenname ist aus Gründen der Übersichtlichkeit ohne Parameterliste angegeben. Nicht dargestellt sind Nutzungsbeziehungen auf Basis von Methodenaufrufen, die als Parameter eine Instanz einer bestimmten Klasse erwarten.

Gestrichelte Linien

- ▶ Eine gestrichelte Linie, die nicht bei einem Objekt beginnt, sondern aus dem leeren Raum kommt, bedeutet, dass der Zugriff über eine in einer Komponente definierte globale Methode erfolgt, die nach Einbindung der Komponente global zur Verfügung steht.
- ▶ Sofern der Weg von Instanzen einer Klasse zu Instanzen anderer Klassen auf mehreren Wegen beschriftet werden kann, sind die Attribut- bzw. Methodennamen durch Kommata getrennt als Beschriftung der Linie angegeben. Im Normalfall beziehen sich alle genannten Attribute bzw. Methoden auf eine andere Instanz der Zielklasse. Sofern es sich um einen Verweis auf ein und dieselbe Instanz handelt, sind die synonymen Attribute und Methoden in Klammern gesetzt.

Weitere Hinweise

Eine sich gabelnde Linie bedeutet, dass das Attribut bzw. die Methode Instanzen verschiedener Klassen zurückliefern kann.

Gabelung

Vollständigkeit

Auf eine vollständige Auflistung aller Attribute, Methoden und Ereignisse innerhalb der Grafiken wurde bewusst verzichtet, da diese sonst sehr unübersichtlich geworden wären. Mitglieder sind

Vollständigkeit

in Form der Verbindungslinien nur insofern dargestellt, als sie Zeiger auf andere Objekte zurückliefern und damit das Objektmodell bestimmen. Hinweise zu weiteren Attributen, Methoden und Ereignissen finden Sie im Text.

E CD-ROM und Website

E.1 Der Inhalt der CD-ROM

Die diesem Buch beiliegende CD-ROM enthält folgende Verzeichnisse:

<i>/code</i>	Das Verzeichnis enthält alle Programmbeispiele aus dem Buch, geordnet nach Kapiteln und Unterkapiteln.
<i>/install</i>	Add-ons und Software Development Kits
<i>/Website zum Buch</i>	Hier finden Sie eine HTML-Datei, die Ihnen sagt, wie Sie sich in den geschützten Leser-Bereich auf der Website zu diesem Buch anmelden können.
<i>/Weitere Informationen</i>	Dieses Verzeichnis enthält einige Original-Dokumentationen sowie andere frei verfügbare Informationen.

Verweise auf die CD-ROM finden Sie im Buch in eckigen Klammern, in der Form [CD:/pfad/Dateiname], [CD:/pfad] oder einfach [Dateiname]. Den letztgenannten Fall wenden wir nur für Code-Beispiele an, die Sie alle im Verzeichnis */code* finden.

Verweise auf die CD

E.2 Die Website zu diesem Buch

Zu diesem Buch gibt es eine eigene Website:

<http://www.dotnet-komponenten.de>

Sie als Leser haben neben den öffentlichen Bereichen auch die Möglichkeit, auf einen zugangsbeschränkten Bereich zuzugreifen:

<http://www.dotnet-komponenten.de/leser>

Dieser zugangsbeschränkte Bereich enthält:

- **Code-Archiv:** Die aktuellsten Versionen der in diesem Buch abgedruckten Code-Beispiele sowie weitere Beispiele.

Weitere Informationen und Unterstützung im WWW

- ▶ **Diskussionsrunde:** Wenn Sie Fragen haben oder eine Meinung zu einem Thema dieses Buches äußern möchten, dann können Sie hier auf Reaktionen anderer Leser und des Autors hoffen.
- ▶ **Bug-Report:** Melden Sie hier Fehler, die Sie in diesem Buch gefunden haben! Hier können Sie auch nachlesen, welche Fehler andere nach Drucklegung gefunden haben.
- ▶ **Mailingliste:** Tragen Sie sich hier ein, wenn Sie über alle Diskussionsbeiträge und Neuerungen auf der Website per E-Mail informiert werden wollen.



Der Zugang zu der Leser-Webseite ist durch ein Passwort geschützt. Wie Sie zu Ihrem persönlichen Zugang kommen, steht auf der CD im Unterverzeichnis /Website zum Buch in der Datei Anmeldung.htm.

F Abkürzungsverzeichnis

ACE	Access Control Entry
ACL	Access Control List
AD	Active Directory
ADB	Active Directory Browser
ADO	ActiveX Data Objects
ADO.NET	ActiveX Data Objects .NET
ADODB	ActiveX Data Objects Database
ADOMD	ActiveX Data Objects Multi Dimensional
ADOX	ActiveX Data Objects Extensions
ADS	Active Directory Service
ADSI	Active Directory Service Interface
ADTG	Advanced Data Telegram
AKM	Active Knowledge Module
ANSI	American National Standards Institute
API	Application Programming Interface
AppDomain	Application Domain
APPID	Application Identifier
ASCII	American Standard Code for Information Interchange
ASP	Active Server Pages
ASP.NET	Active Server Pages .NET
AssemblyRef	Assembly Reference
ATL	Active Template Library
AUO	Active User Objects
BIOS	Basic Input/Output System
BLOB	Binary Large Object
BOF	Begin Of File
C#	CSharp
CAS	Code Access Security
CATID	Category Identifier

CBF	Code Behind Forms
CCM	Change and Configuration Management
CCW	COM Callable Wrapper
CD	Compact Disc
CDO	Collaboration Data Objects
CDOEX	CDO 3.0 for Exchange 2000
CDOEXM	CDO for Exchange Management
CDONTS	CDO for NT Server
CDOSYS	CDO System (CDO 2.0 for Windows 2000)
CDOW2K	CDO 2.0 for Windows 2000
CDOWF	CDO Workflow Objects for Microsoft Exchange
CIM	Common Information Model
CIS	COM Internet Services
CLB	Component Load Balancing
CLI	Common Language Infrastructure
CLR	Common Language Runtime
CLS	Common Language Specification
CLSID	Class Identifier
CMIP	Common Management Information Protocol
CN	Common Name
COM	Component Object Model
COM+	Component Object Model Plus
CORBA	Common Object Request Broker Architecture
CR/LF	Carriage Return / Line Feed
CSV	Comma Separated Value
CTS	Common Type System
DACL	Discretionary Access Control List
DAO	Data Access Object
DAP	Directory Access Protocol
DAV	Distributed Authoring and Versioning
DB	Datenbank / Database
DBMS	Datenbank-Managementsystem
DC	Domain Controller oder Domain Component
DCE	Distributed Computing Environment
DCO	Domino Collaboration Objects
DCOM	Distributed Component Object Model
DFS	Distributed File System

DHCP	Dynamic Host Configuration Protocol
DHTML	Dynamic Hypertext Markup Language
DISPID	Dispatch Identifier
DLL	Dynamic Link Library
DML	Data Manipulation Language
DMO	Distributed Management Objects
DMTF	Desktop Management Task Force
DN	Distinguished Name
DNA	Distributed interNet Application Architecture
DNS	Domain Name Service
DOM	Document Object Model
DOS	Disc Operating System
DSN	Data Source Name
DSO	Decision Support Objects
DTC	Design Time Controls <i>oder</i> Distributed Transaction Coordinator
DTD	Document Type Definition
DTS	Data Transformation Service
ECMA	European Computer Manufacturers Association
EJB	Enterprise Java Beans
EOF	End Of File
EOS	End of Stream
ESATE	Exchange Script Agent Test Environment
EXE	Executable (ausführbare Datei)
FCL	.NET Framework Class Library
FCLBCL	Base Class Library
FMTID	Format Identifier
FQDN	Fully Qualified Distinguished Name
FSMO	Flexible Single Master Operation
FSO	File System Object
FTP	File Transfer Protocol
GAC	Global Assembly Cache
GAL	Global Address List <i>oder</i> Global Catalogue
GC	Garbage Collector <i>oder</i> Global Catalogue
GDI	Graphics Device Interface
GPO	Group Policy Objects
GUI	Graphical User Interface

GUID	Global Unique Identifier
HTA	HTML Application
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	HTTP over SSL
ICMP	Internet Control Message Protocol
ID	Identifier
IDE	Integrated Development Environment
IDL	Interface Definition Language
IE	Internet Explorer
IID	Interface Identifier
IIS	Internet Information Server
IL	Intermediation Language
IMDB	In-Memory Database
IO	Input/Output
IP	Internet Protocol
IPC	Interprocess Communication
IPID	Interface Pointer Identifier
IPM	Interpersonal Message
IPX	Internet Packet eXchange
IrdA	Infrared Data Association
IS	Information Store
ISO	International Organization for Standardization
IV	Initialisierungsvektor
J#	JSharp
JUMP	Java User Migration Path to Microsoft .NET
JVM	Java Virtual Machine
LCID	Locale Country Identifier
LDAP	Lightweight Directory Access Protocol
LIBID	Library Identifier
LPC	Local Procedure Call
LRPC	Lightweight Remote Procedure Call
MAPI	Messaging Application Programming Interface
MDAC	Microsoft Data Access Components
MDAIPP	OLE DB Provider for Internet Publishing
MFA	Multi File Assembly
MIDL	Microsoft Interface Definition Language

MIME	Multipurpose Internet Mail Extensions
MINFU	Microsoft Nomenclature Foul-Up
MIT	Mobile Internet Toolkit
MMC	Microsoft Management Console
MMC	Microsoft Management Console
MMIT	Microsoft Mobile Internet Toolkit
MO	Managed Object
MOF	Managed Object Format
MOM	Microsoft Operations Manager
MS	Microsoft
MSDN	Microsoft Developer Network
MSDTC	Microsoft Distributed Transaction Coordinator
MSIL	Microsoft Intermediation Language
MTS	Microsoft Transaction Server
NDR	Network Data Representation
NDS	Novell Directory Service
NetBIOS	NetETwork Basic Input/Output System
NGWS	Next Generation Windows Service
NLB	Network Load Balancing
NNTP	Network News Transfer Protocol
NT	Windows New Technology
NT4	Windows NT Version 4.0
NTFS	New Technology File System
NTLM	NT LAN-Manager
O	Organisation
OAEP	Optimal Asymmetric Encryption Padding
ODBC	Open Database Connectivity
OLAP	On-Line Analytical Processing
OLE	Object Linking and Embedding
OLE DB	Object Linking and Embedding Database
OM	Operations Management
OMG	Object Management Group
OMT	Object Modelling Technique
OO	Objektorientierung / objektorientiert
OO4O	Oracle Objects for OLE
OpCodes	Operation Codes
ORPC	Object Remote Procedure Call

OSI	Open Systems Interconnection
OU	Organizational Unit
PAB	Personal Addressbook
PC	Personal Computer
PDB	Program Database
PDC	Primary Domain Control
PE	Portable Executable
PERL	Practical Extraction and Reporting Language
PGP	Pretty Good Privacy
PHP	Personal Home Page Tools
PICS	Platform for Internet Content Selection
ProgID	Programmatic Identifier
QFE	Quick Fix Engineering
RA	Regulärer Ausdruck
RAD	Rapid Application Development
RAS	Remote Access Service
RCW	Runtime Callable Wrapper
RDN	Relative Distinguished Name
RDO	Remote Data Objects
RDS	Remote Data Service
RFC	Request for Comment
RGB	Rot-Grün-Blau-Farbschema
ROT	Running Objects Table
RPC	Remote Procedure Call
RRAS	Routing and Remote Access Service
SACL	System Access Control List
SCE	Security Configuration Editor
SCM	Service Control Manager
SD	Security Descriptor
SDDL	Security Descriptor Definition Language
SDK	Software Development Kit
SFA	Single File Assembly
SID	Security Identifier
SMS	Systems Management Server
SMTP	Simple Mail Transfer Protocol
SNA	Strongly Named Assembly
SNMP	Simple Network Management Protocol

SOAP	Simple Object Access Protocol
SP	Service Pack
SPX	Sequenced Packet eXchange
SQL	Structured Query Language
SSH	System Scripting Host
SSL	Secure Socket Layer
SSP	Security Support Provider
TCL	Tool Command Language
TCP	Transfer Control Protocol
TDL	Template Definition Language
TOM	Text Object Model
T-SQL	Transaction SQL
TypeLib	Typbibliothek
UCS	Universal Character Set (»Unicode«)
UDA	Universal Data Access
UDL	Universal Data Link
UDP	User Datagram Protocol
UMI	Universal Management Interface
UML	Unified Markup Language
UNC	Universal Naming Convention
UPN	Umgekehrt polnische Notation oder User Principal Name
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
URN	Uniform Resource Name
UserID	User Identifier
UTC	Universal Coordinated Time
UTF	UCS Transformation Format
UUID	Universal Unique Identifier
VB	Visual Basic
VB.NET	Visual Basic .NET
VB6	Visual Basic Version 6.0
VB7	Visual Basic Version 7.0
VBA	Visual Basic for Applications
VBS	Visual Basic Script
VBScript	Visual Basic Script
VES	Virtual Execution System
VOS	Virtual Object System

VSA	Visual Studio for Applications
VTBL	Virtual Table
W3C	World Wide Web Consortium
W3SVC	Webservice
WBEM	Web Based Enterprise Management
WDM	Win32 Driver Model
WDSDL	Web Services Description Language
WINS	Windows Internet Naming Service
WKGUID	Well Known Global Unique Identifier
WMI	Windows Management Instrumentarium
WML	Wireless Markup Language
WQL	WMI Query Language
WSC	Windows Script Component
WSF	Windows Scripting File
WSH	Windows Scripting Host / Windows Script Host
WWW	World Wide Web
WYSIWYG	What You See Is What You Get
XDR	XML-Data Reduced
XML	Extensible Markup Language
XMLDOM	Extensible Markup Language Document Object Model
XMP	Extended Management Packs
XPATH	XML Path Language
XSD	XML Schema Definition
XSL	Extensible Stylesheet Language
XSLT	XSL Transformation

G Literaturverzeichnis

G.1 Gedruckte Literatur

[AHO96] Aho, A.; Ullman, J.D.: Informatik – Datenstrukturen und Konzepte der Abstraktion. Bonn: International Thomson Publishing, 1996.

[BOX98] Box, D.: COM: Microsofts Technologie für komponentenbasierte Softwareentwicklung. Bonn: Addison Wesley Longman, 1998.

[CHA96] Chadwick, D.: Understanding X.500 – The Directory. London, New York: Chapman & Hal, 1996.

[CWO99] o.V.: »Visual Basic ist beliebt« In: Computerwoche, Ausgabe vom 27.10.99.

[EIC99] Eicker, S.; Nietsch, M.: Standards zum objektorientierten Paradigma, Wirtschaftsinformatik, Heft 4/99.

[ELL01] Eller, F.: C# lernen. anfangen, anwenden, verstehen. München: Addison Wesley, 2001.

[GRI98] Griffel, F.: Componentware. Heidelberg: dpunkt-Verlag, 1998.

[GRU00] Gruhn, V.; Thiel, A.: Komponentenmodelle. München: Addison Wesley, 2000.

[HAH98] Hahn, S.: ADSI ASP Programmer's Reference. Birmingham: Wrox Press, 1998.

[OES97] Oesterreich, B.: Objektorientierte Softwareentwicklung. München, Wien: R. Oldenburg Verlag, 1997.

[OXF97] Oxford Dictionary of Computing. Oxford: Oxford University Press, 1997.

[SCH01a] Schwichtenberg, H.: Windows-Scripting, 2. Auflage. München: Addison Wesley, 2001.

[SCH01b] Schwichtenberg, H.: COM-Komponenten-Handbuch. München: Addison Wesley, 2001.

[SCH02a] Schwichtenberg, H.: Webforms – Webprogrammierung mit ASP.NET. München: Addison Wesley, 2002.

[SCH02b] Schwichtenberg, H.: ASP.NET – Das Entwicklerbuch. München: Microsoft Press, 2002.

[STA93] Stallings, W.: SNMP, SNMPv2 and CMIP. The Practical Guide to Network-Management Standards. Bonn: Addison Wesley Longman, 1993.

[WES02] Westphal, R.: .NET kompakt. Heidelberg, Berlin: Spektrum Akad. Verlag, 2002.

[PLA01] Platt, David S.: Die Microsoft .NET-Plattform. Redmond: Microsoft Press, 2001.

G.2 Quellen im Internet

G.2.1 Links

[ADS02]	http://www.microsoft.com/adsi	Shortcut zur offiziellen Microsoft-ADSI-Site
[BAK01]	http://www.ddj.com/articles/2001/0165/0165h/0165h.htm	Interviews von Mark Baker mit Andrew Clinick
[CLI01]	http://msdn.microsoft.com/workshop/languages/clinic/scripting07142000.asp	Andrew Clinick über die Zukunft der Scriptsprachen
[CLI02a]	http://msdn.microsoft.com/net/ecma	Microsoft-Seite zur ECMA-Standardisierung von C# und CLI
[CLI02a]	http://msdn.microsoft.com/net/ecma/whole.asp	CLI Class Library Browser
[COM00]	http://www.com-objekte.de	Deutsches Komponentenverzeichnis

[DDT02]	http://www.datadirect-technologies.com/news/pr/021302.asp	DataDirect Technologies Announces Industry's First Data Connectivity Components for the Microsoft .NET Framework
[DOM00]	http://www.w3.org/DOM/	Informationen zum Document Object Model
[DOT01a]	http://www.microsoft.com/net	Offizielle .NET-Site von Microsoft
[DOT01b]	http://www.gotdotnet.com/	Weitere .NET-Site von Microsoft
[DVM00]	http://www.developmentor.com/soap	SOAP-Site der Firma Developmentor
[DVX01a]	http://www.devx.com/dotnet/	DevX .NET Center
[LDA00a]	http://www.openldap.org/	kostenlose LDAP-Implementierungen
[LDA00b]	http://www.innosoft.com/ldapworld/	LDAP-Site der Firma Innosoft
[LOW01]	http://www.123aspx.com/b1to2changes/default.asp	Liste der Änderungen an der .NET-Klassenbibliothek zwischen Beta1 und Beta2
[MCO00]	http://www.microsoft.com/	Microsoft COM-Site
[MON02]	http://www.go-mono.com	Mono-Projekt: Implementierung der CLI und C# auf Linux
[MSDC]	http://msdn.microsoft.com/downloads/default.asp	Microsoft MSDN Download Center
[MSLIB]	http://msdn.microsoft.com/library/	Online-Version der MSDN Library (englisch)
[MSR01]	http://research.microsoft.com	Homepage von Microsoft Research (MSR)
[ODB00a]	http://www.microsoft.com/data	Microsoft-Site zum Thema Universal Data Access
[PLA02]	http://www.rollthunder.com	COM-Homepage des Buchautors David S. Platt (kostenloser Newsletter)
[PLA00]	http://www.rollthunder.com/news/v2n2.htm	Thunderclap, the Newsletter of Rolling Thunder Computing, Volume 2, Number 2 Winter 2000

[SEC00]	http://www.15seconds.com	Developer Community, viel Beispiel-Code und zahlreiche Komponenten und Tools
[SOA01a]	http://msdn.microsoft.com/xml/general/soapspec.asp	SOAP-Spezifikation auf dem Microsoft-Webserver
[SOA01b]	http://www.w3.org/TR/SOAP/	SOAP-Spezifikation auf dem W3C-Webserver
[VSA01a]	http://msdn.microsoft.com/vstudio/vsip/vsa	VSA-Site von Microsoft
[VSA01b]	http://vsip.summsoft.com/vsa/	VSA-Site von Summit Software
[WSS00]	http://www.windows-scripting.de	Deutsche Windows Scripting-Site
[XML01]	http://msdn.microsoft.com/xml/default.asp	Microsoft XML Developer Center

G.2.2 Requests for Comment (RFCs)

- RFCs** [RFC1738] Berners-Lee, T.: Uniform Resource Locators (URL): Dezember 1994.
- [RFC1777] Yeong, W.; Howes, T.; Kille, S.: Lightweight Directory Access Protocol: März 1995.
- [RFC1778] Yeong, W.; Howes, T.; Kille, S.: The String Representation of Standard Attribute Syntaxes: März 1995.
- [RFC1779] Kille, S.: A String Representation of Distinguished Names: März 1995.
- [RFC1798] Young, A.: Connection-less Lightweight X.500 Directory Access Protocol: Juni 1995.
- [RFC1959] Howes, T.: The LDAP URL Definition: Juni 1996.
- [RFC1960] Howes, T.: The String Representation of LDAP Search Filters: Juni 1996.
- [RFC1823] Howes, T.; Smith, M.: The LDAP Application Program Interface: August 1995.
- [RFC2254] Howes, T.: The String Representation of LDAP Search Filters: Dezember 1997.
- [RFC2255] Howes, T.; Smith, M.: The LDAP URL Format: Dezember 1997.

[RFC2256] Wahl, M.: A Summary of the X.500(96) User Schema for use with LDAPv3: Dezember 1997.

G.2.3 Newsgroups zu .NET

news://msnews.microsoft.com/microsoft.public.dotnet.general

news://msnews.microsoft.com/microsoft.public.dotnet.framework

news://msnews.microsoft.com/microsoft.public.dotnet.languages.vb

news://msnews.microsoft.com/microsoft.public.dotnet.languages.csharp

news://msnews.microsoft.com/microsoft.public.de.german.

entwickler.dotnet.framework

news://msnews.microsoft.com/microsoft.public.de.german.

entwickler.dotnet.vb

G.2.4 Newsgroups zu speziellen COM-Komponenten

news://msnews.microsoft.com/microsoft.public.adsi.general

news://msnews.microsoft.com/microsoft.public.wbem

Index

.NET 815, 821
 Anwendung 819, 826
 Disassembler 61
 Framework 67
 Garbage Collection 829
 Laufzeitumgebung 826
 Meta-Daten 826
 Sicherheit 829
.NET Compact Framework 816
.NET Data Provider 816
.NET Enterprise Services 816
.NET Framework 19, 21, 34, 67, 816,
 832, 835, 842, 851
 vs. COM 826
.NET Framework Class Library siehe
 Framework Class Library
.NET Framework SDK siehe
 Framework SDK
.NET Runtime 818
.NET-Strategie 19
.sln 84
.vbproj 84
/bin 84
<Param> 99
<returns> 99
<Summary> 99

A

Abfrage, WQL 794
Ablaufumgebung, BCL-Buch 915
Abmelden 808
abstract 40, 104
Access 357, 359
Access Control Entry 873
Access Control List 873

Active Directory 22, 518, 521, 525, 859,
 865, 876, 878, 883
 Klasse 521
 Objektmodell 520
 Suche 541
Active Scripting 845
Active Server Pages 45
Active Server Pages.NET siehe ASP.NET
Active Template Library
 siehe ATL
ActiveScriptEventConsumer 897
ActiveX Data Objects 349, 353, 505
ActiveX Data Objects.NET siehe
 ADO.NET
Add() 189
AddHandler 103, 246, 476f.
AddRange() 189
AddRef() 829
AddressOf 103, 477
Administrator 18
ADO siehe ActiveX Data Objects
ADO+ 349
ADO.NET 22, 42f., 349, 358, 362,
 384, 396, 505, 541, 818
 Visual Studio .NET 399
ADOCCommand 355
ADOConnection 355
ADODataReader 355
ADODatasetCommand 355
ADODB.Connection 506
ADODB.RecordSet 506
ADOPParameter 355
ADOPParameters 355
ADSI 43, 45, 501, 853, 863, 865, 867, 887
 .NET 501, 506, 511
 ADSI-Browser 860, 876

- ADSI-Edit 878
- Architektur 502, 863
- Bindung 506
- Client 864
- COM 506, 511, 514
- Container 514
- Impersonifizierung 507
- Installation 865
- LDAP-Provider 518
- Meta-Objektmodell 867, 869
- Namespace Extension 864
- Pfad 506
- Provider 863, 869
- Query 877
- SchemaBrowser 880
- SDK 866
- Standardschnittstelle 870
- Werkzeug 876
- ADSI Explorer 877
- ADSI-Pfad 517, 522
- ADsSecurity 45
- Advanced Data Tablegram 391
- Agent 852
- Aktuelle Kultur 291
- Alias 111
 - Namespace 839
- Alice und Bob 701
- Anders Heljsberg 829
- Andrew Clinick 65
- Anweisung 97
- Anwendung 179
 - .NET 819
 - beenden 180
 - COM 158
- Anwendungsprotokoll 471
- Anwendungstyp 79
- API 25, 824
 - Reflection 26
 - Win32 25
- Apostroph 98
- AppDomain 43, 743, 819
- Append() 253
- AppendFormat() 253
- Application Center Test 79
- Application Context 818
- Application Domain 43, 136, 185, 819
- Application Domain Host 843
- Application Identifier 158
- Application Programming Interface
 - siehe API
- Application.Run() 613
- Applikation siehe Anwendung
- ArgumentOutOfRangeException 255
- Array 28, 64, 128ff.
 - eindimensional 128
 - mehrdimensional 129
 - redimensionieren 130
 - VB.NET 70
- ArrayList 55, 187, 189, 416
 - absteigend sortieren 194
 - aufsteigend sortieren 194
 - Elemente anhängen 189
 - Elemente einfügen 189
 - Feste Größe 197
 - nur lesen 197
 - Sortierung ändern 194
 - synchronisieren 196
 - unterschiedliche Datentypen eintragen 191
- ASCII 235, 237
- ASP.NET 19, 46, 812, 818f., 837, 849f.
- Assembler 61
- Assemblierung siehe Assembly
- Assembly 23, 26, 88, 92, 729, 819, 836f., 849
 - BCL 48f.
 - erstellen 742
 - kompilieren 73, 76, 92
 - Multi-File 83, 819, 838
 - Private 841, 850
 - Referenz 35, 38, 76, 86, 820
 - Shared 47, 833, 845
 - Signed 846
 - Single-File 83, 819, 846
 - Strongly Named 841f., 848
 - verbreiten 34, 839
 - Versionsnummer 850
- AssemblyAccess 744
- AssemblyBuilder 741, 744
- AssemblyInfo.vb 92
- AssemblyName 743
- Assoziation, WMI 887, 891
- Asymmetrische Verschlüsselung 681

- Asymmetrische Verschlüsselung 687
- ATL 23, 827
- Attribut 28, 31, 820, 834
 - Active Directory 518
 - indiziert 28, 363, 780
 - Notation 919
 - Verzeichnisdienst 855, 857, 874
 - vs. Meta-Attribut 912
- Attribut-Wert-Paar 862
- Aufgabenliste 82, 92
- Aufrufhierarchie 180
- Aufzählungstyp 27, 29, 33, 48, 820, 834
 - in der FCL 17
 - in Schnittstelle 844
- Ausgabefenster 90, 915
- Auswertung, VB.NET 72
- Authenticode 819, 846, 848
- Authentifizierung, WMI 779
- Automatisierungslösung 18
- Autovervollständigen 81
- B**
- Backslash 234
- Backup Domain Controller 519
- Banyan Vines 855
- base 104
- Base Class Library siehe Framework
 - Class Library
- Basisdatentyp 145, 169, 832
- Basisklasse, Object 133
- Basisklassenbibliothek siehe
 - Framework Class Library
- Batch Update 349, 362
- BCL-Buch, Projektmappe 913
- Bedingte Zuweisung 108
- Bedingung 70
- Benutzer 521, 547f., 854, 872, 882
 - angemeldeter 179
 - anlegen 531
 - entfernen 537
 - hinzufügen 537
 - im Active Directory 521
 - löschen 533
 - umbenennen 534
 - verschieben 535
- Benutzergruppe siehe Gruppe 872
- Benutzerkennwort 532
- Benutzerkonto 538
- Benutzername, WMI 779
- Benutzeroberfläche 22, 75
- Benutzerschnittstelle 595
- Benutzersitzung 545f.
- Beta 25
- Betriebssystem 27, 179
 - Version 180
- Bezeichner 35
- Beziehung
 - 1-zu-1 921
 - 1-zu-n 921
 - Kann-Beziehung 921
 - Muss-Beziehung 921
- Big Endian 542, 862
- Binärdatei 223, 237
 - lesen 243
 - schreiben 242
 - Serialisierung 404
- BinaryFormatter 422
- BinaryReader 237, 243, 759
- BinaryWriter 237, 242, 759
- Binden, über eine GUID 522
- Binding 604
- BindingFlags 725
- Bindung
 - Active Directory 522
 - ADSI 506
- BIOS 882
- Bit 69
- BitArray 187, 201
 - verwenden 202
- BitConverter 167
- Bitmap 662, 669
 - erzeugen 669
 - speichern 675
- BizTalk Server 79
- Blatt 869
- Blatt siehe Leaf
- Blatt-Klasse 506
- Blend 645
- BLOB, Serialisierung 405
- bool 146
- Boolean 146
- Boot-Konfiguration 882

Borland Delphi 829
 Bottom-Up-Syntax 862
 Box, Don 65
 Boxing 107, 821, 849
 break 119
 Browser 45
 Brush 640, 642
 Build Number 163
 Button 602
 Byte 145, 406
 Bytefolge 237, 401, 406

C

C# 67, 80, 825, 829, 835
 Datentyp 144
 C++ 67f., 829, 835
 Calendar 274
 Call by Reference 70
 Call by Value 70
 Callout 918
 camelCasing 36, 96
 Capture 258, 261
 CaptureCollection 258
 CAS 821
 Code Access Security 821, 829
 CaseInsensitiveComparer 187
 Case-sensitive 96
 Casting 126
 Category Identifier 158
 CCW 821
 CDb1 126
 CD-Laufwerk 244
 CD-ROM zu diesem Buch 923
 Char 69, 146
 CheckBox 601
 CheckedListBox 601
 CHKDSK 883
 CIL 822, 833
 CIM Studio 902
 CIMV2.Printer 805
 CInt 126
 Class 27f., 68, 72, 103, 110
 VB.NET 68
 Class Identifier 157
 Class Viewer 57
 Classic Code 66, 821, 835
 CLI 62, 133, 822, 831, 833
 CLI Class Library 62f.
 Clinick, Andrew 65
 Clipboard 617
 ClipBounds() 640
 Close() 117
 CLR 22, 67, 204, 822, 827, 832ff.
 CLS 65, 69, 133, 144f., 824, 850
 Datentyp 145
 konform 832
 Regel 824
 Sprache 825, 832
 CLS-konform 145
 CN siehe Common Name
 Code
 Abschnitt 81
 Datei 92
 Editor 81
 Code Access Security 821f., 829
 Code Access Security siehe CAS
 Codeblöcke 101
 C# 101
 Visual Basic .NET 101
 COFF 841
 Collection 64, 918
 erzeugen 207
 CollectionBase 187
 CollectionsUtil 220
 ColorBlend 646
 ColorConverter 641
 ColorDialog 601
 ColorTranslator 640
 COM 18, 23, 35, 44, 66, 133, 157, 223, 350,
 820f., 826, 830, 832, 836, 843, 849, 863,
 867
 API 67
 Automation 843, 864
 DLL 863
 GUID 157
 Interoperabilität 832
 Komponente 23, 835, 883
 Moniker 506
 Sicherheit 895
 WMI 775f.
 COM Callable Wrapper 26, 821
 COM+ 19, 22, 43, 816, 821, 827
 Administration Objects 43
 versus .NET 827

- ComboBox 601
 - Command 353
 - Command Line
 - Ereigniskonsumenten 897
 - Commerce Server 79
 - Common Information Model 881
 - Klasse 890
 - Common Information Model siehe CIM
 - Common Intermediate Language 822, 833
 - Common Intermediate Language siehe CIL
 - Common Language Infrastructure
 - siehe CLI
 - Common Language Runtime siehe CLR
 - Common Language Specification siehe CLS
 - Common Management Information Protocol 881
 - Common Name 861f.
 - Common Object File Format 841
 - Common Type System 27, 825
 - Compact Framework 816
 - Compare() 194
 - Comparer 187
 - Compiler 22, 39, 42, 47, 92, 826
 - VB.NET 74
 - Compileroption, VB.NET 72
 - Component 803, 835
 - Component Object Model 826
 - Composite Control 849
 - ComputeHash() 684
 - Computer 506, 521, 547f., 854, 870, 872f.
 - herunterfahren 873
 - im Active Directory 521
 - Computer-Klasse, WinNT-Provider 868
 - Computerkonto
 - anlegen 540
 - löschen 541
 - ComRuntimeLibrary 49
 - ConfigurationNamingContext 520, 524
 - Connection 353, 358
 - Connection Pooling 358
 - Connection String siehe Verbindungszeichenfolge
 - ConnectionOptions 779
 - Console 75f., 90, 164f.
 - Console Application 75
 - Console Application siehe Konsolenanwendung
 - Const 103
 - ConstructorBuilder 741
 - Container 514, 521, 855, 867, 918
 - ADSI 869
 - auflisten 539
 - im Active Directory 521
 - Container-Klasse 506
 - Containment-Hierarchie 36, 899
 - ADSI 858
 - WMI 887
 - ContextMenu 602
 - Continue 119
 - Control 599, 845
 - ControlBindingCollection 604
 - Convert 169
 - Cookie 751
 - CookieCollection 751
 - Cool 829
 - CounterCreationData 492, 494
 - CounterCreationDataCollection 492, 494
 - CounterSample 493
 - Country 861
 - Crawler 852
 - CryptographicException 692
 - CryptoStream 695
 - Crystal Reports 83
 - CSharp 829
 - CStr 126
 - CTS 144f.
 - CType 126
 - CultureInfo 280, 380
 - CurrentCulture 148, 154
 - CurrentThread 148
 - CurrentUICulture 291
 - Cursor, erstellen 83
 - Custom Control 849
 - Customizing 851
- D**
- DashStyle 654
 - Data Definition Language 898
 - Data Manipulation Language 898
 - Data Provider 349, 351, 353, 357, 816
 - ODBC 351

- OleDb 351
- SQL 350
- DataColumn 353, 363, 384
- DataColumnCollection 363
- DataDirect Technologies 351
- DataGrid 399
 - XML 397
- DataReader 349, 353, 360
- DataRelation 384f., 396
- DataRow 353, 363, 378, 385
- DataSet 349, 353, 361f., 365f., 373, 382, 384, 388, 411, 912
 - XML 390, 393f., 396
 - Zustand 391
- DataSetCommand 362
- DataTable 349, 353, 362f., 365, 388
 - Ereignis 399
- DataGridView 399
- Datei 223, 225, 882
 - anlegen 232
 - geöffnete 545
 - kopieren 233
 - lesen 241, 243
 - löschen 232
 - Rechte 883
 - schreiben 239, 242
 - suchen 226
 - verschieben 245
 - XML 396, 413, 417
- Dateiattribut 244
- Dateiextension 236
- Dateiinhalt 237
- Dateisystem 22, 34, 44, 223, 872
 - überwachen 244
- Dateisystempfad 234
- Dateizugriff 44, 64
- Daten
 - ändern 373, 376
 - anfügen 378
 - lesen 360
 - löschen 380
- Datenabfrage 794, 796
- Datenadapter 365, 374, 376
- Datenbank 22, 356, 358
 - Client/Server 358
 - Serialisierung 405
 - Visual Studio .NET 802
 - Datenbankdatei 357, 359
 - Datenbankserverprozess 357
 - Datenbankverbindung, Designer 83
 - Datenbindung 602
 - einfach 603
 - komplex 604
 - mit Steuerelementen 604
 - Daten-Mitglied 837
 - Datenmodell 350
 - Datenquelle 43
 - Datensatz
 - ändern 376
 - anfügen 378, 380
 - bewegen 362
 - löschen 380
 - Datenströme siehe Stream
 - Datentyp 22, 43, 69f.
 - CLR 388, 781
 - Fließkomma 64
 - VB.NET 69
 - WMI 781, 892
 - Datentypkonvertierung 169
 - DateTime 380
 - DateTimePicker 277, 601
 - umrechnen 277
 - Datumsformat 380
 - DayOfWeek 276
 - DCE 157
 - DCOM 830
 - Konfiguration 883
 - versus .SOAP 828
 - WMI 779
 - DDL 898
 - DE.ITVisions 33, 36
 - Debuggen 82, 92
 - Debugging 44
 - Decimal 64, 69, 146
 - Decrypt() 689
 - DefaultNamingContext 520, 524
 - DefineDynamicAssembly() 743
 - DefineDynamicModule() 744
 - DefineEnum() 744
 - DefineGlobalMethod() 744
 - DefineInitializedData() 744
 - DefineType() 744

- Deklaration 69f.
 - VB.NET 69
- Delegate 103
- DELETE 374
- Delphi 829
- Deployment 34, 839, 841
- Dequeue() 198
- DES 679
- DESCryptoServiceProvider 686
- Deserialisierung 401, 423
 - Datenbank 406
 - XML 414
- Designer 83
 - Visual Studio.NET 837
 - WMI 803
- Desktop 882
- Destruktor 29, 115, 178, 834
- Detaildatensatz 385
- Detailtabelle 384f.
- DFS 883
- DictionaryBase 188
- Dienst 22, 27, 45, 179, 545, 547, 835, 873, 882
 - ADSI 854
 - auflisten 796
 - Dateisystem 552
 - Designer 83
 - IADsService 872
 - IADsServiceOperations 873
 - installieren 440
 - starten 551
 - stoppen 551
 - überwachen 801
 - Überwachung 800
 - Visual Studio .NET 802
- Dienstkontroll-Manager siehe SCM
- DiffGram, ADO.NET 391
- Digitale Signatur 682
- Dim 110
- Direct Control 849
- Directory 223
- Directory Access Protocol 859
- Directory Service siehe
 - Verzeichnisdienst 854
- Directory System Agent 855
- Directory Tree siehe
 - Verzeichnisbaum 854
- Directory User Agent 855
- DirectoryEntry 503f., 506, 508f., 512, 514ff., 527f., 554
- DirectoryInfo 223, 225
- DirectorySearcher 506
- Direktfenster 89
- Disassembler 61
- DisplaySpecifier 880
- Dispose() 117
- Distinguished Name 507, 512, 858, 861
- Distributed Component Object Model
 - siehe DCOM
- Distributed Computing Environment
 - siehe DCE
- Distributed File System siehe DFS
- Distributed Transaction Coordinator 929
- DLL 23, 35
- DLL-Hölle 830
- DML 898
- DNS 751, 772
- do 120
- Document Object Model siehe DOM
- DoEvents 67
- Dokument, XML 83, 390, 397, 410
- Dokumentation 80
 - .NET Framework SDK 57
 - BCL 54f.
 - deutsch 80
 - Visual Studio .NET 54, 80
- DOM 46, 299, 393f.
- Domäne 179, 545, 547, 872
 - im Active Directory 519
- Domänenbestandteil 521
- Domänen-Controller 519
- Domain Name Service siehe DNS
- DomainUpDown 601
- Don Box 65
- DOS 225
 - Muster 246
 - Suchmuster 226
- DOS-Anwendung siehe
 - Konsolenanwendung
- DOS-Fenster 43, 75, 164
- DOTNET siehe .NET
- Double 64, 146

- Drag&Drop 629
 - mit Daten 630
 - mit Steuerelementen 633
- DragEventArgs 630, 632
- DrawArc() 639
- DrawBezier() 639
- DrawEllipse() 639
- DrawIcon() 639
- DrawImage() 639
- DrawLine() 639
- DrawLines() 639
- DrawPath() 639
- DrawPie() 639
- DrawPolygon() 639
- DrawRectangle() 639
- DrawRectangles() 639
- DrawString() 639
- Druckauftrag 546, 873, 883
- Drucker 521, 545, 547, 805, 854, 870, 874, 883
 - im Active Directory 521
- Druckerwarteschlange 546f., 872f., 883
- DSACryptoServiceProvider 686
- DVD-Laufwerk 244
- Dynamic Host Configuration Protocol 927
- Dynamic Object Interface 872
- E**
- ECMA 35, 62f., 133, 822, 831, 834
- ECMAScript 834
- Editor, Visual Studio .NET 81
- Edit-Resume 82
- Eiffel 67
- Eigenschaft 112, 831, 837
- Einfachvererbung 30
- Eingabeunterstützung 38
- Einrücken 81
- Einsprungpunkt 819
- Element, XML 396
- Elterntabelle 384
- E-Mail
 - EmailEvent 800
 - kontrollieren 767
- EmailEvent 800
- Empty 69
- Encrypt() 688
- End Class 110
- Enqueue() 198
- EnsureCapacity() 252
- Enterprise Architect 79, 831
- Enterprise Services 816, 822
- Enterprise Template 79, 831
- Entry Point 819
- EntryWrittenEventArgs 476
- Entwicklungsumgebung 38
- EnumBuilder 742
- Enumeration 27
- Enumeration siehe Aufzählungstyp
- Environment 179
- Equals() 136
- Ereignis 29, 32, 834
 - Dateisystem 245f.
 - WMI 800, 886, 895
- Ereignisabfrage 794, 799ff., 803
- Ereigniskonsument 32, 897, 906
 - permanent 896
- Ereignisprotokoll 22, 44, 469, 882, 897
 - anlegen 473
 - auslesen 470, 797
 - Designer 83
 - Eintrag anlegen 475
 - Liste der Protokolle 472
 - löschen 474
 - überwachen 476
 - Überwachung 800
 - Visual Studio .NET 802
- Ereignistyp, WMI 799
- ERM 831
- Erstellungsdatum 244
- Erstversionsnummer 163
- European Computer Manufactures Association 831
- Event 103
- Event Correlation 886
- Event Filter 906
- Event Registration Tool 905
- Event Viewer 907
- EventArgs 185, 247
- EventBuilder 742
- EventLog 470, 472ff.
- EventLog siehe Ereignisprotokoll
- EventLogEntry 470
- EventLogEntryCollection 470

- EventLogEntryType 475
- EventQuery 801
- EventViewerConsumer 897
- EventWatcher 801
- Exception 71
 - VB.NET 71
- Exchange Server 79, 853, 859
 - Version 2000 876
- Execution Engine 824
- Exit Do 120
- Exit For 118
- Exit Function 112
- Exit Sub 112
- ExportParameters() 688
- Extended Array Library 64
- Extended Numerics Library 64
- Extended WQL 898
- Extensible Markup Language 402
- Extensible Markup Language siehe XML
- Extensible Stylesheet Language 46
- Extensible Stylesheet Language
 - siehe XSL
- extern 105
- F**
- Farbverlauf 643
- FCL 19, 21, 34, 95, 822, 832
 - Dokumentation 54
 - Namespaces 42
 - Source Code 822
 - Standardisierung 62
 - verwenden 38
- Feld 110, 837
- Fernausführung 828
- Fernzugriff, Ereignisprotokoll 472
- Festplatte 245
- Field 31, 353, 820, 832, 837
- FieldBuilder 742
- Fields 362
- FIFO 197
- File 223
- File System Objects siehe FSO
- FileInfo 223
- Fileservice 552, 873
- FileStream 239, 241, 391
- FileSystemEventArgs 247
- FileSystemObject 223
- FileSystemWatcher 245, 247
- FileVersionInfo 478, 489
- FileWebRequest 752, 755
- FileWebResponse 752, 756
- FillEllipse() 639
- FillPie() 639
- FillPolygon() 640
- FillRectangle() 640
- Finalize 177
- Finalize() 116
- Finalizer 115
- FindMembers() 725
- Firewall 828
 - Datentransport 349
- First Use Lazy Registration 845
- First-In-First-Out siehe FIFO
- FixedSize() 197
- Fließkomma 64
- float32 145
- float64 146
- Flush() 640
- FontConverter 641
- FontDialog 601
- For 117
- For Each 122
- foreach 121
- Forest 519
- Form 605
- Formatierung 64
 - Zeichenkette 153
- Formatierungsvorschrift 153
- FoxPro 835
- Fraktale 669
 - Produktionsregeln 670
 - Turtle-Grafik 670
- Framework, Versionsnummer 25
- Framework Class Library 17, 67, 144, 818, 822
- Framework Class Library siehe FCL
- Framework SDK 19, 54, 57, 73, 80
- Framework Software Development Kit 818
- Frank Eller 20
- Freigabe 521, 546, 548, 870, 872, 883
- Friend 103
- FromHDC() 638
- FromHwnd() 638

FromImage() 638
FromXmlString() 688
FSO 44, 223
FTP 821
Füllwerkzeuge 642
Fujitsu Software 831
Fully Qualified Distinguished Name
 siehe Distinguished Name 858
Function 111
Funktion 71
 Überladen 68
 VB.NET 72
Funktionsbibliothek 23

G

GAC 833
Ganzzahlwert 33
Garbage Collection 115, 829
Garbage Collector 22, 43, 174, 178, 822,
 829, 832
GC 174
GDI 637
GDI+ 637
 Features 637
Get 113
GetBytes() 697
GetHashCode() 136, 204
 eigene Implementierung 204
 Standardimplementierung 204
GetILGenerator() 746
GetMembers() 725
GetModules() 731
GetNonZeroBytes() 697
GetResponseStream() 759
Getter 28
GetThumbnailImage() 665
GetType() 134, 191
GetTypes() 731
Global Assembly Cache 47, 77, 849
Global Assembly Cache siehe GAC
Global Unique Identifier 512, 522
Global Unique Identifier siehe GUID
Grafikkarte 882
Graph, Objektgraph 36
Graphical User Interface 882
Graphics 638
Graphics Device Interface 43

Graphics Device Interface siehe GDI
GraphicsPath 648, 650
GregorianCalendar 274
Group 259, 506
group 536
GroupBox 602
GroupCollection 259
Gruppe 521, 547, 872
 anlegen 536
 im Active Directory 521
 löschen 538
 verändern 537
Gruppenzugehörigkeit 538
GUID 34, 157, 830, 839
Gullivers Reisen 862

H

Hailstorm 833, 838
Handles 103
Hardware 882
Hash 136
Hash-Algorithmen 682
Hashcode 681
 erzeugen 682
HashTable 188, 204
Hash-Wert 842
Hauptformular festlegen 613
Hauptversion 163
Heap 32, 842
HebrewCalendar 274
Heljsberg, Anders 829
HelloWorld 74
Hersteller 821
Herunterfahren 808
Hewlett-Packard 831
Hierarchie
 Containment 36, 887
 Namespace 36
 Objekt 36, 887
 Vererbung 36f.
HijriCalendar 274
Hintergrundkompilierung 82, 131
Holger Schwichtenberg 20
Host 843
Host Integration Server 79
Hotfix 883

- HTML 26, 82, 852, 923
 - downloaden 753
 - Editor 79
- HtmlTextWriter 849
- HTTP 44, 64, 821, 852
- HttpRequest 752, 755
- HttpResponse 752, 756
- HybridDictionary 220
- Hype 818
- Hypertext Transfer Protocol siehe HTTP
- I**
 - IADs 506, 513, 868f., 872, 874
 - IADsAccessControlEntry 873
 - IADsAccessControlList 873
 - IADsClass 873
 - IADsComputer 506, 868
 - IADsComputerOperations 873
 - IADsContainer 506, 868f., 872, 874f.
 - IADsFileService 552, 872
 - IADsFileServiceOperations 553
 - IADsFileShare 872
 - IADsGroup 506
 - IADsO 872
 - IADsOpenDSObject 873
 - IADsPrintJobOperations 873
 - IADsPrintQueue 873
 - IADsProperty 874
 - IADsPropertyEntry 874
 - IADsPropertyList 874
 - IADsPropertyValue 874
 - IADsResource 873
 - IADsSecurityDescriptor 873
 - IADsService 550
 - IADsServiceOperations 550, 553
 - IADsSession 873
 - IADsSyntax 874
 - IADsUser 506, 872
 - IBM 832
 - IClonable 141
 - ICollection 208, 211
 - IComparer 194, 208
 - IComponent 835
 - Icon, erstellen 83
 - ICryptoTransform 695
 - Identität 136
 - IDeserializationCallback 423
 - IDictionary 208
 - IDictionaryEnumerator 208
 - IDispatch 23, 826
 - Idle-Prozess 479, 483, 485
 - IEnumerable 208, 211
 - IEnumerator 208, 215
 - eigene Implementierung 215
 - If 122
 - IFormatter 410
 - IHashCodeProvider 204, 208
 - IIf 108
 - IIS 22
 - ILGenerator 742, 746
 - IList 208, 211
 - eigene Implementierung 211
 - Image 40, 662
 - ImageList 601
 - Impersonifizierung 507
 - ADSI 507, 873
 - WMI 779, 895
 - Implementierung, CLI 62, 63
 - Implementierungsvererbung 68
 - Implements 68, 103
 - import 39
 - ImportParameters() 688
 - Imports 39, 78, 88, 103, 132
 - Index 363
 - Indexer 28, 780
 - Indikator siehe Leistungsindikator
 - Inheritance Control 849
 - Inherits 68, 103
 - Initialisierungsvektor 696
 - InitializeComponent() 595
 - Inkonsistenzen, VB6/VB.NET 69
 - INSERT 374
 - Insert() 189, 253, 255
 - InsertRange() 189
 - Install Time Compilation 840
 - InstalledFontCollection 451
 - Installer 442
 - Instanziierung, VB.NET 71
 - Instanz-Member 175
 - int 144f.
 - Int16 145
 - Int32 145f.
 - Int64 145f.
 - Integer 144

Intel Corporation 831
 Intellisense, WMI 776
 Interface 31, 68, 103, 189
 VB.NET 68, 72
 interface 27
 Interface Definition Language 893
 Interface Identifier 157
 Interlocked 589
 Increment() 589
 Intermediation 828f., 833
 Intermediation Language 22, 829, 833
 internal 103
 Internet 821
 Internet Engineering Task Force 859
 Internet Explorer 482, 486f., 860
 Sicherheit 821
 Internet Information Server 865f.
 Internet Information Services 883
 Interoperabilität 44, 158
 IP
 Adresse 797
 Konfiguration 797
 IP Routing 883
 IPAddress 752, 772
 IPEndPoint 752, 772
 IPHostEntry 772
 IRemotingFormatter 410
 IRQ 882
 Is 106, 125
 is 107
 ISE 832
 ISerializable 422
 IsFixedSize 197
 ISO Management Framework 881
 IsReadOnly 197
 IsSynchronized 197
 IUnknown 23, 826
 IV 696
 IXPathNavigable 342
J
 J# 19, 80
 J++ 834
 JapaneseCalendar 274
 Java 68, 829, 833f.
 Java Enterprise Beans 927

Java User Migration Path to Microsoft .NET 834
 Java Virtual Machine 834
 JavaBeans 835
 JavaScript 69
 Join 362
 JScript.NET 19, 80, 834f.
 JSharp 834
 JSharp.NET 80
 JSharp.NET 19
 JulianCalendar 274
 JUMP 834
 Just-in-Time-Compiler 822, 829, 834
K
 Kalenderklassen 274
 Kante, Objektdiagramm 919
 Kardinalität 920
 Kennwort 532
 WMI 779
 Kerberos 895
 Key 696
 Kindtabelle 384
 Klasse 27f., 36, 41, 48f., 68, 70, 109, 146, 834, 837, 921
 Active Directory 518, 521
 ADSI 874
 COM 157
 FCL 775
 gruppieren 34, 839
 in der FCL 17
 Modul-Klasse 30, 41
 Notation 918
 VB.NET 68, 71
 versiegelt 40
 Verzeichnisdienst 857, 873
 WMI 775, 789f., 886, 890
 Klassenbibliothek 17, 21
 .NET 21
 .NET Framework 21
 CLI 62
 Dokument 55
 Klassendefinition 403
 Knoten 855
 Objektdiagramm 918
 KnownColor 641
 Kommandozeilen-Compiler 79

- Kommandozeilenparameter 179
- Kommentar 98, 916
 - C# 98
 - einzeilig 98
 - mehrzeilig 98
 - Visual Basic .NET 98
 - XML 99
- Kommentarseiten erstellen 99
- Kompatibilität, VB6/VB.NET 67, 851
- Komponente 23, 819
 - .NET 83, 819, 829, 834
 - ADSI 863
 - COM 18, 826, 830, 849, 861
 - Definition 835
 - entfernte 45
 - Installation 827
 - Version 827
- Komponentenarchitektur 835
- Komponentenkategorie, COM 158, 796
- Komponentenklasse 83
- Konfigurationscontainer 521
- Konfigurationsdatei 834
- Konfigurationsdaten 43
- Konsolenanwendung 75, 90, 92
- Konstante 29, 834
 - definieren 33
 - in Schnittstelle 844
- Konstantenliste siehe Aufzählungstyp
- Konstruktor 29, 32, 115, 834
 - VB.NET 68, 71
 - WMI 791
- Kontakt 521
 - im Active Directory 521
- Kontext 818
- Konvention 911
 - Formatierung 912
 - Nutzungsbeziehung 919
 - Objektdiagramm 917
 - Sprache 911
- Konvertierung 169
- Kopie 140
 - flach 140
 - tief 141
- KoreanCalendar 275
- Kultur zuweisen 290
- L**
 - Label 600
 - Ländercode 836
 - Ländereinstellung 148, 154
 - Länderkennung siehe LCID
 - Länderstring 279
 - lanmanserver 550, 552
 - Last-In-First-Out siehe LIFO
 - Laufwerk 180, 782
 - Laufwerksbuchstabe 234
 - Laufzeitumgebung 22, 826f., 835
 - Lazy Resource Recovery 832
 - LCID 279
 - LDAP 518, 856, 865
 - Namen 861
 - Query-Syntax 542
 - LDAP-Pfad 527
 - LDAP-Provider 518, 549
 - LDAP-Query 542
 - Leaf 514, 855, 867
 - Leerzeichen, entfernen 156
 - Leistungsdaten, ASP.NET 812
 - Leistungsindikator 22, 27, 490
 - anlegen 494
 - beschreiben 496
 - Designer 83
 - Kategorie 495
 - lesen 493
 - löschen 495
 - Visual Studio .NET 802
 - Library Identifier 158
 - LIFO 197
 - Lightweight Directory Access Protocol
 - siehe LDAP
 - LinearGradientBrush 643
 - LinkLabel 600
 - Linux 838
 - List Members 131
 - ListBox 601
 - ListDictionary 220
 - ListView 601
 - Literal 69
 - Little Endian 542, 862
 - Local Country ID siehe LCID
 - LocaleBuilder 742
 - lock 582

- Log File Ereigniskonsumenten 897
- LogicalDisk 793
- LogicalDiskCollection 793
- Lokalisierung 271, 287, 889
 - Einstellungen festlegen 288
 - Fehler vermeiden 272
 - mit Visual Studio .NET 287
 - Ressourcendatei 293
 - Spracheinstellung 288
- Lotus Notes 859
- M**
- Main()-Methode auslagern 613
- MainMenu 602
- Major Version 163
- Managed Code 47, 66, 501, 816, 834f.
- Managed Execution 836
- Managed Object 776, 881, 886
- Managed Object Format siehe MOF
- Managed Provider 349, 541, 836
- Managed Provider siehe Data Provider
- Management Extension,
 - Visual Studio .NET 802
- Management Strongly Typed Class
 - Generation 789
- Management Strongly Typed Class
 - Generator 791
- ManagementBaseObject 776
- ManagementClass 776, 778,
 - 786, 788f., 886
- ManagementEventWatcher 801
- ManagementObject 776, 781,
 - 786, 788, 795, 886
- ManagementObjectCollection 788, 795,
 - 797, 886
- ManagementObjectSearcher 795, 797
- ManagementPath 779
- ManagementScope 779
- Manifest 819, 836
- ManualResetEvent 588
 - WaitOne() 591
- MarshalByRefObject 600
- MarshalByValueComponent 835
- Marshalling, COM 350
- Masterdatensatz 385
- Mastertabelle 384
- Match 259, 261
- MatchCollection 259
- Math 172
- MD5CryptoServiceProvider 684
- MDI 616
- MdiBase 620
- MDI-Fenster
 - anordnen 616
 - hinzufügen 616
- MdiLayout siehe MDI
- Me 103
- Mehrfachvererbung 30, 68
- Member 56, 109
 - auflisten 82, 131
 - Dokumentation 56
 - Shared 40, 47
- MemberInfo 714, 720
- MemoryStream 406, 414
- Message Queue 598
- Message Queue Service 44, 886
- Message Store 857
- Meta Directory Server 859
- Meta-Attribut 27, 64, 185, 402,
 - 426f., 818, 820, 836
 - Assembly 92
 - vs. Attribut 912
- Metadaten 44, 49, 64, 131, 836f.
- Meta-Directory 859
- MetaFile 662, 669
- Meta-Objekt 512, 778
- Meta-Objektmodell 776, 789
 - ADSI 853, 863, 867
- Meta-Schnittstelle 867
- MethodBuilder 742
- Methode 29, 31, 111, 834, 921
 - Getter 28
 - Notation 921
 - Setter 28
- Methodenaufruf, WMI 781
- MethodRental 742
- MFC 23
- mgmtclassgen.exe 789f.
- Microsoft 832
- Microsoft (Namespace) 42
- Microsoft Access siehe Access
- Microsoft Development Environment 79
- Microsoft Exchange siehe Exchange
 - Server 859

- Microsoft Intermediation Language 22, 829, 833, 837
 - Assembler 61
 - Disassembler 61
 - Microsoft Message Queue Service
 - siehe MSMQ
 - Microsoft Meta Directory Server 859
 - Microsoft MSMQ 886
 - Microsoft Nomenclature Foul-Up 912
 - Microsoft SQL Server *siehe* SQL Server
 - Microsoft Systems Management Server 898
 - Microsoft Transaction Server 821
 - Microsoft Visio 79, 831
 - Microsoft.ComServices 43
 - Microsoft.CSharp 42
 - Microsoft.JScript 42
 - Microsoft.VisualBasic 42, 47, 74
 - Microsoft.VSA 42
 - Microsoft.Win32 21, 42
 - MINFU 837, 912
 - Minor Version 163
 - MMC 899
 - Mobile Internet Designer 837
 - Mobile Internet Toolkit 80, 837
 - MobileControls 837
 - MobileUserControl 849
 - Modul 71, 109, 731
 - .NET 836f.
 - erzeugen 744
 - Liste 483
 - VB.NET 71
 - Visual Basic 30, 41
 - ModuleBuilder 742, 744
 - MOF 893, 905, 909
 - Compiler 909
 - Monash University 832
 - Moniker 506
 - Monitor 575
 - Methoden 575
 - Mono 838
 - MonthCalendar 601
 - MouseEventArgs 630
 - MoveNext() 216
 - mscorlib.dll 824
 - mscorlib 49, 77
 - mscorlib.dll 832
 - MSDN Library 54
 - MsgBox 74, 76
 - MSIL 145, 824, 829, 833
 - MSMQ 19, 44
 - MSXML 46
 - MulticastOption 762
 - Multiple Document Interface *siehe* MDI
 - Multitasking 557
 - Multithreading 45, 557
 - Mustervergleich 257
 - MustInherit 40, 104
 - MustOverride 104
 - Mutex 575
 - Mutual exclusion lock *siehe* Mutex
 - My Services 833, 838
 - MyBase 104
- N**
- Nachrichtenschleife 598
 - Nachschlagewerk 19
 - Namenskonflikt 837
 - Namensraum 888
 - Namensraum *siehe* Namespace
 - Namensraumhierarchie 36
 - NameObjectCollectionBase 221
 - Namespace 21, 34, 733, 839
 - .NET 42
 - Alias 39, 839
 - BCL 49
 - Hierarchie 36
 - importieren 38
 - Microsoft 21
 - System 21
 - WMI 887f.
 - Wurzel 21
 - Namespace-ID 506, 547
 - NameValueCollection 221
 - Native Code 834, 840
 - Native Image 840
 - Native Image Cache 840
 - NativeObject 504, 513
 - NDS 876
 - Nested Type 29
 - NETMODULE 838
 - Netscape 832
 - Communicator 860
 - Directory Server 859

- Netware Directory Service 855
 - Network Library 64
 - NetworkStream 762f.
 - Netzdrucker 805
 - Netzlaufwerk 883
 - Netzmanagement 44
 - Netzwerkfunktion 832
 - Netzwerkkarte 797, 882
 - Netzwerkmanagement 775
 - Netzwerkprotokoll 44
 - Netzwerkverbindung 810, 883
 - Neustart 808
 - New 40, 71, 104, 115, 128
 - new 32
 - Next 118
 - Next Generation Windows Service 840
 - ngen.exe 840
 - NGWS Runtime 824, 840, 849
 - NonSerialized 422, 426
 - Notation
 - Objektdiagramm 917
 - umgekehrt polnische 542
 - Notepad 73
 - Nothing 69
 - NotifyIcon 602
 - NotInheritable 40, 104
 - NotOverridable 104
 - NotPublic 49
 - Novell 518, 853, 855, 865
 - Novell Directory Service 876
 - NT 5.1 820
 - NT Event Log Ereigniskonsumenten 897
 - ntlansmanserver 553
 - Null 69
 - NumericUpDown 601
 - Nutzungsbeziehung, Notation 919
- O**
- Object 32, 133, 146
 - VB.NET 69
 - Object Linking and Embedding Database
 - siehe OLEDB
 - Object Pooling 845
 - Objekt
 - Serialisierung 401
 - sperren 575
 - Verzeichnisdienst 856
 - WMI 781, 793, 886
 - Objektbeziehung 36
 - Objektbindung, WMI 778
 - Objektbrowser 58, 60
 - Objektdiagramm, Notation 917
 - Objekthierarchie 36
 - Serialisierung 411
 - Verzeichnisdienst 858
 - WMI 887
 - Objektidentifikation
 - ADSI 506
 - LDAP 861
 - Verzeichnisdienste 858
 - Objektkatalog 58, 899
 - Objektmenge 43, 920
 - Objektmodell 36
 - Active Directory 520
 - ADSI 867
 - Meta-Objektmodell 776, 867
 - Serialisierung 403, 416
 - System.Management 776
 - WinNT 546
 - Objektorientierung 881
 - VB.NET 66, 68
 - Objektvariable 71, 916
 - ODBC 840
 - Treiber 351
 - Office 2000 894
 - OLEDB 349, 352, 357, 360, 816, 840
 - Provider 351, 359, 505, 541
 - OleDbCommand 353f., 374, 382
 - OleDbCommandBuilder 374, 394
 - OleDbConnection 352ff., 358
 - OleDbDataAdapter 353f., 365, 373
 - OleDbDataReader 353f., 360
 - OleDbTransaction 399
 - OpCodes 833, 840f.
 - Open Database Connectivity,
 - Einstellung 883
 - Open Software Foundation siehe OSF
 - Open Source 838
 - OpenFileDialog 601
 - OpenWave 832
 - Operation Codes 840
 - Operator 72, 105
 - Option Compare 102
 - Option Explicit 69, 102, 118
 - Option Strict 69, 72, 102, 364
 - Orchestrierung 83

- Ordner 223
 - anlegen 229
 - auflisten 225
 - Dateisystem 882
 - kopieren 233
 - löschen 229
 - suchen 226
- Organisation 872
 - ADSI-Schnittstelle 872
 - LDAP 861
- Organisationseinheit 521, 861, 872
 - anlegen 529
 - im Active Directory 521
 - löschen 530
- Organizational Unit siehe Organisationseinheit
- ORM 831
- OSF 157
- out() 915
- Outlook Express 860
- Overloading 68
 - VB.NET 68, 72
- Overloads 104
- Overridable 104
- override 104
- Overrides 104
- P**
- P/Invoke 25
- Pagelet 849
- Panel 602
- Parameter, VB.NET 70
- ParameterBuilder 742
- Partitionen 524
- Pascal 67
- PascalCasing 96
- Path 234
- PathGradientBrush 648
- Pattern 257
- Patterns 260
- PE 836
- Peek() 198
- Pen 640, 653
 - erzeugen 654
- Performance Counter
 - Provider 44, 811, 813
- Performance Counter siehe Leistungsindikator
- Performance Monitor 883
- PerformanceCounter 492f., 496f.
- PerformanceCounterCategory 492, 494f.
- Perl 825
- Persistenz 401
 - Datei 404
 - Datenbank 405
- Pfad
 - ADSI 506
 - LDAP 861
 - WMI 778
- Pfadangabe 235, 912
- Pfadtrennzeichen 234
- PGP 679
- PictureBox 601, 664
- Ping 883
- PInvoke 25
- PixelFormat 663
- Platform Invocation Service 25
- Platform Invoke 841
- Plattform 27
- Plattformunabhängigkeit 234, 829
- Platzhalter 153, 225, 257
 - DOS 226
- Plum Hall 832
- Polymorphismus 68
 - COM 867
- Pop() 198
- Port 859
- Portable Executable 841
- Postfix-Notation 542
- preemptives Multitasking 557
- PreJITting 841
- Preserve 130
- Pretty Good Privacy siehe PGP
- Primary Domain Controller 519
- PrintDialog 602
- PrintPreviewDialog 602
- Private 41, 104
- Private Key 841f.
- Privilege, WMI 895
- Process 478ff., 482f., 486f., 489, 811
- ProcessModule 478, 483
- ProcessModuleCollection 478, 480, 483
- ProcessThread 478, 485
- ProcessThreadCollection 478, 480, 485
- Professional Developers Conference 25
- Profile, CLI 63

Program Database 841
 Programmdatei 489
 Programmiersprache, objektorientiert 68
 Programmierung, deklarativ 836
 Project 7 841
 Projekt 83, 92
 BCL-Buch 913
 lokal 85
 Visual Basic 83, 88, 132
 Visual Studio 83
 Web 85
 Projektelemente, Visual Studio .NET 85
 Projektmappe 83, 913
 erstellen 92
 Visual Studio .NET 86
 Projektoption 89
 Property 28, 31, 70, 113, 508, 512, 780,
 785, 820, 837, 842
 in Schnittstelle 844
 Property Cache 514
 PropertyBuilder 742
 PropertyCollection 504
 PropertyValueCollection 509, 512, 528
 Protected 41, 56, 104
 Protokoll, schreiben 240
 Protokolldatei 897
 Provider
 ADSI 863, 865
 WMI 884
 Prozedur 70
 Prozess 477, 882
 .NET 819
 Liste 480
 Pseudo-Prozess 819
 Remoting 401
 starten 486
 vs. Application Domain 819
 Prozessortyp 829
 Pseudo-Zufallszahlen 172
 Public 41, 104
 Public Key 35, 842
 Verfahren 681, 842
 Public Key Token 842
 Pull-Modell 300
 Push() 198
 Push-Modell 300

Q

Qualifier, WMI 891
 Quellcode 38
 Quelltext-Formatierung 132
 Query, LDAP 877
 Queue 188, 198
 verwenden 198
 Queued Component 845
 Quick Fix Engineering 163

R

RadioButton 601
 Random 172
 RC2 679
 RC2CryptoServiceProvider 686
 Reader 237
 ReadOnly 104, 113
 ReadOnly() 197
 Reboot, Neustart 808
 Rechte, Assembly 836
 RecordSet 349, 353, 360, 362, 912
 RectangleF 640
 ReDim 70, 130
 Redistributable 840
 Redundanz, VB6/VB.NET 69
 ReferenceEquals() 137
 Referenz 836
 Assembly 38, 76
 zirkulär 411, 417, 829
 Referenzidentität 136f.
 Referenz-Typ 144, 821, 834,
 842, 849, 852
 Referenzzählung 829
 Reflection 22, 44, 61, 776, 832, 842
 Reflection Library 64
 Reflection-API 713
 Reflector 60
 regasm.exe 26f.
 RegCol 42
 Regex 258f., 261
 RegexCompilationInfo 259
 RegExp 45
 Region 81
 Regions 131
 Registry 22, 26f., 42, 449, 454, 827, 834,
 866, 883, 911
 Remote-Zugriff 465

- Unterschlüssel öffnen 456
 - Zugriff auf Schlüssel 456
 - RegistryHive 455, 465
 - RegistryKey 449, 454f.
 - Regulärer Ausdruck 45
 - Relative Distinguished Name 512, 515, 858, 862
 - Release Candidate 25
 - Release() 829
 - REM 98
 - Remark siehe REM
 - Remote Data Service 45
 - Remote Procedure Call 157
 - Remoting 22, 45, 401, 828, 832
 - Remove() 189, 253
 - RemoveAt() 189
 - RenamedEventArgs 247
 - Replace() 253
 - Replikation 519, 883
 - Repository, WMI 780, 803, 887
 - Request, Eigenschaften ermitteln 757
 - Request for Comment 936
 - Resource Kit, ADSI 554, 866
 - ResourceManager 293, 295
 - Ressource 854, 870, 873
 - Datei 836
 - Ressourcendatei 272, 280, 293
 - Namensbelegung 294
 - unter Windows 3.1 293
 - ResultPropertyValueCollection 510
 - Return 72
 - return 112
 - Revision 163
 - RFC 936
 - RichTextBox 600
 - Rijndael 679
 - RijndaelManaged 686
 - RNGCryptoServiceProvider 697
 - Robot 852
 - Root 855
 - rootDSE 524
 - Routing and Remote Access Service 930
 - Rows 397
 - RPC siehe Remote Procedure Call 157
 - RSA 679, 689
 - anwenden 689
 - RSACryptoServiceProvider 686, 689, 692
 - RSAParameters 688
 - Rückgabewert 146
 - Runtime, Visual Basic 827
 - Runtime Callable Wrapper 25, 843
 - Runtime Host 819, 843
 - Runtime Infrastructure Library 64
- S**
- Satellite-Assembly 294
 - SaveFileDialog 601
 - SAX 300
 - say() 915
 - SByte 145
 - sbyte 145
 - Schema 512, 521, 548
 - Active Directory 520f.
 - ADSI 870
 - URI 160
 - Verzeichnisdienst 857
 - WinNT 548
 - WMI 886f.
 - XML 83, 350, 391
 - SchemaNameCollection 548
 - SchemaNamingContext 520, 524
 - Schleife 70, 117
 - VB.NET 71
 - Schlüsselattribut 862
 - WMI 891
 - Schlüsselpaar, asymmetrisch 842
 - Schnittstelle 27, 31, 36, 48, 189, 207, 844
 - ADSI 867, 871
 - COM 157
 - in der FCL 17
 - VB.NET 68, 72
 - Schnittstellenvererbung 844
 - Schreibweisen 96
 - SCM 429
 - Screen Scraping 852
 - Script for .NET 845, 851
 - Scripting 845
 - Scripting Runtime Library 43, 223, 489
 - sccrun.dll 223
 - sealed 40, 104
 - SearchResult 510
 - SearchResultCollection 506

- Secure Socket Layer 859
- Security Descriptor 873
- Security Service Provider 895
- Selbstbeschreibung 826
- SELECT 394, 796, 799, 898
- Select Case 123
 - mehrere Auswertungen 124
- SelectQuery 795, 797
- Semikolon 97
- Serialisierer
 - binär 403
 - SOAP 409
 - XML 402
- Serialisierung 46, 401
 - benutzerdefiniert 422, 426
 - binär 403
 - nachträgliche Änderung 421
 - Objektmodell 416
 - SOAP 409
 - XML 64, 410
- Serializable 402
- Server Control 819, 845
- Server Explorer 802f.
- Service 550
- Service Control Manager siehe SCM
- ServiceBase 431
- ServiceController 444
- ServiceControllerStatus 446
- Serviced Component 845
- ServiceInstaller 435
- ServiceProcessInstaller 435
- Services Component 822
- Set 113
- Setter 28
- SHA1 682
- SHA1CryptoServiceProvider 684
- SHA1Managed 682, 684
- SHA256 682
- SHA256Managed 685
- SHA384 682
- SHA384Managed 685
- SHA512 682
- SHA512Managed 685
- Shadows 104
- shallow copy 140
- Shared 41, 104, 110
- Shared Assembly 845, 847
- Shared Member 41, 174
- Shared Name 845
- Shared Source 822, 829
- Shared Source CLI 63
- Short 69
- Short-Circuit-Auswertung, VB.NET 72
- Sicherheit
 - ADSI 872
 - COM 895
 - Dateisystem 883
 - Freigabe 883
 - WMI 895
- Sicherheitseinstellung, Komponente 45
- Sicherheitskonfiguration 832
- Sicherheitssystem 821
- Sicht, ADO.NET 399
- Side-by-Side-Execution 845
- Signatur 72
 - Authenticode 846
- Simple API for XML siehe SAX
- Simple Network Management 881, 883
- Simple Object Access Protocol siehe SOAP
- Single 64, 145
- Site Server 859
- Sitzung 873
- Slash 234
- SmallTalk 67, 69
- Snap-In 880
- SOAP 828, 830, 846, 852
 - Serialisierer 409
 - Serialisierung 402, 409
- SOAPFormatter 422
- SoapFormatter 409
- Socket 762
- Software 882
- Software Development Kit 818, 847, 923
- SolidBrush 642
- Solution siehe Projektmappe
- Sonderordner 180
- Sort() 194
- SortedList 188
- Source Code, CLI 63
- SourceSafe 79
- Spaltenname 363
- Speicher 22, 179
- Speichernutzung 811

- Spezialisierte Collections 220
- Sprache
 - .NET 80
 - .NET-Sprache 23, 822, 832
 - COM 827
- Sprachumstellung 271
- Sprechblase 918
- SQL 374, 382, 898
- SQL Server 79, 351f., 357
- SqlCommand 353
- SqlConnection 353
- SqlDataAdapter 353
- SqlDataReader 353
- Stack 32, 188, 198
 - verwenden 198
- Standardattribut 71f.
- Standardausgabe 164
- Standardeingabe 164
- Standardisierung 831
 - .NET Framework 21, 62
- Standardkultur 290
- Standardwert, VB.NET 70
- Startmenü 882
- Static 41, 104, 110
- StatusBar 600
- Steuerelement 83
 - anordnen 598
 - ASP.NET 845
 - dynamisch erzeugen 625
 - Eingaben ändern 628
 - Eingaben einschränken 627
 - Hierarchie 602
- STMAdmin 44
- Strategie, .NET 19
- Stream 64, 237, 353, 391, 413
- StreamReader 237, 241, 243
- StreamWriter 237, 239, 242, 414
- String 146
- StringBuilder 251
 - mehrfach einfügen 255
 - String erzeugen 253
- StringCollection 221
- StringDictionary 221
- Strom siehe Stream
- Strong Name 88, 819, 847
- Strongly Typed Class Generation 789
- Strongly Typed Class Generator 791
- struct 32, 104
- Structure 27, 32, 104, 145
- Structured Query Language siehe SQL 898
- Struktur 27, 32, 48f., 145, 848
 - in der FCL 17
- Stub 844
- Sub 111
- Suchausdrücke 260
- Suche, Active Directory 541
- Sun 832
- Support Tools 878
- SWbemObject 886
- SWbemObjectSet 886
- switch 124
- Symmetrische Verschlüsselung 680
- Synchronisation von Threads 574
- Synchronized() 197
- SyncLock 71, 581, 589
- Syntaxbeschreibung 874
- System 34, 42f., 47, 75, 92, 832
- System.BitConverter 167
- System.Collections 43, 55, 187, 220
- System.Collections.Specialized 187, 220
- System.ComponentModel 835
- System.Configuration 43
- System.Configuration.Install 442
- System.Console 164
- System.Data 43, 92, 349, 352, 355, 391
- System.Data.ADO 355
- System.Data.Common 352
- System.Data.ODBC 351
- System.Data.Odbc 351
- System.Data.OleDb 351f., 354f., 357
- System.Data.SqlClient 350, 352, 357
- System.Data.SqlXml 351
- System.Diagnostics 27, 44, 469, 477, 490, 811, 832
- System.DirectoryService 43
- System.DirectoryServices 20, 501, 503, 506, 518, 853
- System.dll 93, 832
- System.Double 146
- System.Drawing 39f., 43, 637
 - Hilfsklassen 640
- System.EnterpriseServices 19, 43
- System.Enum 49

System.Environment 179, 181
 System.GC 174
 System.Globalization 44, 271
 System.GUID 157
 System.Int32 146
 System.Int64 146
 System.IO 27, 40, 44, 47, 90, 93, 164, 223f.,
 239, 406, 832
 System.Management 20, 44,
 775, 777, 795
 System.Math 172
 System.Messaging 19, 44
 System.Net 44, 751, 832
 Hilfsklassen 772
 System.Net.Sockets 762
 System.Object 36, 133
 System.Random 172
 System.Reflection 26, 44, 832, 843
 System.Reflection.Emit 741
 System.Resources 271
 System.Runtime 832
 System.Runtime.InteropServices 44
 System.Runtime.Remoting 19, 45
 System.Runtime.Seria-
 lization 45, 401, 409
 System.Security 45, 832
 System.Security. 19
 System.Security.Cryptography 679
 System.ServiceProcess 27, 45, 429, 444
 System.String 148
 System.Text 45, 251
 System.Text.RegularExpressions 45, 251
 System.Threading 45, 557, 832
 System.Timers 45
 System.Type 134
 System.Uri 159f.
 System.UriBuilder 159
 System.ValueType 144, 852
 System.Version 163
 System.Void 146
 System.Web 27, 40, 45
 System.Web.UI 46
 System.Web.UI.MobileControls 837
 System.Windows.Forms 46, 67, 595
 System.WinForms 46
 System.Xml 46f., 77, 92
 System.Xml.dll 93

System.Xml.Schema 299
 System.Xml.Serialisation 46
 System.Xml.Serialization 299, 401
 System.Xml.XPath 46, 299, 341
 System.Xml.Xsl 46, 299
 System32 824, 833
 Systemattribut, WMI 892
 Systembaustein 83
 SystemBrushes 640
 SystemColors 641
 Systemereignis 42
 SystemEvents 449
 SystemIcons 641
 Systeminformation 22
 Systemklassen, WMI 891
 Systemmanagement 44, 775
 Systemmonitor 491, 500
 SystemPens 641
 System-Prozess 480
 Systems Management Server 898
 Systemstart 180
 Systemverzeichnis 180

T

TabControl 602
 Tabelle 353, 356, 385
 dynamisch 388
 Tabelleninhalt 363
 TaiwanCalendar 275
 Taskscheduler 883
 TCP 830
 TCP/IP 22, 44, 64, 859
 TcpClient 762f.
 E-Mails kontrollieren 767
 Whois-Abfrage 764
 TcpListener 763
 TDL 831
 TechEd 65
 Technical Preview 25
 Template Definition Language 831, 848
 TextBox 600
 Textcodierung 45
 Textdatei 223, 237, 834
 lesen 241
 schreiben 239
 TextReader 237, 411
 TextWriter 164, 411

- ThaiBuddhistCalendar 275
 - this 103
 - Thread 45, 148, 476, 478, 558
 - anhalten 562, 568
 - beenden 562
 - fortsetzen 562
 - Liste 485
 - mehrere Threads 563
 - Priorität 572
 - starten 559
 - synchronisieren 574
 - Zeitscheibe freigeben 569
 - ThreadAbortException 562
 - Threading 832
 - ThreadPool 582
 - mit Arbeitsobjekten 587
 - QueueUserWorkItem() 583, 592
 - ThreadPriority 573
 - ThreadStart 558
 - ThreadStateException 563
 - Threadsteuerung 68
 - Thumbnails erstellen 665
 - Timeout.Infinite 569
 - TimeSpan 568
 - TimeZone 185
 - TLBIMP.EXE 843
 - TLBINF32.DLL 843
 - To 125
 - TODO 82
 - ToolBar 602
 - Top-Down-Syntax 862
 - ToString() 135
 - ToXmlString() 688
 - Tracing 44
 - TrackBar 601
 - Transaktion 816, 818, 848
 - ADO.NET 399
 - Transaktionsmonitor 845
 - Transaktionsverwaltung 22
 - Tree 519
 - TreeView 601
 - Trennzeichen 234
 - TripleDES 679
 - TripleDESCryptoServiceProvider 686
 - Try...Catch 71
 - Turbo Pascal 829
 - Typ 27, 34, 48, 839
 - eingebettet 29
 - Export 836
 - in der FCL 17
 - Namensgebung 34, 839
 - Referenz 842
 - Wert 842
 - Typbibliothek 27, 158, 836f., 849
 - Type 134, 185, 410, 413, 714
 - TypeBuilder 742, 745
 - TypeOf 106
 - typeof 714
 - Typhierarchie 36
 - Typinformation 22
 - ermitteln 717
 - Typkonvertierung 72
 - Typsignatur 69
- U**
- UdpClient 763
 - Überarbeitung 163
 - Überladung 406
 - Überwachung, Dateisystem 223, 247
 - uint 145
 - UInt16 145
 - UInt32 145
 - UInt64 145
 - ulong 145
 - Umgebungsvariable 180, 234, 883
 - UML 831
 - Unboxing 821, 849
 - Undo 401
 - ungarische Notation 97
 - Unicode 45
 - Unified Modelling Language 917
 - Uniform Resource Identifier siehe URI
 - Uniform Resource Locator 159, 862
 - Universal Coordinated Time 892
 - Universal Data Access 351
 - Universal Management Interface 931
 - Universal Runtime 824, 849
 - Unmanaged Client 26
 - Unmanaged Code 835, 849
 - unsafe 105
 - unsigned 145
 - Unterobjekt 920
 - Unterroutine 70

- Unterversion 163
- Unterverzeichnis 236
- UPDATE 374
- Upgrade-Wizard 67, 851
- URI, Definition 159
- Uri 159f.
- UriBuilder 159, 162
- URL 159, 821
- Ursprungsinformation 489
- User 506, 525, 548
- User Control 83, 849
- UserController 849
- ushort 145
- Using 39, 103, 132
- V**
- Validation Server Controls 850
- ValidatingEventHandler siehe DOM
- ValidationEventHandler 312
- ValidationType 314
- Value 113, 512
- ValueType 32, 144, 185, 842, 852
- Variable 110
 - VB.NET 69
- Variablendeklaration 70
- Variablenname 916
- Variant, VB.NET 69
- VB.NET siehe Visual Basic .NET
- VB6 71, 76
- VBA 72
 - versus VB.NET 65
- vbc.exe 65, 74
- VBIInstance 908
- VBScript 18, 65, 72, 851
 - versus VB.NET 65
- Verbindung, Datenbank 358
- Verbindungszeichenfolge 357f.
- Vererbung 30, 68
 - in .NET 828
 - visuell 68
- Vererbungsbeziehung 36
- Vererbungshierarchie 36f., 842, 902
 - System.ValueType 144
 - Verzeichnisdienst 858
 - WMI 887
- Vergleich
 - auf Ungleichheit 108
 - mit System.Object 107
 - Objekte 106
 - Werte 105
- Verknüpfung
 - DataSet 362, 384
 - Tabelle 384
- Verschlüsseln 688
- Verschlüsselungstechniken 680
- Version 163
- Version Policy 850
- Versionierung 827, 850
- Versionsinformation 489
- Versionskontrolle 850
- Versionsnummer 35, 850
- Versionsnummerstring 163
- Verwalteter Code siehe Managed Code
- Verwaltungsinstrumentation 884
- Verweis
 - Eigenschaften 88
 - hinzufügen 87
 - Visual Studio .NET 86
- Verzeichnis 854
 - aktuell 180
- Verzeichnis siehe auch Ordner
- Verzeichnisbaum 854
- Verzeichnisdienst 43, 512, 518, 542, 546, 853f., 861
- Verzeichnisdienstklasse 512
- Verzeichnisfreigabe siehe Freigabe
- Verzeichnisklasse 521
- Verzeichnisobjekt 507, 516, 871
- Verzweigungen 122
- View State 850
- virtual 104
- Visio 79, 831
- Visual Basic
 - Definition 911
 - Upgrade-Wizard 67, 851
 - Version 6.0 65, 68, 72, 82
 - Version 7.0 65
 - versus VB.NET 65
- Visual Basic .NET 18, 65, 67, 80, 825, 835, 850, 911
 - Bewertung 72
 - Compiler 65, 74f.
 - Datentyp 144
 - Runtime 71

- Sprachkonstrukte 69
 - versus VB 6.0 66
- Visual Basic for Applications 851
- Visual Basic for Applications siehe VBA
- Visual C++ 67
- Visual C++ siehe C++
- Visual FoxPro siehe FoxPro
- Visual Inheritance 68
- Visual InterDev 58
- Visual SourceSafe 79
- Visual Studio .NET 66, 79, 83, 164, 837
 - ADO.NET 399
 - Dokumentation 54
 - Projekttyp 83
 - Referenz 38, 86
 - WMI 776, 802
- Visual Studio Analyzer 79
- Visual Studio for Applications siehe VSA
- VisualBasic.dll 47, 71, 93
- void 111
- volatile 105
- Vorkompilierung 840
- VSA 42, 851

W

- WaitCallback 583
- WBEM siehe Web Based Enterprise Management 881
- WbemScripting 886
- WDSL 851
- Web Based Enterprise Management 881
- Web Form 83, 803
- Web Server Control 851
- Web Services Description Language
 - siehe WDSL
- Webanwendung 67
- Webbenutzersteuerelement 850
- WebClient 752
- Webcontrol 849, 852
- Webform 67, 852
 - ADO.NET 399
- Web-Projekte 85
- WebRequest 40, 752, 755
 - Anfrage 756
 - Download 759
- WebResponse 752, 756
- Webseite 46

- Webserver 45, 551
- Webservice 852
- Website, zu diesem Buch 923
- Websteuerelement 850
- Well Known-GUID 522
- Werkzeug 57
 - ADSI 876
- Wert 70
 - Rückgabewert 72
- Werteklasse siehe Struktur
- Wertidentität 136
- Wert-Typ 144, 842
- while 121
- Whitespace 258, 264
- Whois-Abfrage 764
- Win32_ComponentCategory 796
- Win32_LogicalDisk 782, 786, 790, 793
- Win32_NetworkAdapterConfiguration 797
- Win32_NetworkConnection 810
- Win32_NTLogEvent 797, 800
- Win32_OperatingSystem 808
- Win32_PerfFormattedData_ASPNET_ASPNETApplications 812
- Win32_PerfRawData_PerfProc_Process 811f.
- Win32_Process 487, 811f.
- Win32_Service 796, 800
- Win32-API 18, 23, 25, 67
- WinCV 57
- Windows .NET 886
- Windows 2000 821, 827
 - Server 545
- Windows 95 27
- Windows 98 27
- Windows 9x 911
- Windows Application 75
- Windows CE 825
- Windows Class Viewer 57
- Windows Explorer 249
- Windows Forms 83, 803
 - ADO.NET 399
- Windows Management Instrumentation
 - Query Language siehe WQL 898
- Windows Management Instrumentation
 - siehe WMI
- Windows ME 27

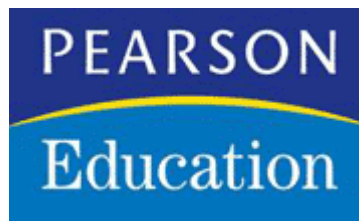
- Windows NT 853
- Windows Scripting 18
- Windows XP 27, 820, 886, 897
- Windows-Anwendung 46, 90
- Windows-Desktop 46
- Windows-Dienst
 - Bestandteile 430
 - Funktionalität hinzufügen 438
 - Installationsfunktionen hinzufügen 443
 - installieren 434
 - Projekt erzeugen 436
 - Rechte 430
 - Registrierung 430
 - starten 433
- Windows-Dienste 429
 - Setup hinzufügen 442
- WindowsForms 67f., 397
- winexe 76
- WinForms siehe WindowsForms
- WinMgmt.exe 884
- WinNT-Provider 518, 545f., 549
- Wireless Markup Language 837
- With 127
- WithEvents 105
- WMI 44f., 888
 - CIM Studio 902
 - Class Explorer 902
 - Ereignis 886, 895, 905
 - Erforschen 898
 - Event Query 897
 - Event Viewer 907
 - Grundlagen 881
 - Instanz 902, 904
 - Klasse 890, 901f.
 - Konfiguration 899
 - Namespace 890, 901
 - Object Browser 899, 902
 - Object Explorer 902
 - Provider 884, 888
 - Query Language 884, 897f., 904
 - Repository 896
 - Schema 887
 - SDK 884
 - Testprogramm 908
 - WMI Query Language 776, 794
 - Extended WQL 898
 - WMI-Pfad 778, 782
 - Wohlgeformtheit siehe DOM
 - WQL 898
 - WQLEventQuery 794
 - WQLObjectQuery 794
 - WrapMode 644
 - Wrapper 18, 23
 - WMI 789
 - WriteOnly 105, 113
 - Writer 237
 - WSHRuntime 44
 - Wurzel siehe Root 855
- X**
 - X.500 859, 861
 - X.700 881
 - XCOPY 841
 - XCOPY-Deployment 34, 828
 - XML 22, 46, 64, 388, 396, 834, 852
 - ADO.NET 350, 390f.
 - Datei 417
 - Dokument 410
 - Editor 79
 - Serialisierer 410, 417
 - Serialisierung 419
 - XML Schema Definition 932
 - XML Validieren 311
 - XmlAttributeAttribute 426
 - XmlAttributeCollection 327
 - XmlConvert 310
 - XML-Data Reduced 927
 - XmlDataDocument 393f.
 - XML-Datei schreiben 323
 - XML-Daten schreiben 318
 - XML-Datentypen konvertieren 310
 - XmlDocument 77, 316, 324, 411
 - Daten ändern 332, 334
 - Daten einfügen 333
 - Nodes auslesen 325
 - XmlElement 411
 - XmlAttribute 311
 - XmlIgnore 426
 - XML-Knoten auslesen 326
 - XML-Kommentare 99

XML-Library 64
XmlNode 77, 316, 324, 411
XmlNode siehe DOM
XmlNodeReader 301, 316
XmlNodeType 303
XmlReader 411, 413
XmlSerializer 402, 410, 413
XML-Strings einlesen 314
XmlTextReader 301, 311, 315
XmlTextWriter 302, 318
XmlValidatingReader 301, 312
XmlWriteMode 391
XmlWriter 318, 411
XmlWriter siehe DOM
XPath 46, 64, 333
XPathDocument 342
XPathExpression 342
XPathNavigator 342
XPathNodeIterator 342f.

XSD 64
 erstellen 83
XSD siehe XML Schema Definition
XSL 46, 345
XSLT 64
XslTransform siehe DOM

Z

Zeichen, VB.NET 69
Zeichenkette 148
 in Array umwandeln 156
 VB.NET 69
Zeichenkettenfunktion 45
Zeiger 842
Zeile, Datenbank 363
Zeitzone 882
Zoomit 859
Zufallszahlen 172
Zuweisung 105
Zyklus
 Objektmodell 36



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 und zugehöriger Materialien und Informationen, einschliesslich der Reproduktion, der Weitergabe, des Weitervertriebs, der Plazierung auf anderen Websites, der Veränderung 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