

COM-Komponenten-Handbuch

Holger Schwichtenberg

COM-Komponenten-Handbuch

Systemprogrammierung und
Scripting mit COM-Komponenten



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 Titelsatz für diese Publikation ist bei
Der Deutschen Bibliothek erhältlich**

© 2001 Addison-Wesley Verlag,
ein Imprint der Pearson Education Deutschland GmbH
Martin-Kollar-Straße 10–12, D-81829 München/Germany
Alle Rechte vorbehalten

Einbandgestaltung: viertel gestaltung, Köln
Korrektur: Astrid Schürmann, Düsseldorf
Sonja Fischer, München
Lektorat: Sylvia Hasselbach, shasselbach@pearson.de

Herstellung: Philipp Burkart, pburkart@pearson.de

Satz: reemers publishing service gmbh, www.reemers.de.
Gesetzt aus der Palatino 9,5/12 pt.

Druck: Bercker Graphischer Betrieb, Kevelaer

Printed in Germany

Die Informationen in diesem Produkt werden ohne
Rücksicht auf einen eventuellen Patentschutz veröf-
fentlicht.

Warennamen werden ohne Gewährleistung der
freien Verwendbarkeit benutzt.

Bei der Zusammenstellung von Texten und Abbil-
dungen wurde mit größter Sorgfalt vorgegangen.
Trotzdem können Fehler nicht vollständig ausge-
schlossen werden.

Verlag, Herausgeber und Autoren können für feh-
lerhafte Angaben und deren Folgen weder eine juri-
stische Verantwortung noch irgendeine Haftung
übernehmen.

Für Verbesserungsvorschläge und Hinweise auf Feh-
ler sind Verlag und Herausgeber dankbar.

Alle Rechte vorbehalten, auch die der fotomechani-
schen Wiedergabe und der Speicherung in elektro-
nischen 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 gleichzei-
tig auch eingetragene Warenzeichen oder sollten
als solche betrachtet werden.

Umwelthinweis:

Dieses Produkt wurde auf chlorfrei gebleichtem
Papier gedruckt.

Die Einschrumpffolie – zum Schutz vor Verschmut-
zung – ist aus umweltverträglichem und recycling-
fähigem PE-Material.

10 9 8 7 6 5 4 3 2 1

04 03 02 01

ISBN 3-8273-1936-6

Inhaltsverzeichnis

Vorwort	21
1 Einführung	25
1.1 Systemzugriff: Wer? Wie?	25
1.2 COM im Überflug	26
1.2.1 COM-Begriffe	26
1.2.2 Eigenschaften von COM	29
1.3 Verfügbare Komponenten	29
1.3.1 Benutzeroberfläche	31
1.3.2 Benutzerverwaltung	31
1.3.3 Systemmanagement	31
1.3.4 Ereignisprotokolle	31
1.3.5 Dienste	32
1.3.6 Registry	32
1.3.7 Zeitplandienst (Scheduler)	32
1.3.8 Verwaltung von Komponenten	32
1.3.9 Dateisystem	32
1.3.10 Netzwerkkommunikation	33
1.3.11 Messaging und Groupware	33
1.3.12 Textdateien	34
1.3.13 Verschlüsselung	34
1.3.14 Datenbanken und andere Datenquellen	34
1.3.15 Microsoft SQL Server (MSQL)	34
1.3.16 Oracle-Datenbanken	34
1.3.17 Verzeichnisdienste	35
1.3.18 Internet Information Server (IIS)	35
1.3.19 Microsoft Exchange Server 5.5	35
1.3.20 Microsoft Exchange Server 2000	36
1.3.21 Lotus Notes	37
1.3.22 SAP R/3	37
1.3.23 Microsoft Office	37
1.4 Der Rest dieses Buchs	38

2	COM-Theorie	39
2.1	Binärstandard	39
2.2	Programmiersprachen für COM	39
2.3	Laufzeitumgebungen	40
2.4	COM-Bausteine	42
2.4.1	Die wichtigsten Bausteine im Kurzüberblick	42
2.4.2	Global Unique Identifier (GUIDs)	45
2.4.3	Moniker	46
2.5	COM-Dienste	47
2.6	COM-Konfigurationsdaten	48
2.6.1	Die Registry als Konfigurationsspeicher	48
2.6.2	XML-Konfigurationsdateien	49
2.7	Komponentenarten	49
2.8	Verpackungsform	52
2.8.1	EXE-Server im Vergleich zu DLL-Server	53
2.8.2	Der Zusammenhang zwischen Komponentenart und Verpackungsform	55
2.8.3	Zusatzinformationen	55
2.9	Registrierung von Komponenten	56
2.10	COM-Klassen	57
2.10.1	Klassen-Identifikation mit CLSIDs	58
2.10.2	Programmatic Identifier	58
2.10.3	Friendly Class Name	59
2.10.4	Klassen in der Registry	59
2.11	COM-Schnittstellen	61
2.11.1	Standardschnittstelle einer COM-Klasse	62
2.11.2	Namensgebung	62
2.11.3	Schnittstellen in der Registry	63
2.11.4	Virtuelle Tabellen	63
2.11.5	Die Grauzone zwischen Klasse und Schnittstelle	64
2.11.6	Mangelnde Selbsterkenntnis bei den Schnittstellen	65
2.11.7	Bullet-and-Stick-Diagramme	67
2.12	Klassen-Mitglieder	67
2.13	Typinformationen	69
2.13.1	Interface Definition Language (IDL)	71
2.13.2	Typbibliotheken	74
2.14	Statischer Aufruf versus Automation	76

Inhaltsverzeichnis

2.15	COM-Standardschnittstellen	78
2.15.1	Statisches Binden mit IUnknown	78
2.15.2	Automation mit IDispatch	80
2.15.3	Duale Schnittstellen	83
2.15.4	IDispatchEx	84
2.16	Distributed COM	84
2.16.1	DCOM-Protokoll	84
2.16.2	DCOM-Installation und -Konfiguration	85
2.16.3	DCOM im Internet	86
2.17	Objektaktivierung	88
2.17.1	Service Control Manager (SCM)	89
2.17.2	Erzeugung neuer Instanzen	89
2.17.3	Zugriff auf bestehende Instanzen	92
2.17.4	Aktivierungsfehler	94
2.18	COM-Anwendungen	95
2.19	COM-Assemblies	96
2.20	COM-Kategorien	100
2.21	Persistenz und Structured Storage	102
2.22	COM-Sicherheit	104
2.22.1	Authentifizierung	105
2.22.2	Identität	105
2.22.3	Impersonifizierung	107
2.22.4	Zugriffsberechtigungen	107
2.23	Active Scripting	109
2.23.1	Active Scripting Hosts	110
2.23.2	Active Scripting Engines	111
2.23.3	COM-Komponenten beim Active Scripting	114
2.24	Windows Script Components (WSCs)	115
2.25	Microsoft Transaction Server (MTS)	116
2.25.1	MTS-Administration	117
2.25.2	Interception	117
2.25.3	Packages	118
2.25.4	Programmierung	119
2.26	COM+	119
2.26.1	Änderungen gegenüber dem MTS	120
2.26.2	Neue Dienste in COM+	121
2.26.3	COM+-Administration	122

Inhaltsverzeichnis

2.27	Objektmodelle in COM-Komponenten	122
2.27.1	Objektorientierte Konzepte in COM	122
2.27.2	Bausteine von COM-Objektmodellen	124
2.27.3	Best Practice für Objektmodelle	125
2.27.4	Meta-Objektmodelle	126
2.28	COM-Werkzeuge	131
2.28.1	Microsoft Registry-Editoren	131
2.28.2	Registry Crawler	132
2.28.3	Regsvr32, SWBregsvr und CliReg	133
2.28.4	COM-Viewer (OLE Viewer)	135
2.28.5	COM-Explorer	139
2.28.6	Dependency Walker	140
2.28.7	Objektkatalog	140
2.28.8	comTLBrowser	143
2.28.9	DCOM-Konfigurationswerkzeug	144
2.28.10	MTS Explorer	147
2.28.11	Snap-In »Komponentendienste«	149
2.28.12	ROT-Viewer	149
2.29	Bewertung von COM	151
2.29.1	Vorteile von COM	151
2.29.2	COM auf anderen Plattformen	152
2.29.3	Unzulänglichkeiten von COM	152
2.30	.NET (»DOTNET«)	156
3	COM-Praxis mit Visual Basic und VBScript	161
3.1	COM in den verschiedenen Visual Basic-Dialekten	161
3.2	Objektvariablen	163
3.3	Instanziierung von COM-Klassen	164
3.3.1	Instanzieren mit CreateObject()	164
3.3.2	Fernaktivierung mit DCOM	165
3.3.3	Instanzieren mit dem NEW-Operator in VB6/VBA	166
3.3.4	New vs. CreateObject()	166
3.3.5	Frühes oder spätes Binden	167
3.4	Zugriff auf bestehende Instanzen	168
3.5	Arbeit mit COM-Objekten	169
3.5.1	Methodenaufrufe	170
3.5.2	Attributzugriffe	170

Inhaltsverzeichnis

3.5.3	Zugriff auf Unterobjekte	170
3.5.4	Arbeit mit Schnittstellen	171
3.5.5	Ereignisse empfangen	173
3.5.6	Tipps zur Objektverwendung	174
3.6	Objektoperationen	176
3.6.1	Wertzuweisung	176
3.6.2	Objektvergleich	176
3.6.3	Typüberprüfung	176
3.6.4	Beispiel	177
3.7	Objektmengen (Collections)	178
3.7.1	Lesezugriff auf Collections	178
3.7.2	Iteration über die Elemente einer Collection	179
3.7.3	Veränderung einer Collection	180
3.7.4	Beispiel	180
3.8	Objektvernichtung	183
3.9	Erzeugung von COM-Komponenten	184
3.9.1	Projekt und Komponente	185
3.9.2	Klassendefinition	188
3.9.3	Schnittstellen und Mitglieder	190
3.9.4	Unterobjekte	196
3.9.5	Selbstreferenzierung	197
3.9.6	Property-Routinen	197
3.9.7	Erzeugung von Collections in VB6/VBA	203
3.9.8	Ereignisse	205
3.9.9	Vererbung	210
3.9.10	Mehrfachschnittstellen	211
4	WSH Runtime Library	217
4.1	Objektmodelle der WSH Runtime Library	218
4.2	WSHNetwork und untergeordnete Klassen	225
4.2.1	Druckerverbindungen verwalten	226
4.2.2	Netzlaufwerke verwalten	227
4.3	WSHShell und untergeordnete Klassen	229
4.3.1	Aktuelles Arbeitsverzeichnis	229
4.3.2	Zeitgesteuerte Dialogboxen	229
4.3.3	Zugriff auf die Registry	230
4.3.4	Schreiben in das Anwendungsprotokoll	233

Inhaltsverzeichnis

4.3.5	Programme ausführen	233
4.3.6	Fernsteuerung von Windows-Fenstern	239
4.3.7	Umgebungsvariablen lesen und bearbeiten	241
4.3.8	Zugriff auf Sonderordner	243
4.3.9	Verknüpfungen erstellen	243
5	Scripting Runtime Library	247
5.1	Die Dictionary-Klasse	248
5.2	File System Objects (FSO)	251
5.2.1	Funktionsüberblick	251
5.2.2	FSO-Objektmodell	252
5.2.3	Lesezugriff auf das Dateisystem	254
5.2.4	Dateisystemoperationen	258
5.3	Arbeit mit Textstreams	260
6	Microsoft Shell Objects	265
6.1	Objektmodell der Shell Objects	266
6.2	Arbeit mit Explorer-Fenstern	268
6.3	Arbeit mit Ordnern und Dateien	271
7	Microsoft Internet Controls	275
7.1	Objektmodell des Webbrowser Controls	276
7.2	Anwendungsbeispiele	276
8	Document Object Model (DOM)	279
8.1	HTML Document Object Model	280
8.1.1	MSHTML-Objektmodell	281
8.1.2	Zugriff auf ein einzelnes Element	283
8.1.3	Arbeit mit Collections	286
8.2	XML Document Object Model (MSXML) Version 4.0	288
8.2.1	MSXML-Objektmodell	290
8.2.2	XML-Dateien für die Skripte	293
8.2.3	Lesezugriff auf XML-Dokumente	293
8.2.4	Erzeugen eines XML-Dokuments	301
8.2.5	Aufruf des XSLT-Processors in MSXML	302

Inhaltsverzeichnis

9	ActiveX Data Objects (ADO)	305
9.1	Von ODBC zu OLE DB	305
9.2	Einführung in die ActiveX Data Objects (ADO)	309
9.3	Beschreibung von Datenverbindungen	312
9.3.1	ODBC-Data Source Names (DSN)	313
9.3.2	OLE DB Data Links	314
9.3.3	Verbindungszeichenfolgen (Connection Strings)	315
9.4	Das ADO-Objektmodell	318
9.5	Tabellenzugriff mit Recordsets	320
9.5.1	Lesezugriff auf Daten	321
9.5.2	Schreibzugriff auf Daten	327
9.5.3	Dynamische Recordsets im Speicher	330
9.5.4	Persistenz von Recordsets	330
9.5.5	Weitere Features der Recordset-Klasse	332
9.6	Befehlsausführung mit der Command-Klasse	333
9.7	Dauerhafte Verbindungen mit der Connection-Klasse	337
9.8	Zugriff auf einzelne Elemente mit der Record-Klasse	340
9.9	Transaktionssteuerung	343
9.10	ADO-Fehlerbehandlung	345
10	Taskscheduler-Komponente	349
10.1	Grundlagen des Taskschedulers	349
10.2	Programmierung	351
10.3	Objektmodell	352
10.4	Liste der Tasks und ihrer Trigger	356
10.5	Erzeugung neuer Tasks	356
10.6	Manueller Start und Stopp	358
10.7	Löschen eines Tasks	359
10.8	Benutzerkontext setzen	359
11	Registry-Zugriff mit RegCol	361
11.1	RegCol-Objektmodell	361
11.2	Lesen der Registry	365
11.3	Änderungen in der Registry	367
12	Zugriff auf die NT-Ereignisprotokolle	369
12.1	EventLog-Zugriff mit STMAAdmin	369
12.2	OLE DB-Provider für das NT-EventLog	373

13 Active Directory Service Interface (ADSI)	375
13.1 Verzeichnisdienste	375
13.1.1 Aufbau eines Verzeichnisbaums	376
13.1.2 Verzeichnisdienstschemata	377
13.1.3 Objektidentifikation	378
13.1.4 Meta-Directories	378
13.1.5 Lightweight Directory Access Protocol (LDAP)	379
13.2 Die ADSI-Grundlagen	382
13.2.1 Architektur	383
13.2.2 Installation	383
13.2.3 Verfügbare ADSI-Provider	384
13.2.4 ADSI Software Development Kit	385
13.3 Das ADSI-Meta-Objektmodell	385
13.3.1 Überblick über die ADSI-Standardschnittstellen	388
13.3.2 Die Meta-Schnittstellen IADs und IADsContainer	392
13.4 ADSI-Werkzeuge	393
13.4.1 Active Directory Service Browser (ADB)	393
13.4.2 ADSI Explorer	394
13.4.3 ADSI Edit	395
13.4.4 ADSISchemaBrowser	397
13.5 Grundlagen der ADSI-Programmierung	397
13.5.1 Objektidentifikation und Bindung	397
13.5.2 Methodenaufrufe und Attributzugriffe	400
13.5.3 Zugriff auf Container-Objekte	403
13.5.4 Instanzenverwaltung	405
13.5.5 ADSI-Sicherheit	407
13.5.6 Schemazugriff	409
13.5.7 ADSI-Queries	412
13.6 ADS Namespaces Container	416
13.7 ADSI-Provider für Windows NT 4.0 (WinNT)	418
13.7.1 WinNT-Objektmodell	419
13.7.2 Identifikation und Bindung	420
13.7.3 Verwaltung von NT-Domänen	421
13.7.4 Die WinNT-Klasse »Computer«	423
13.7.5 Benutzerkonten verwalten mit der »User«-Klasse	424
13.7.6 Terminalserver-Einstellungen	429
13.7.7 Gruppenverwaltung mit der Group-Klasse	429

Inhaltsverzeichnis

13.7.8	Verwaltung von NT-Diensten mit der »Service«-Klasse	431
13.7.9	Verzeichnisfreigaben verwalten	432
13.7.10	Benutzersitzungen und geöffnete Ressourcen	433
13.7.11	Druckersteuerung	434
13.8	Windows2000 Active Directory-Administration mit ADSI	435
13.8.1	ADS-Objektmodell	436
13.8.2	Identifikation und Bindung	438
13.8.3	Benutzerverwaltung	440
13.8.4	Organisationseinheiten verwalten	443
13.9	Exchange Server-Administration mit ADSI	444
13.9.1	LDAP-Konfiguration	444
13.9.2	Exchange-Objektmodell	445
13.9.3	Identifikation und Bindung	448
13.9.4	Das Exchange-Schema	449
13.9.5	Empfänger auflisten	452
13.9.6	Erzeugung neuer Empfängereinträge	452
13.9.7	Erzeugung von E-Mail-Verteilern	456
13.9.8	Zugriff auf Konfigurationsinformationen	457
13.10	Internet Information Server-Administration mit ADSI	458
13.10.1	IIS-Objektmodell	458
13.10.2	Identifikation und Bindung	460
13.10.3	Arbeit mit virtuellen Webservern	460
13.11	Zusatzkomponenten für ADSI	464
13.11.1	ADSI-Versionsabfrage mit ADsVersion	465
13.11.2	Detaillierte Fehlermeldungen mit ADsError	466
13.11.3	GetObject() durch die Hintertür mit ADsFactory	467
13.11.4	RAS-Konfiguration mit ADsRAS	468
13.11.5	Zugriff auf Sicherheitseinstellungen mit ADsSecurity	470
13.11.6	IADsTools	481
14	Windows Management Instrumentation (WMI)	483
14.1	WMI-Grundlagen	483
14.1.1	WMI-Funktionsumfang	484
14.1.2	WMI in Whistler	485
14.1.3	Installation und Konfiguration	486
14.1.4	WMI-Provider	488
14.1.5	Programmierung mit der WMI-COM-Komponente	489

Inhaltsverzeichnis

14.1.6	Wichtige WMI-Konzepte im Überblick	491
14.1.7	Bindung an Managed Objects	502
14.2	Objektmodell der WMI-Komponente	505
14.2.1	Die Meta-Klasse SWbemObject	508
14.2.2	SWbemServices	509
14.3	Arbeit mit der WMI-Komponente	510
14.3.1	Zugriff auf lokale Managed Objects	510
14.3.2	Zugriff auf Schemainformationen	512
14.3.3	Zugriff auf entfernte Systeme	516
14.3.4	Änderung von Attributwerten	516
14.3.5	Aufruf von Methoden	517
14.3.6	Ausgabe eines ObjectSets	520
14.3.7	WQL-Queries im Detail	521
14.3.8	Semi-synchrone Verarbeitung	524
14.3.9	Asynchrone Aufrufe mit SWbemSink	525
14.4	WMI ODBC-Treiber	527
14.5	ActiveScriptEventConsumer	527
14.5.1	WMI-ADSI-Integration	529
14.6	Weitere WMI-Beispiele	531
14.7	WMI-Werkzeuge	535
14.7.1	WMI Object Browser	535
14.7.2	WMI CIM Studio	537
14.7.3	WMI Event Registration Tool	540
14.7.4	WMI Event Viewer	541
14.7.5	VBInstance	542
14.7.6	WMI-Testprogramm	543
14.7.7	MOF Compiler	543
14.8	Ihre Expedition in die WMI	543
15	COM+ Administration Objects	545
15.1	Basis-Objektmodell	546
15.2	Meta-Objektmodell	548
15.3	Arbeit mit der COMAdminCatalog-Klasse	552
15.4	MTS Administration Objects	556

Inhaltsverzeichnis

16 SQL Server Distributed Management Objects (DMO) 7.0/2000	559
16.1 DMO-Objektmodell	560
16.2 Application-Klasse	565
16.3 SQLServer-Klasse	566
16.4 Anlegen einer neuen Datenbank	567
16.5 Anlegen einer neuen Tabelle	571
16.6 Backups	574
17 DTS-Package Objects	577
18 Collaboration Data Objects (CDO)	579
18.1 CDO-Grundlagen	579
18.1.1 Aufbau von Messaging-Systemen	579
18.1.2 Messaging Application Programming Interface (MAPI)	580
18.1.3 CDO-Versionen	582
18.2 CDO-Objektmodell	583
18.3 CDO- und MAPI-Basiskonzepte	587
18.3.1 MAPI-Identifikationsnummern	587
18.3.2 Gemeinsame Attribute	588
18.3.3 Kleine und große Auflistungen	589
18.3.4 Erweiterbarkeit der Klassen	590
18.3.5 Property-Tags	591
18.4 CDO-Anmeldung	593
18.5 Zugriff auf Infostores	597
18.6 Zugriff auf Ordner	599
18.7 Zugriff auf die Nachrichten	604
18.8 Anlegen neuer Nachrichten in Ordnern	610
18.9 Adressbücher und Adresseinträge	612
18.10 Senden einer E-Mail	613
18.11 Arbeit mit Attachments	615
19 Komponenten für den Exchange Server 2000	619
19.1 Überblick	619
19.1.1 Webstore	619
19.1.2 Active Directory	626
19.1.3 Weitere Komponenten	626
19.1.4 Grundlegende Konzepte des Webstores	627

Inhaltsverzeichnis

19.2	Exchange-Programmierung mit ADO	633
19.2.1	OLE DB-Provider für den Webstore	633
19.2.2	ADO-Klassen	635
19.2.3	SQL-Abfragen auf den Webstore	636
19.2.4	Hilfsroutinen für die ADO-Exchange-Programmierung	639
19.2.5	Zugriff auf Ordner	641
19.2.6	Zugriff auf Nachrichten	648
19.3	CDO 3.0 für Exchange 2000	656
19.3.1	Neue Konzepte in CDO 3.0	657
19.3.2	CDO 3.0-Objektmodell	658
19.3.3	Arbeit mit der Klasse »Message«	659
19.3.4	Arbeit mit der Klasse »Person«	661
19.3.5	Exchange-Objekte erzeugen mit CDO 3.0	662
19.3.6	Senden einer E-Mail	664
19.4	CDO for Exchange Management	666
19.4.1	Empfänger-Typen	667
19.4.2	Objektmodell	668
19.4.3	Anlegen eines Postfachs mit ADSI und CDOEXM	671
19.4.4	Festlegung der Postfacheigenschaften	675
19.4.5	Einrichtung von E-Mail für externe Empfänger	678
20	SMTP-Mailkomponenten	681
20.1	CDONTS für NT4	682
20.2	CDO for Windows2000	684
20.3	JMAIL	688
21	TCP/IP-Kommunikationskomponenten	691
21.1	Microsoft WinSock-Komponente	691
21.2	Internet Transfer-Komponente	695
21.3	ASPInet	700
21.4	Mabry FTPX	702
22	Reguläre Ausdrücke mit RegExp	705
22.1	RegExp-Objektmodell	706
22.1.1	Klasse »RegExp«	706
22.2	Einfacher Mustervergleich mit Test()	708
22.3	RA-Symbole	709

Inhaltsverzeichnis

22.4	Ausführlicher Mustervergleich mit Execute()	712
22.5	Submatches	715
22.6	Musterersetzung mit Replace()	716
23	WindowsScripting-Komponente	719
23.1	Klasse »IniFile«	720
23.2	Klasse »WinNTUser«	720
23.3	Klasse »Util«	722
23.4	Klasse »ADSI«	722
23.5	Klasse »CSV«	722
24	Microsoft Common Dialog Control	725
24.1	Objektmodell	725
24.2	Einfaches Beispiel	727
24.3	Öffnen-Dialog	729
24.4	Speichern-Dialog	731
24.5	Schriftarten-Dialog	732
25	Microsoft Operations Manager-Hilfsobjekte (OnePoint Utility Objects)	735
25.1	OnePoint Utility Objects for Scripting	735
25.1.1	SCManager-Klasse	737
25.1.2	EventLog	738
25.1.3	SystemManager-Klasse	739
25.1.4	INet-Klasse	740
25.1.5	Utility-Klasse	741
25.2	OnePoint Utility Objects for Active Directory 1.0	742
25.3	OnePoint Utility Objects for Exchange	745
25.3.1	Klasse »MailboxAnalyzer«	746
25.3.2	Klasse »PublicFolderAnalyzer«	748
26	Microsoft Office-Komponenten	751
27	Komponentenerforschung	753
27.1	Suche nach Komponenten	753
27.1.1	Lokale Suche	753
27.1.2	Suche bei Microsoft	756

Inhaltsverzeichnis

27.1.3	Suche bei anderen Anbietern	757
27.1.4	Newsgroups	757
27.2	Analyse von Komponenten	758
27.2.1	Installation	758
27.2.2	Typbibliothek betrachten	758
27.2.3	Dokumentation lesen	759
27.2.4	Schrittweise Testcodierung	760
27.2.5	Ausführliches Studium der Dokumentation	761
27.2.6	Implementierung	761
A	Grundlagen objektorientierter Komponentenarchitekturen	763
A.1	Objektorientierung	763
A.1.1	Objekte	763
A.1.2	Schnittstellen (Interfaces)	765
A.1.3	Klassen	766
A.1.4	Vererbung (Inheritance)	768
A.1.5	Beziehungen zwischen Objekten und Klassen	770
A.1.6	Objektmodelle	774
A.1.7	Polymorphismus	776
A.1.8	Dynamische Bindung	777
A.2	Komponentenarchitekturen	778
A.3	Verfügbare Komponentenarchitekturen	781
B	Hinweise zum Buch	783
B.1	Sprachliche Konventionen	783
B.2	Hinweise zur Formatierung des Textes	784
B.3	Querverweise	785
B.4	Konventionen in den Code-Beispielen	786
B.5	Die Umgebung für dieses Buch	788
B.6	Grafische Notation in den Objektdiagrammen	789
B.6.1	Knoten (geometrische Formen)	791
B.6.2	Zusätze	791
B.6.3	Kanten (Verbindungslinien)	792
B.7	Komponenten-Schnellinfo	794

Inhaltsverzeichnis

C	CD-ROM und Website	797
C.1	Der Inhalt der CD-ROM	797
C.2	Die Code-Beispiele auf der CD-ROM	797
C.3	Die Website zu diesem Buch	799
D	Abkürzungsverzeichnis	801
E	Literaturverzeichnis	809
E.1	Gedruckte Literatur	809
E.2	Quellen im Internet	812
E.3	Requests for Comment (RFCs)	820
E.4	Newsgroups	820
E.5	Andere Quellenangaben	821
	Stichwortverzeichnis	823

Vorwort

"Ein vergleichbares Werk, das eine so große Anzahl von Komponenten übersichtlich und detailliert behandelt, ist in deutscher Sprache sonst nicht erhältlich" urteilte das Visual Basic-Fachmagazin *BasicPro* über mein Buch **»Windows- und BackOffice-Scripting«**, das im Juli 2000 erschienen ist.

Dieses Urteil war eine gute Motivation, den COM-Komponenten ein eigenes Buch zu widmen. Das **»COM-Komponenten-Handbuch«** ist ein aktualisiertes und erweitertes Spin-Off von **»Windows- und BackOffice-Scripting«**. Zum einen wurde in diesem Buch der Theorie- und Praxisteil zu COM und der COM-Programmierung mit Visual Basic komplett überarbeitet. Andererseits sind nun noch mehr Komponenten enthalten (z.B. die Microsoft Operations Manager-Objekte). Einige bestehende Kapitel wurden erweitert.

Spin-Off

Parallel dazu gibt es ein zweites Buch: **»Windows Scripting«** ist der direkte Nachfolger von **»Windows- und BackOffice-Scripting«** mit dem Fokus auf der automatisierten Systemadministration mit dem Active Scripting. Dieses Buch habe ich gegenüber dem Originalbuch aktualisiert und in den Themenbereichen Scripting Hosts, Remote Scripting, Scripting-Sicherheit sowie Scripting-Werkzeuge erweitert. Völlig neu ist hier auch ein umfangreiches Kapitel zum neuen .NET-Framework.

Administratoren

Für wen dieses Buch geeignet ist

Während sich **»Windows Scripting«** primär an Administratoren richtet, ist das **»COM-Komponenten-Handbuch«** in erster Linie ein Buch für Software-Entwickler, die nicht den Weg über komplexe Application Programming Interfaces (APIs) gehen wollen, sondern die stattdessen mit komfortablen objektorientierten Software-Komponenten auf vergleichsweise einfache Art auf die Funktionen des Windows-Betriebssystems und diverse Server-Anwendungen wie den Microsoft Exchange Server und den Microsoft SQL Server zugreifen wollen. Dieses Buch beschäftigt sich fast ausschließlich mit nicht-visuellen COM-Komponenten – also nicht mit ActiveX-Steuerelementen zur Entwicklung von Benutzeroberflächen.

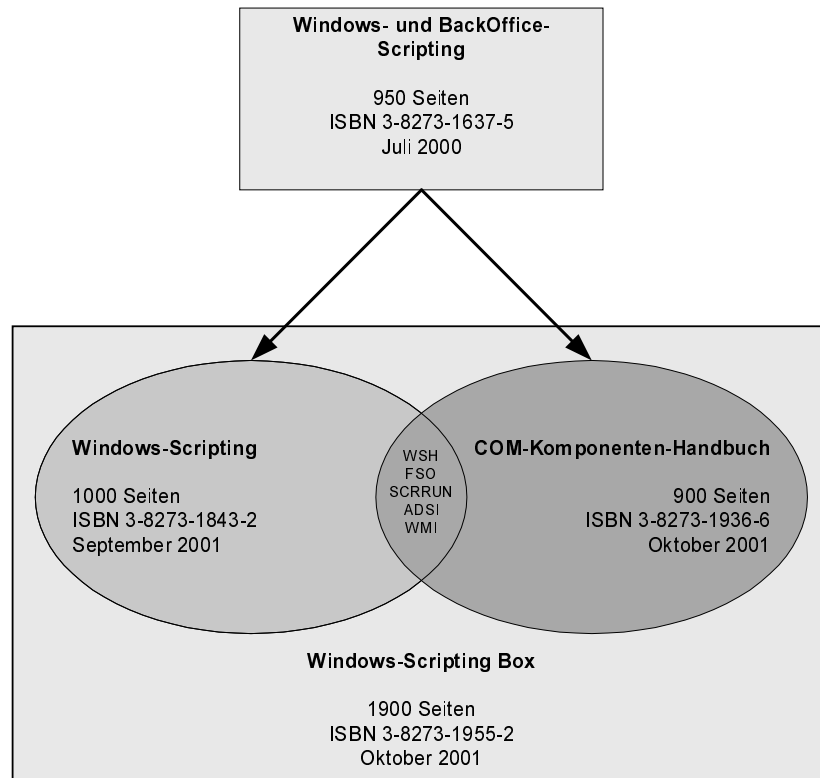
Entwickler

Das "COM-Komponenten-Handbuch" ist kein Buch für Programmier-Einsteiger, sondern setzt fortgeschrittene Kenntnisse in mindestens einem Visual Basic-Dialekt (Visual Basic 6, Visual Basic for Applications oder Visual Basic Script) voraus. Sofern Windows-Administratoren über diese Vorkenntnisse verfügen, werden auch ihnen die in diesem Buch beschriebenen COM-Klassen eine wertvolle Quelle sein, um wiederkehrende und arbeitsaufwändige administrative Aufgaben durch Skripte zu automatisieren.

Administratoren

Vorwort

Bundle Daher ist es nur konsequent, beide Bücher in einem günstigen Paket anzubieten. Dieses Bundle heißt **"Die Windows Scripting-Box"**. Dabei ist das "COM-Komponenten-Handbuch" als Aufbauwissen zu "Windows Scripting" zu verstehen.



Wie Sie dieses Buch lesen sollten

Dieses Buch ist eine Referenz. Man kann dieses Buch auch sequenziell lesen, jedoch ist dies zum Verständnis nicht unbedingt erforderlich.

Kapitel 1 bis 3 Kapitel 1 führt in die Thematik der Systemprogrammierung mit COM ein und gibt einen Überblick über die wichtigsten COM-Komponenten. Kapitel 2 erklärt die Konzepte des Component Object Model (COM). Praktischer wird es in Kapitel 3, das die Nutzung und Erstellung von COM-Komponenten mit Visual Basic behandelt.

Kapitel 4 bis 26 In den Kapiteln 4 bis 26 werden verschiedene COM-Komponenten zur Automatisierung administrativer Aufgaben in den Windows-Betriebssystemen sowie in Produkten der Microsoft BackOffice-Produktfamilie beschrieben. Der Schwerpunkt liegt dabei verständlicherweise auf den von Micro-

Website zum Buch

soft gelieferten Komponenten. Daneben wird eine Auswahl interessanter Komponenten anderer Anbieter vorgestellt.

Mit Sicherheit werden Sie nach der Lektüre dieses Kapitels noch die eine oder andere Funktionalität vermissen. Auf der Suche nach Komponenten hilft Ihnen das in Kapitel 27 vorgestellte Vorgehensmodell.

Kapitel 27

Der Anhang enthält neben einem Abkürzungsverzeichnis und einem sehr ausführlichen Stichwortverzeichnis auch eine kompakte Einführung in die Welt der Objekte und Komponenten. Diese ist als Hilfe gedacht für Leser, die bislang nicht objektorientiert programmiert haben oder in diesem Bereich noch unsicher hinsichtlich der Fachbegriffe sind.

Anhang

Wenn Sie Visual Basic-Entwickler mit Erfahrung in der COM-Programmierung sind, dann können Sie direkt eine der Komponentendokumentationen in den Kapiteln 4 bis 26 aufblättern. Allen anderen lege ich die vorherige Lektüre der Kapitel 2 und 3 als Grundlage zum besseren Verständnis der COM-Architektur nahe.

Website zum Buch

Mehr über die Themen dieses Buches erfahren Sie auch im World Wide Web auf der Site

[HTTP://WWW.COM-KOMPONENTEN.DE](http://www.com-komponenten.de)

Dank

Für die zahlreichen guten Ideen, die fachliche Unterstützung und den liebevollen Rückhalt bei der Erstellung dieses Buches bedanke ich mich bei Barbara Hildebrandt.

Zum Schluss

Ich hoffe, dieses Buch ist Ihnen eine ebenso wertvolle Hilfe bei Ihrer Arbeit wie kurzweilige Lektüre. Über Ihr Feedback freue ich mich. In Anhang C ist beschrieben, wie Sie auf die Leser-Seite zu diesem Buch gelangen.

Holger Schwichtenberg
Essen-Byfang, im September 2001

1 Einführung

1.1 Systemzugriff: Wer? Wie?

Der Zugriff per Programmcode auf die Dienste des Betriebssystems und der Zugriff auf Server- und Desktop-Anwendungen ist gleichermaßen für Entwickler als auch Administratoren wichtig.

Entwickler müssen Informationen über das System gewinnen, um die von Ihnen erstellten Anwendungen auf die jeweilige Umgebung anzupassen. Moderne Windows-Anwendungen nutzen an vielen Stellen vom Betriebssystem bereitgestellte Dienste, wie zum Beispiel den Verzeichnisdienst Active Directory.

Entwickler

Administratoren stehen vor der Aufgabe, immer größer werdende Netzwerke, die immer mehr Konfigurationsmöglichkeiten bieten, in immer kürzerer Zeit zu verwalten und trotzdem immer ständige Verfügbarkeit sicherzustellen. Der Einsatz von Batch-Dateien und Skripten hilft Administratoren, aufwendige und wiederkehrende Aufgaben zu automatisieren.

Administratoren

Application Programming Interfaces

Der Zugriff auf das Windows-Betriebssystem und Windows-Anwendungen war lange Zeit mühsam, weil es nur komplexe Application Programming Interfaces (APIs) gab. Entwicklern standen auf den Programmiersprachen C und C++ basierende APIs wie das Win32-API für den Zugriff auf das Windows-32-Bit-Betriebssystem oder das Messaging Application Programming Interface (MAPI) für den Zugriff auf den Exchange Server zur Verfügung. Diese APIs können zwar grundsätzlich auch aus anderen Programmiersprachen angesprochen werden, doch aufgrund der Komplexität der APIs stehen längst nicht allen Programmiersprachen alle API-Funktionen zur Verfügung.

Win32-API, MAPI

Administratoren, die sich nicht in diese Tiefen der API-Programmierung stürzen wollten, konnten nur mit der sehr einfachen DOS-Batch-Sprache bzw. mit einzelnen parametrisierbaren Kommandozeilenwerkzeugen mühsam Automatisierungslösungen entwickeln.

Component Object Model (COM)

Es wäre untertrieben, einfach nur zu sagen, der Trend, objektorientierte Softwarekomponenten einzusetzen, habe auch vor Windows nicht halt gemacht. Vielmehr hat Microsoft mit seinem Component Object Model (COM) die Verbreitung objektorientierter Softwarekomponenten stark gefördert. Das neue Komponentenmodell .NET (sprich »DOTNET«) wird für einen weiteren Schub sorgen.

Softwarekomponenten

Einführung

Laufzeitumgebungen	Tief im Inneren ist COM nicht weniger komplex als das Win32-API. Das wissen aber heute eigentlich nur noch die C++-Entwickler. Denn Microsoft hat für Visual Basic und verschiedene Skriptsprachen Laufzeitumgebungen geschaffen, die die Arbeit mit COM sehr einfach machen.
Viele Komponenten	COM-Komponenten sind in Windows inzwischen allgegenwärtig: in der Benutzeroberfläche, beim Datenzugriff, bei der Webentwicklung, in den Office- und BackOffice-Anwendungen. Viele Dienste des Betriebssystems und auch viele Anwendungen lassen sich inzwischen über COM-Komponenten ansteuern und sind damit gleichermaßen für Visual Basic-Entwickler auch als für Administratoren, die Skriptsprachen einsetzen, einfach zu programmieren.
Keine Makrorekorder	Die Bewertung »einfach« sollte man im Vergleich zur Programmierung des Win32-APIs oder zur "Hardcore"-COM-Programmierung mit C++ sehen. Auch Visual Basic und die Skriptsprachen verlangen vom Nutzer noch viel Wissen über Programmierung. Einen Makrorekorder, der aus den Mausebewegungen des Administrators in den Management-Konsolen automatisch Programmcode generiert, gibt es zumindest bisher noch nicht.

1.2 COM im Überflug

Microsofts Technologie für die Entwicklung und Nutzung objektorientierter Softwarekomponenten heißt Component Object Model (COM). COM definiert den Aufbau und die Benutzung von Komponenten.

Zumindest innerhalb der Windows-Welt ist COM ein erfolgreicher Ansatz: COM ist existenzieller Bestandteil aller 32-Bit-Windows-Betriebssysteme. COM wird nicht nur von zahlreichen Anwendungen, sondern auch von Windows selbst genutzt. So beruht auch das Active Scripting (alias Windows Scripting) auf COM-Komponenten.



Diese Kapitel erklärt nur die wichtigsten (Ober-)begriffe von COM. Eine detaillierte Einführung in COM erhalten Sie in Kapitel 2 (Theorie) und Kapitel 3 (Praxis). Grundlagen zum Thema objektorientierte Softwarekomponenten erläutert der Anhang A.

Begriffe	1.2.1 COM-Begriffe
	Durch die Fachwelt geistert im Zusammenhang mit COM bzw. anstelle des Begriffs COM eine Reihe von weiteren Namen, wobei die Marketing-Maschinerie von Microsoft selbst einen großen Teil zur Verwirrung beigetragen hat.
Oberbegriff	Häufig stellt sich die Frage, was überhaupt der Oberbegriff ist: COM, DCOM oder ActiveX? Der Autor dieses Buches verwendet stets COM als Oberbegriff. Die Nachwehen der ActiveX-Manie sind aber noch in vielen

Bereichen spürbar und lassen sich nicht so leicht aus der Fachwelt entfernen. So würde der Autor lieber von *COM Scripting* als von *Active Scripting* sprechen, beugt sich aber in diesem Fall der üblichen Sprechweise.

Component Object Model (COM)

Der Begriff Component Object Model (COM) umfasst zwei unterschiedliche Aspekte:

- ▶ **COM-Spezifikation:** Die COM-Spezifikation ist die schriftliche Niederlegung des Aufbaus von COM-Komponenten und der von COM angebotenen Komponentendienste.
- ▶ **COM-Implementierung:** Anders als der Urheber der OMG-Komponentenarchitektur CORBA liefert Microsoft mit der Spezifikation bei COM auch eine passende Implementierung zu dieser Spezifikation. Die COM-Implementierung liegt in Form der sogenannten *COM-Bibliothek* für alle 32-Bit-Windows-Betriebssysteme vor. Das Herzstück der COM-Bibliothek bildet eine DLL, die aus historischen Gründen OLE32.DLL heißt. Diese Bibliothek wird auch *das COM API* genannt.

Spezifikation

Implementierung

Wenn jemand von *einem COM API* spricht, dann meint er damit nicht das in OLE32.DLL implementierte COM API, sondern eine COM-Komponente. Das ist verwirrend. Vermeiden Sie also bitte, eine Komponente so zu nennen.



Object Linking and Embedding (OLE)

OLE ist eine Technologie zur Erzeugung von *Compound Documents*, d.h. von Dokumenten, die aus Bausteinen mehrerer Anwendungen bestehen. Die bereits Anfang der 90er Jahre entwickelte OLE-Technologie ist Basis für COM gewesen. Heute ist OLE in Form der ActiveX-Technologie (s.u.) ein Teil von COM.

OLE

Distributed Component Object Model (DCOM)

DCOM ist die Middleware zu COM. DCOM erweitert COM um die Netzwerkfähigkeit, also die rechnerübergreifende Benutzung der COM-Komponenten in einem verteilten System.

DCOM

ActiveX

Microsoft hat den Begriff ActiveX seit 1996 so sehr in den Vordergrund gestellt, dass er schließlich mit COM gleichgesetzt wurde. Erst im Laufe der Jahre 1998 und 1999 fand eine Rückbesinnung statt, die schließlich zur Einschränkung von ActiveX auf einen Dienst von COM führte. Microsoft definiert in einem Beitrag in der Knowledge Base (»Description of ActiveX Technologies«, [Q154544], Stand August 1999) ActiveX über folgende Technologien:

ActiveX

Einführung

- ▶ ActiveX-Steuerelemente (ActiveX Controls)
- ▶ ActiveX-Dokumente (ActiveX Documents)
- ▶ ActiveX Scripting

Steuerelemente	<i>ActiveX-Steuerelemente</i> (früher OLE-Steuerelemente oder OCX-Steuerelemente) sind visuelle COM-Komponenten zur Gestaltung von Benutzeroberflächen. Diese Steuerelemente werden in <i>ActiveX Control Hosts</i> platziert. Es gibt jedoch eine spezielle Form von Steuerelementen, sogenannte <i>Design Time Controls (DTC)</i> , die, wie der Name schon sagt, nur zur Entwurfszeit Steuerelemente sind. Während der Programmausführung sind sie innerhalb des ActiveX Containers nicht sichtbar. Beispiele dafür sind das Winsock-Steuerelement (<i>MSWinsockLib</i> , <i>MSWINSCK.OCX</i>) und das Skript-Steuerelement (<i>MSScriptControlCtl</i> , <i>MSCRIPT.OCX</i>). Einige Steuerelemente (z.B. das Winsock-Steuerelement) können zudem auch ganz unabhängig von einem ActiveX Container genutzt werden. Diese Steuerelemente lassen sich auch wie normale nichtvisuelle Komponenten instanziiieren und per Programmcode steuern.
Dokumente	<i>ActiveX-Dokumente</i> sind Dokumente, die aus verschiedenen, aus unterschiedlichen Anwendungen stammenden Teildokumenten bestehen können. Dabei können die Teildokumente innerhalb der Host-Anwendung des Gesamtdokuments bearbeitet werden. ActiveX-Dokumente verwenden Sie, wenn Sie eine eingebettete Excel-Tabelle in einem Word-Dokument bearbeiten. Früher hieß diese Technik Object Linking and Embedding (OLE) bzw. Vor-Ort-Aktivierung. Ein <i>ActiveX Document Host</i> ist eine Umgebung, die ActiveX-Dokumente anzeigen kann. Ein Beispiel für einen Document Host ist der Internet Explorer.
Container	ActiveX Container ist der Oberbegriff über <i>ActiveX Control Host</i> und <i>ActiveX Document Host</i> .
Automation	Immer wieder verwenden Microsoft und andere Quellen den Begriff <i>ActiveX</i> synonym zu <i>COM-Automation</i> . Demnach wäre eine automationsfähige Komponente eine ActiveX-Komponente. So erklärt sich zumindest der Name <i>Active Scripting</i> (auch: <i>ActiveX Scripting</i>). Active Scripting basiert auf automationsfähigen Komponenten. Es ist auch nicht korrekt, ActiveX auf eine »Spezifikation zur Benutzung von COM-Objekten über das Internet« (z.B. wie in [RED97], S. 4) zu reduzieren, da ActiveX auch außerhalb des Internets eingesetzt wird (z.B. die ActiveX-Steuerelemente in Windows-Fenstern).
COM+	
COM+	Mit Windows2000 hat Microsoft neue COM-Dienste ausgeliefert, die COM erweitern. Man kann sich darüber streiten, ob COM+ nur diese neuen Dienste umfasst oder als neuer Oberbegriff über alle COM-Technologien anzusehen ist.

Distributed interNet Application Architecture

Die Distributed interNet Application Architecture (DNA) ist ein Softwarearchitekturmodell für mehrschichtige Anwendungen. DNA verwendet COM bzw. COM+ und Web-Scripting-Technologien (Active Server Pages und Dynamic HTML).

DNA

1.2.2 Eigenschaften von COM

Die folgenden Abschnitte beschreiben kurz wesentliche Eigenschaften von COM.

Verwendung in
diesem Buch

Binärstandard

COM umfasst einen Binärstandard für Komponenten und ist daher programmiersprachenunabhängig. Es sind inzwischen viele Sprachen (z.B. C++, C, Java, Visual Basic, Delphi) verfügbar, die COM-Komponenten nutzen und erzeugen können.

Objektorientierung

COM ist ein objektorientiertes Komponentenmodell, das mit Konzepten wie Klassen, Schnittstellen, Attributen und Methoden arbeitet. Die Klassen sind in der Regel in hierarchischen Objektmodellen angeordnet. Für den Nutzer der Komponente (den sogenannten *COM-Client*) ist die Komponente (der sogenannte *COM-Server*) eine Klassenbibliothek.

Integration in Windows

COM ist stark in das Windows-Betriebssystem integriert. COM verwendet zum Beispiel die Windows-Sicherheit und die Windows-Registry als Konfigurationsspeicher.

Verteiltheit

COM bietet mit dem Dienst *Distributed COM (DCOM)* eine transportprotokollneutrale Middleware zur entfernten Nutzung von Komponenten.

1.3 Verfügbare Komponenten

Es sind bereits zahlreiche Komponenten für den Zugriff auf unterschiedliche Betriebssystem- und Anwendungsfunktionen verfügbar. Dieses Kapitel unternimmt einen Streifzug durch die Systemfunktionen, die derzeit durch Komponenten angesprochen werden können. Sofern die Komponenten in diesem Buch besprochen werden, finden Sie Hinweise auf die entsprechenden Kapitelnummern.

Überblick

Einführung

Die Komponenten sind in diesem Kapitel nach verschiedenen Diensten bzw. Anwendungen getrennt und werden mehrfach erwähnt, sofern sie für verschiedene Aufgaben eingesetzt werden können.

Weitere Komponenten

Die hier vorgestellten Komponenten sind nur ein Querschnitt der verfügbaren COM-Komponenten zur Automatisierung von Windows. Darüber hinaus gibt es aber noch weitere, zum Teil sehr spezielle Komponenten, die im Rahmen von Automatisierungslösungen interessant sein könnten.

Hersteller

Microsoft und andere Anbieter

Nicht alle COM-Komponenten stammen von Microsoft selbst: Es gibt inzwischen unzählige Komponenten anderer Anbieter – zum Teil auch als Share- und Freeware. Auch selbst entwickelte COM-Komponenten können unabhängig von der Programmiersprache, in der sie implementiert wurden, verwendet werden. Anwendungen, die direkt komponentenbasiert entwickelt werden, können sehr einfach angesteuert werden.

Funktionen

Kapselung von APIs und eigene Funktionen

Einige Komponenten kapseln den Zugriff auf bestehende Application Programming Interfaces (APIs) von Betriebssystem und Anwendungen. Die Komponenten sind hier Stellvertreter, die die (komplexen) API-Funktionen kapseln. Als positiver Nebeneffekt entsteht dabei in der Regel ein einfaches Objektmodell als Ersatz für komplexe Reihen von API-Aufrufen. Andere Komponenten implementieren eigenständige Funktionalitäten, für die keine weiteren Anwendungen nötig sind.

Auswahl

Vielfältige Möglichkeiten

Die Möglichkeiten des Systemzugriffs mit COM-Komponenten sind vielfältig, deshalb wird der Aufwand eines Entwicklungsprojekts (egal, ob es um ein Skript oder eine »große« Software geht) maßgeblich durch die Auswahl der Komponenten bestimmt.

Make-or-Buy

Eine intensive Recherche nach vorhandenen COM-Komponenten ist ein entscheidender Erfolgsfaktor. Mit der zunehmenden Verbreitung des Komponentengedankens werden Make-or-Buy-Entscheidungen zu einem ständigen Begleiter im Softwareentwicklungsprozess. Entwickler werden sich fragen lassen müssen, ob es notwendig war, eigene Routinen zu entwickeln, anstatt auf dem Markt verfügbare Komponenten zu einer Anwendung zusammenzubauen. Eine unterlassene Internet-Recherche kann dazu führen, dass man sich tagelang mit Problemen beschäftigt, die andere mit wesentlich geringeren Kosten längst gelöst haben.

Kapitel 27 stellt Ihnen ein Vorgehensmodell zur Suche und Erforschung von Komponenten vor.

1.3.1 Benutzeroberfläche

Für die Manipulation der Benutzeroberfläche ist in erster Linie die *WSH Runtime-Komponente* (Kapitel 4) zuständig. Ergänzende Funktionen bieten die *Shell Objects* (Kapitel 6).

**WSH Runtime,
Shell Objects**

Eine Fensterverwaltung ist auch Teil von *MiniOLE* [KNU01]. Das *Common Dialog Control* ermöglicht die Nutzungen von Windows-Standarddialogen (Daten öffnen, Drucker auswählen etc.). Einfache Windows-Bildschirmmasken kann man mit *QuickPrompts* [TOP00] erzeugen.

Hilfreich ist oft der Einsatz der *Microsoft Internet Controls-Komponente* (Kapitel 7) und der *MSHTML-Komponente* (Kapitel 8), die zusammen die Funktionalität des Internet Explorers verkörpern.

**MSHTML, Internet
Controls**

1.3.2 Benutzerverwaltung

Die Benutzerverwaltung unter Windows NT4, Windows2000, Windows XP und Windows.NET (jeweils einschließlich des Active Directory) ermöglicht das *Active Directory Service Interface (ADSI)*, siehe Kapitel 13. Funktionen zur Benutzerverwaltung bietet auch die *ISPSignup-Komponente* aus dem IIS Resource Kit.

ADSI

Für die Verwaltung von Benutzern im Site Server 3.0 gibt es die *Active User Objects (AUO)*.

AUO

1.3.3 Systemmanagement

Die *Windows Management Instrumentation (WMI)* ist ein übergreifender Ansatz zum Zugriff auf alle möglichen Arten von System- und Netzwerkinformationen (Kapitel 14). WMI ermöglicht auch den Zugriff auf Informationen aus Quellen wie Registry, Dateisystem und Ereignisprotokollen, die durch andere Einzelkomponenten bereits abgedeckt sind. Während WMI den Vorteil der Einheitlichkeit bietet, sind die speziellen Komponenten im konkreten Anwendungsfall oft etwas einfacher zu handhaben. Eine bunte Mischung ausgewählter Funktionen bieten auch die in drei Komponenten aufgeteilten *OnePoint Utility Objects* (Kapitel 25). Für den Windows Installer Service gibt es ein *Windows Installer Object Model*.

WMI

1.3.4 Ereignisprotokolle

Die *WSH Runtime-Komponente* (Kapitel 4) von Microsoft kann Einträge nur erzeugen, aber nicht lesen. Diese Komponenten können zudem nur in das Anwendungsprotokoll, nicht aber in andere Protokolle schreiben.

WSH Runtime

Die Freeware-Komponente *STMAAdmin* von Sean McLeod kann Einträge über ein Objektmodell in jedes beliebige NT-Ereignisprotokoll schreiben und auch Einträge lesen (Kapitel 12.1). Die Firma ACES bietet im Rahmen des *ACES WinNT Automation Kit* auch die Möglichkeit, auf ein Ereignispro-

**STMAAdmin, ACES,
OnePoint**

protokoll wie auf eine Datenbanktabelle zuzugreifen (Kapitel 12.2). In den *OnePoint Utility Objects for Scripting*, die mit dem Microsoft Operations Manager (MOM) mitgeliefert werden, gibt es eine Klasse, mit der man ein Ereignisprotokoll in eine Datei sichern und danach leeren kann (Kapitel 25).

1.3.5 Dienste

ADSI, WMI,
OnePoint

NT-Dienste können gestartet, gestoppt, angehalten und fortgesetzt werden. Sie haben einen aktuellen Status und einen Starttyp. Den Zustand auslesen und ändern können:

- ▶ das *Active Directory Service Interface (ADSI)* (Kapitel 13)
- ▶ die *Windows Management Instrumentation (WMI)* (Kapitel 14)
- ▶ die *OnePoint Utility Objects for Scripting* (Kapitel 25)

1.3.6 Registry

Dienste

Wieder bietet die *WSH Runtime-Komponente* (Kapitel 4) nur rudimentäre Zugriffsmöglichkeiten auf die Registry. Als bessere Alternative wird in diesem Buch die Komponente *RegCol* (Kapitel 11) vorgestellt. Rechte auf Registry-Schlüssel können nur mit der *ADSSecurity-Komponente* (Kapitel 13) geändert werden (ADS steht für Active Directory Service).

1.3.7 Zeitplandienst (Scheduler)

Taskscheduler-Komponente

Auch der Windows-Schedule-Dienst kann durch WMI automatisiert werden; einfacher in der Benutzung ist jedoch die *Taskscheduler-Komponente* (Kapitel 10) aus dem Site Server 3.0.

1.3.8 Verwaltung von Komponenten

WMI,
MTS Objects,
COM+ Objects

Die Basiskonfiguration der auf dem System installierten COM-Komponenten ist entweder über direkte Registry-Manipulation oder durch WMI möglich. Für die erweiterten Einstellungen im Zusammenhang mit dem Microsoft Transaction Server und COM+ gibt es eigene Komponenten (Kapitel 15).

Die in Visual Studio 6.0 mitgelieferte *TypeLib Information-Komponente* ermöglicht das Auslesen von COM-Typbibliotheken.

1.3.9 Dateisystem

FSO, Internet
Transfer

Den Zugriff auf das Dateisystem ermöglichen in erster Linie die *File System Objects (FSO)* von Microsoft, die als ein Teil der Scripting Runtime Library in Kapitel 5 vorgestellt werden. Zum Teil ergänzt, zum Teil redundant abgebildet werden die Funktionen von FSO in den *Shell Objects* (Kapitel 6). Zum

Verfügbare Komponenten

Zugriff auf die Rechte in NTFS-Dateisystemen benötigen Sie die *ADsSecurity-Komponente* (Kapitel 13.10.5). Zum Zugriff auf entfernte Dateisysteme via FTP gibt es neben der *Internet-Transfer-Komponente* von Microsoft zahlreiche interessante Lösungen von Drittanbietern (Kapitel 21).

Die in Windows2000 mitgelieferte *DfsCore Komponente* ermöglicht die Verwaltung des Distributed File Systems (DFS). Für die Verwaltung des Quota-Managements gibt es in Windows2000 die *Microsoft Disk Quota-Komponente*.

DfsCore, Quota

Das Komprimieren und Dekomprimieren von Dateien (ZIP/UNZIP) ermöglicht die Komponente *DynaZIP-AX* [DYN01].

DynaZip

1.3.10 Netzwerkkommunikation

TCP/IP-Kommunikation auf Transportebene ermöglicht die *Microsoft Winsock-Komponente* (Kapitel 21.1). Auf Anwendungsebene bietet Microsoft die *Internet Transfer-Komponente* zur HTTP- und FTP-Kommunikation (Kapitel 21.2). Gerade in diesem Bereich gibt es auch viele andere Anbieter, z.B. *ASPInet* (Kapitel 20.3), *Mabry FTPX* (Kapitel 21.4) und die *OnePoint Utility Objects for Scripting* (Kapitel 25).

Winsock, ASPInet, FTPX, OnePoint

Komponenten zur SMTP-Kommunikation (*JMAIL*, *CDONTS*, *CDOSYS*) stellt Kapitel 20 vor.

JMAIL, CDONTS

Für Verwaltung und Aufbau von Remote Access Service-Verbindungen ist die Komponente *Mabry Internet RAS/X* [MAR00] und *MiniOLE* [KNU01] empfehlenswert. Für das Telefon Application Programming Interface (TAPI) in Windows2000 ist dort eine Komponente unter dem Namen *TAPI* enthalten.

Mabry, MiniOLE

1.3.11 Messaging und Groupware

Für die Verarbeitung von elektronischen Nachrichten stellt Microsoft mit den *Collaboration Data Objects (CDO)* inzwischen eine ganze Familie von Komponenten zur Verfügung (Kapitel 18 und 19). Es gibt auf dem Free- und Sharewaremarkt zahlreiche Komponenten für das Handling von SMTP-Nachrichten (z.B. *JMAIL*, Kapitel 20). Einige CDO-Versionen beherrschen darüber hinaus aber die Nachrichtenkommunikation via MAPI. Auch die Firma Lotus bietet mit den *Domino Collaboration Objects (DCO)* für Lotus Notes 5.x bzw. *Notes OLE Objects* für Notes 4.x zwei COM-Komponenten [LOT01a] für den Zugriff auf ihr Groupware-System an.

CDO, Notes Objects

Im Gegensatz zu den großen Brüdern Microsoft Outlook 97/98/2000/2002 kann keine der bisher veröffentlichten Versionen von Outlook Express automatisiert werden, da es kein in einer COM-Komponente implementiertes Objektmodell dafür gibt.

Outlook Express

1.3.12 Textdateien

ADO, MSXML,
TOM

Beliebige Textdateien können mit der *Scripting Runtime-Komponente* (Kapitel 5) bearbeitet werden. Für INI- und CSV-Dateien gibt es u.a. die Komponente *WindowsScripting* (Kapitel 23) und MiniOLE [KNU01]. Die *ActiveX Data Objects (ADO)* können auf strukturierte Textdateien zugreifen (Kapitel 9). XML-strukturierte Textdateien werden mit Hilfe der *MSXML-Komponente* verarbeitet (Kapitel 8). ADO kann Datenbanktabellen in XML-Dateien umwandeln. Das *Text Object Model (TOM)* für den Zugriff auf Rich-Text-Dateien wird in diesem Buch nicht vorgestellt.

1.3.13 Verschlüsselung

Verschlüsselungsfunktionen bietet Microsoft in der *ISPSignup-Komponente* aus dem IIS Resource Kit. Eine Komponente zur Nutzung von Pretty Good Privacy (PGP) unter dem Namen *NSDPGP* gibt es bei [PGP01].

1.3.14 Datenbanken und andere Datenquellen

ADO, ADOX,
ADOMD

Die *ActiveX Data Objects (ADO)* (Kapitel 9) sind allgemeine Komponenten für den Zugriff auf Datenquellen. Dazu gehören neben Datenbanken auch Textdateien und Verzeichnisdienste. Neben der ADO-Kernkomponente gibt es inzwischen zwei Erweiterungen: *Microsoft ADO Extensions for DDL and Security (ADOX)* und *ADO Multi Dimensional (ADOMD)* für die OLAP-Dienste.

ODBCTools

Die Erzeugung von ODBC-Datenquelleneinträgen (DSNs) ermöglichen die *ODBCTools* aus Visual Studio 6.0.

1.3.15 Microsoft SQL Server (MSQL)

ADO, ADOX,
ADOMD, DMO,
DSO

Der Zugriff auf den Inhalt von Microsoft SQL Server-Datenbanken ist mit den *ActiveX Data Objects (ADO)* möglich. Die Konfiguration des SQL Servers und der Datenbanken selbst wird dagegen durch die *SQL Server Distributed Management Objects (SQL-DMO; Kapitel 16)* abgebildet. Für die Administration der OLAP-Dienste (Online Analytical Processing) gibt es die *Decision Support Objects (DSO)*.

DTS

Ein weiterer abgeschlossener Bereich innerhalb des SQL Servers ist der *Data Transformation Service (DTS)*. Dieser kann über die *Microsoft DTSPackage Object Library* (Kapitel 17) automatisiert werden.

1.3.16 Oracle-Datenbanken

OO4O

Microsoft liefert einen OLEDB-Provider für Oracle, so dass auch Oracle-Datenbanken via ADO angesprochen werden können. Darüber hinaus bietet Oracle mit der Komponente *Oracle Objects for OLE (OO4O)* eine eigene, spezialisierte Lösung für den Zugriff von COM-Sprachen auf die eigenen Datenbanken an.

1.3.17 Verzeichnisdienste

Eine der wichtigsten Komponenten ist das *Active Directory Service Interface (ADSI)* (Kapitel 13), mit dem Verzeichnisdienste unterschiedlicher Art (z.B. Active Directory, Exchange Server, Internet Information Server) verwaltet werden können. Neben der Verwaltung des Active Directory unter Windows 2000 gehört dazu auch die Benutzer- und Ressourcenverwaltung unter NT 4.0. Ausgewählte Funktionen zum Umgang mit dem Active Directory bietet auch die Komponente *OnePoint Utility Objects for Active Directory* (Kapitel 25).

ADSI

ADSI kann den Domain Name Service (DNS) in Windows nicht verwalten. Diese Fähigkeit besitzt aber die *ISPSignup-Komponente* aus dem IIS Resource Kit.

1.3.18 Internet Information Server (IIS)

Der IIS wird über seine Metabase konfiguriert, die wie ein Verzeichnisdienst behandelt wird; die Automatisierung erfolgt hier also über das *Active Directory Service Interface (ADSI)* (Kapitel 13.9). Spezielle Konfigurationsfunktionen bietet auch *ISPSignup-Komponente* aus dem IIS Resource Kit.

ADSI

1.3.19 Microsoft Exchange Server 5.5

Für den Zugriff auf den Exchange Server gibt es inzwischen mehrere Lösungen, die zum Teil redundant sind. Da der Exchange- Verzeichnisdienst LDAP-fähig ist, kann auch hier das *Active Directory Service Interface (ADSI)* (Kapitel 13.8) eingesetzt werden.

ADSI

Für die Konfiguration der Rechte auf Exchange-Verzeichnisobjekte benötigen Sie die *ADsSecurity-Komponente* (Kapitel 13.10.5). Eine Ausnahme bilden auch hier die öffentlichen Ordner, deren Rechte Sie nur über die *ACL-Komponente* (ACL steht für *Access Control List*) aus dem Exchange Resource Kit setzen können. Die ebenfalls mit dem Exchange Resource Kit gelieferte *AcctCrt-Komponente* (AcctCrt steht für *Account Control*) bietet dagegen nur Funktionen, die sich auch mit ADSI und der *ADsSecurity-Komponente* erledigen lassen.

ADsSecurity, ACL

Primär auf die Inhalte, also den Message Store, fokussiert die Komponente *Collaboration Data Objects (CDO)* (Kapitel 18 und 19). Zum Teil kann CDO aber auch auf den Verzeichnisdienst zugreifen. Für spezielle, versteckte Nachrichten wie Posteingangsregeln und Event Agents gibt es eigene Automatisierungskomponenten in Form der *Rule-Komponente* (*Microsoft Exchange SDK 5.5 Rules 1.0*) sowie der *ESConfig-Komponente* (*Microsoft Exchange Event Service Config 1.0 Library*). Öffentliche Ordner können nur mit CDO, nicht aber mit ADSI eingerichtet werden. Die Entwicklung von einfachen Workflow-Anwendungen für Exchange 5.5 ermöglichen die *Microsoft Routing Objects*.

CDO, Rule, ESConfig

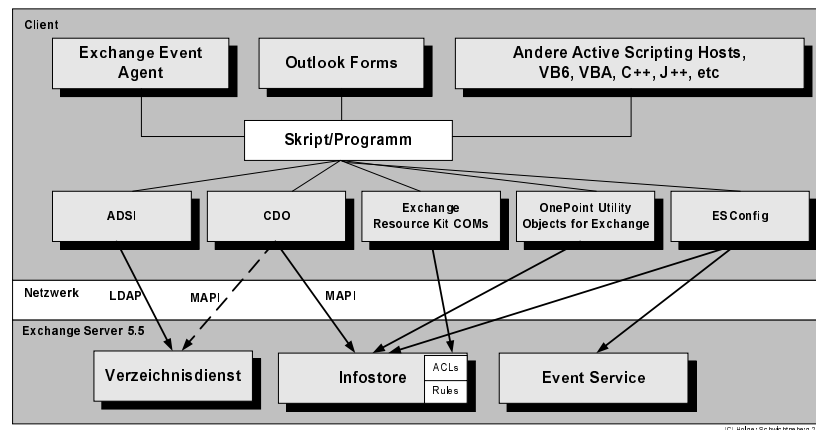
Die *OnePoint Utility Objects for Exchange* (Kapitel 22) bieten eine wichtige Funktion, die man in allen anderen Microsoft-Automatisierungskomponenten vergeblich sucht: Mit dieser Komponente ist man in der Lage zu ermit-

OnePoint

Einführung

teilen, wie groß die einzelnen Postfächer und öffentlichen Ordner auf einem Exchange Server sind.

Abbildung 1.1:
Komponenten für
die Automatisierung
des Exchange
Servers 5.5

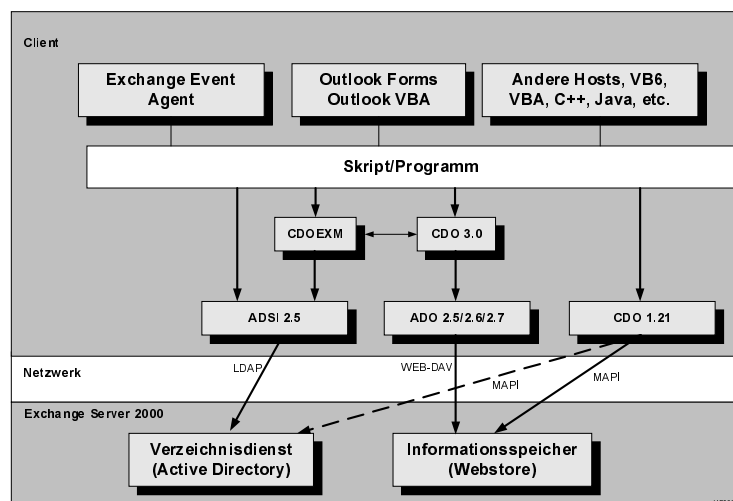


1.3.20 Microsoft Exchange Server 2000

**CDO 3.0,
ADSI, ADO,
CDOEXM**

An dieser Aufteilung der Automatisierungsschnittstellen hat sich mit Exchange2000 einiges geändert. Die *Collaboration Data Objects 3.0* haben sich gegenüber den CDO-Versionen für Exchange 5.5 stark verändert (Kapitel 19.3). Das *Active Directory Service Interface (ADSI)* spielt aufgrund der Integration von Exchange2000 in das Active Directory weiterhin eine zentrale Rolle. Hinzugekommen ist einerseits der Zugriff auf den Exchange Information Store via *ActiveX Data Objects (ADO)* und zum anderen eine neue Komponente mit dem Namen *CDO for Exchange Management (CDOEXM)*, beschrieben in Kapitel 19.4.

Abbildung 1.2:
Überblick über die
Programmierschnittstellen des
Exchange Server
2000



Verfügbare Komponenten

Für den Bereich der Workflow-Funktionen bietet Exchange2000 die Komponente *Microsoft CDO Workflow Objects for Microsoft Exchange* (CDOWF). Einige Funktionen von Exchange2000 können auch über die Windows Management Instrumentation (WMI) überwacht und gesteuert werden. Das komplexe Thema der Automatisierung von Exchange2000 wird in Kapitel 19 zusammenhängend behandelt.

CDOWF, WMI

1.3.21 Lotus Notes

Die Firma Lotus bietet mit den

DCO, OLELibrary

- ▶ *Domino Collaboration Objects* (DCO) für Lotus Notes 5.x bzw.
- ▶ *Notes OLE Objects* für Notes 4.x

zwei COM-Komponenten.

Weitere Information zu DCO bietet Lotus auf seiner Entwickler-Seite [LOT01b] oder die Firma Duke Communications International auf [LOT01c].

1.3.22 SAP R/3

Die Firma SAP bietet mit dem *SAP DCOM Connector* eine COM-Komponente an, die den Zugriff auf Daten und Transaktionen im SAP-System ermöglicht. Der SAP DCOM Connector ist naturgemäß eine sehr komplexe Komponente. Empfehlenswerte Lektüre zu diesem Thema ist das Buch [SAP99].

SAP DCOM
Connector

1.3.23 Microsoft Office

Die Produkte der Microsoft Office-Produktfamilie sind heute das Vorzeigeelement hinsichtlich der Automatisierbarkeit durch COM-Komponenten. Nicht nur die großen Produkte wie

Office-
Komponenten

- ▶ Microsoft Excel,
- ▶ Microsoft Word,
- ▶ Microsoft PowerPoint,
- ▶ Microsoft Access,
- ▶ Microsoft Outlook und
- ▶ Microsoft FrontPage

können per COM gesteuert werden, sondern auch Hilfsanwendungen wie Microsoft Graph und der Microsoft Agent. Da dieses Buch im Schwerpunkt Serveranwendungen behandelt, gibt es in Kapitel 26 hierzu nur ein kleines Beispiel. Die Dokumentation der Automatisierungskomponenten von Microsoft Office füllen mehrere dicke Bücher, die bei verschiedenen Verlagen erhältlich sind.

1.4 Der Rest dieses Buchs

In den Kapiteln 4 bis 26 werden verschiedene COM-Komponenten zur Automatisierung administrativer Aufgaben in den Windows-Betriebssystemen sowie in Produkten der Microsoft BackOffice-Produktfamilie beschrieben. Der Schwerpunkt liegt dabei verständlicherweise auf den von Microsoft gelieferten Komponenten. Daneben wird eine Auswahl interessanter Komponenten anderer Anbieter vorgestellt.

Die Ausführlichkeit der Darstellung in diesem Buch richtet sich vor allem nach Bedeutung und Komplexität der Komponenten. Das *Active Directory Service Interface (ADSI)* zur Verzeichnisdienstverwaltung und die *Windows Management Instrumentation (WMI)* als übergreifender Ansatz zum Systemmanagement nehmen daher den größten Raum ein. In Wichtigkeit und Umfang folgen die *Scripting Runtime Library* für den Dateisystemzugriff und die *ActiveX Data Objects (ADO)* für den Datenbankzugriff.

Die Reihenfolge der Darstellung der Komponenten in diesem Kapitel ist dagegen eher didaktischer Natur: Mit der *WSH Runtime Library* und der *Scripting Runtime Library* stehen zwei Komponenten am Anfang, die einfach zu handhaben sind.

Ein repräsentativer Ausschnitt

Gerade bei großen Komponenten kann an dieser Stelle nur ein repräsentativer Ausschnitt der Komponente besprochen werden. Besonderes Ziel ist es daher, Ihnen ein Grundverständnis jeder einzelnen Komponente zu geben, damit Sie sich anschließend selbst weiter orientieren können. Zur Veranschaulichung ist das Objektmodell in Form einer Grafik wiedergegeben. Hinweise zu der dort verwendeten Notation sowie zu den Listings finden Sie im Anhang.

Auf eigener Suche

Mit Sicherheit werden Sie nach der Lektüre dieses Kapitels noch die eine oder andere Funktionalität vermissen. Auf der Suche nach Komponenten hilft Ihnen das in Kapitel 27 vorgestellte Vorgehensmodell.

2 COM-Theorie

Dieses Kapitel stellt die Aspekte von COM dar, die aus der Sicht eines Komponentennutzers wichtig sind. Dazu gehört auch ein gewisses Maß an Theorie. Die in diesem Zuge vermittelten theoretischen Grundlagen werden Sie in den Stand versetzen, COM-Komponenten wesentlich besser verstehen und gebrauchen zu können. Außerdem werden Sie lernen, die Dokumentationen von Komponenten besser zu verstehen.

Inhalt

Das Kapitel beginnt mit einer zunächst kurzen Erklärung grundlegender Begriffe wie *Binärstandard*, *Laufzeitumgebungen*, *Komponente*, *Klasse*, *Schnittstelle*, *Typbibliothek*, *GUID*, *Moniker* etc. Auf diesen knappen Definitionen aufbauend folgt die ausführlichere Darstellung aller wichtigen Bausteine und Konzepte von COM. Das Kapitel ist eher referenzartig aufgebaut – scheuen Sie sich nicht, einen der zahlreichen Querverweise zu folgen, wenn sie nähere Erläuterungen zu einem Begriff benötigen.

Aufbau

Wenn Sie darüber hinaus mehr über die Interna von COM oder die Komponentenprogrammierung mit C++ erfahren wollen, so sind das Buch des COM-Experten Don Box [BOX98], das aktuelle COM+-Werk von Eddon und Eddon [EDD00] sowie das DCOM-Buch von Victor Sirotn [SIR99] zu empfehlen. Detaillierte Informationen erhalten Sie auch auf der Microsoft COM-Site [MCO00], in der MSDN Library [MSD01d] und bei der Open Group [ATX00].

**Weitere
Informationen**

2.1 Binärstandard

COM ist ein Binärstandard für Komponenten und daher programmiersprachenunabhängig. Binärstandard bedeutet, dass

Binärstandard

- ▶ es einen definierten Satz von Datentypen gibt,
- ▶ es eine definierte Umsetzung dieser Datentypen in Bytefolgen gibt (z.B. wie ein String im Speicher abgelegt wird),
- ▶ es einen festgelegten Mechanismus gibt, wie ein bestimmter Block von Programmcode lokalisiert und aufgerufen wird.

2.2 Programmiersprachen für COM

COM ist grundsätzlich programmiersprachenunabhängig. Die Arbeit mit COM-Komponenten verlangt einer Programmiersprache die Unterstützung des COM-Binärstandards ab. Sofern diese Unterstützung implementiert ist, kann die Sprache mit einer Vielzahl unterschiedlicher COM-Objekte aus verschiedenen COM-Komponenten zusammenarbeiten.

Es sind inzwischen viele Sprachen verfügbar, die COM unterstützen. Diese Sprachen werden als COM-fähige Sprachen (engl. COM-enabled languages) bezeichnet.

Dabei bieten die meisten Sprachen eine Zwei-Wege-Unterstützung. Das bedeutet, dass es sowohl möglich ist, bestehende Komponenten zu nutzen als auch in dieser Sprache eigene Komponenten zu erzeugen.

**Verfügbare
Sprachen**

Die folgende Liste zeigt eine Auswahl der COM-fähigen Sprachen. Bei einigen dieser Sprachen ist die COM-Fähigkeit allerdings nicht integraler Bestandteil, sondern ein Add-on, das zusätzlich installiert werden muss.

- ▶ Visual Basic
- ▶ VBScript
- ▶ Visual Basic for Applications
- ▶ Delphi
- ▶ Visual C++ (Microsoft C++-Variante)
- ▶ Visual J++ (Microsoft Java-Variante)
- ▶ JScript
- ▶ Haskell
- ▶ Perl
- ▶ REXX
- ▶ PHP4
- ▶ alle .NET-Sprachen (C#, VB.NET, JScript.NET etc.)

Skriptsprachen

Früher boten die ActiveX-Skriptsprachen wie VBScript und JScript nur eine Ein-Weg-Unterstützung für COM. Durch das Konzept der Scriptlets können aber inzwischen auch in Skriptsprachen neue COM-Komponenten geschaffen werden.

2.3 Laufzeitumgebungen

**Vereinfachungen
für COM**

COM ist ein komplexes Modell und der direkte Zugriff auf COM ist entsprechend anspruchsvoll. Als Entwickler ist eine bessere Unterstützung bei der Nutzung und Erstellung von Komponenten wünschenswert. Dies fasst Griffl in dem Satz »DCOM liefert ... kein Komponentenframework im eigentlichen Sinne« [GRI98, Seite 84] zusammen.

Daher existieren für verschiedene Programmiersprachen Laufzeitumgebungen, die den Zugriff auf die Funktionen der COM-Bibliothek und den Umgang mit einzelnen COM-Komponenten ebenso wie die Implementie-

Laufzeitumgebungen

Die Laufzeitumgebungen vereinfachen die Entwicklung neuer COM-Komponenten. Diese Laufzeitumgebungen bieten Implementierungen für verschiedene COM-Standschnittstellen (insbesondere `IUnknown` und `IDispatch`) und ermöglichen somit die einfache Nutzung zahlreicher COM-Dienste.

Die Laufzeitumgebungen der verschiedenen COM-fähigen Programmiersprachen sind nicht zu verwechseln mit der COM-Bibliothek, welche das Application Programming Interface (API) von COM verkörpert (siehe Kapitel 2.15).

Leider hat bisher jede COM-fähige Sprache ihre eigene Laufzeitumgebung und daher ihre eigene Weise der COM-Programmierung. Eine einheitliche Laufzeitumgebung für COM-Komponenten ist bislang nicht vorhanden. Der COM-Standard definiert nur, wie Komponenten auf binärer Ebene genutzt werden sollen; aus der Sicht des Programmierers ergeben sich jedoch je nach Sprache, in der der COM-Client implementiert werden soll, große Unterschiede in der Art des Umgangs mit den Komponenten.

Uneinheitlichkeit

Die *VB Runtime* für Visual Basic und *Active Template Library (ATL)* für Visual C++ sind zwei Ansätze für eine Laufzeitumgebung. Die ATL befreit den C++-Entwickler beispielsweise davon, den `QueryInterface()`-Mechanismus und die Referenzzählung mit `AddRef()` und `Release()` für jede COM-Klasse selbst zu implementieren. (Diese Funktionen werden in Kapitel 2.15 erläutert!)

VB Runtime, ATL

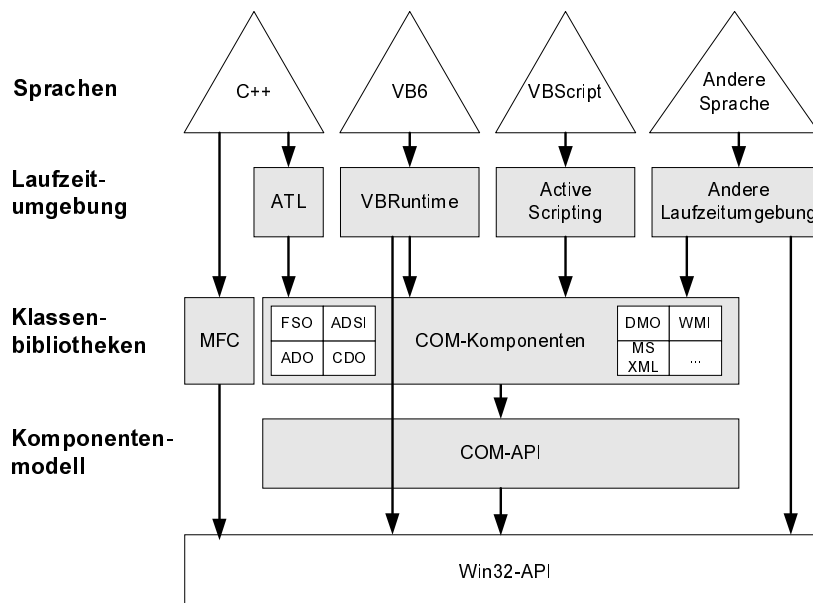


Abbildung 2.1:
Verschiedene
Laufzeitumgebungen in COM

Es wäre wünschenswert, wenn COM selbst eine solche Laufzeitumgebung bereitstellen würde, damit diese Laufzeitumgebung für alle Sprachen gleich wäre. Dieses Feature war ursprünglich für COM+ angekündigt (vgl. die

inzwischen berühmte Ankündigung einer COM+-Laufzeitumgebung von Mary Kirtland aus dem Microsoft Systems Journal [KIR98]), wurde jedoch nicht realisiert.

2.4 COM-Bausteine

Bausteine Die COM-Spezifikation definiert eine Reihe von Bausteinen (hier sollte man nicht von »Objekten« sprechen, weil der Begriff Objekt in der objektorientierten Programmierung bereits als Bezeichnung für die Instanz einer Klasse benutzt wird – vgl. Anhang A). Diese Bausteine betreffen einerseits den Aufbau einer Komponente, andererseits auch die mit einer Komponente in Beziehung stehenden Konfigurationsdaten. Die wichtigsten COM-Bausteine und ihre Beziehungen zueinander sind in der folgenden Grafik in Form eines aus der Datenmodellierung bekannten Entity-Relationship-Diagramms (ER-Diagramm) dargestellt. Das ER-Diagramm zeigt auch weitere COM-spezifische Konzepte, die in diesem Kapitel noch erläutert werden (siehe folgende Abbildung).

2.4.1 Die wichtigsten Bausteine im Kurzüberblick

Komponente COM-Komponenten sind objektorientierte Komponenten in dem in Anhang A beschriebenen Sinne. Sie sind die binäre Form einer Menge von COM-Klassen.



Nicht jede Ansammlung von Klassen im objektorientierten Sinne ist aber eine (COM-)Komponente. So kann eine Visual Basic-Anwendung auf Klassen basieren, die mangels der Implementierung von COM-Standardschnittstellen keine COM-Klassen sind und daher auch keine COM-Komponente bilden.

Klassenbibliothek Eine weitere Verwirrung in der COM-Begrifflichkeit entsteht dadurch, dass zum Teil die Begriffe Klassenbibliothek, Objektbibliothek oder COMBibliothek synonym mit COM-Komponente benutzt werden. Dabei ist der Begriff COM-Bibliothek natürlich besonders ungeeignet, da er in Konflikt mit der eigentlichen COM-Bibliothek, die Teil der Standardimplementierung von COM ist, steht.

Klasse Eine COM-Komponente fasst COM-Klassen zu einer Einheit zusammen. Eine COM-Komponente enthält mindestens eine COM-Klasse; in der Regel sind es jedoch mehrere Klassen. Weitere Informationen zu COM-Klassen finden Sie in Kapitel 2.10.

Objekt Wenn eine COM-Klasse instanziiert wird, entsteht ein COM-Objekt (Instanz der Klasse).

Schnittstelle Eine COM-Klasse ist die Implementierung einer oder mehrerer COM-Schnittstellen, wobei jede Schnittstelle eine beliebige Anzahl von Methoden umfasst. Weitere Informationen zu COM-Schnittstellen finden Sie in Kapitel 2.11.

COM-Bausteine

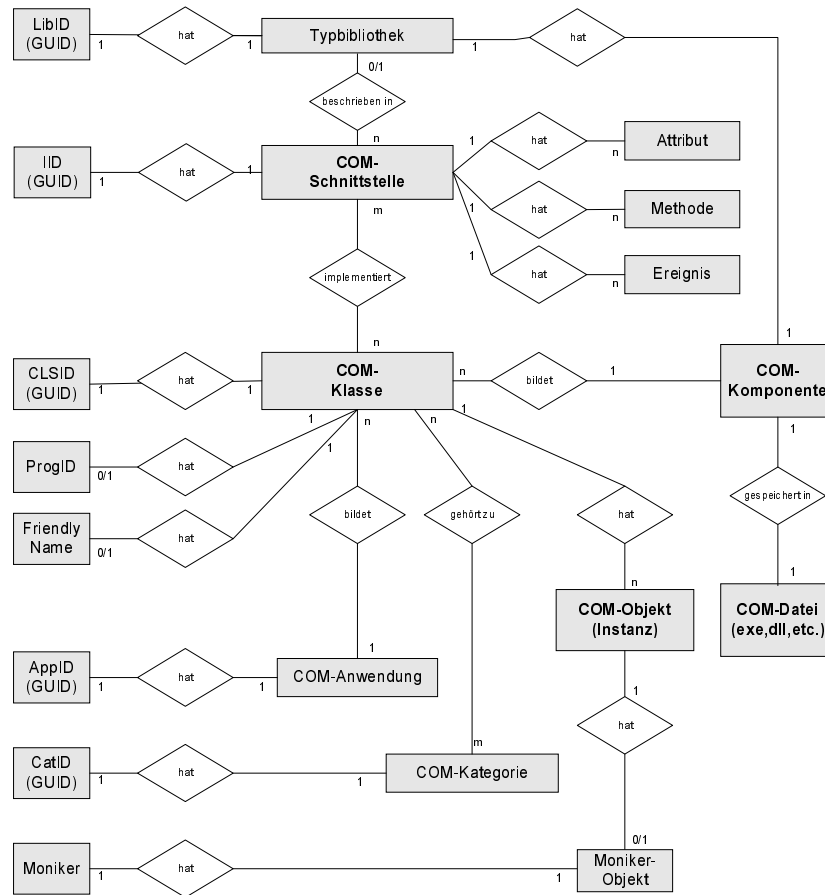


Abbildung 2.2:
ER-Diagramm der
Bausteine von COM
(Windows 9x/ME/
2000)

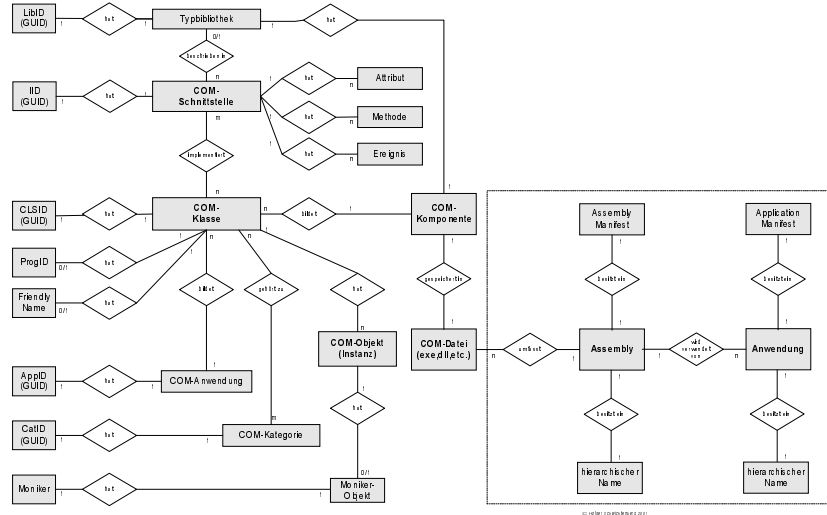
Eine Typbibliothek (engl. Type Library, kurz: TypeLib) enthält eine formale Beschreibung der Klassen und ihrer Schnittstellen und deren Mitglieder. Eine COM-Komponente kann eine Typbibliothek haben, sie muss aber keine haben. Typinformationen sind Thema in Kapitel 2.13.

Typbibliothek

Eine COM-Anwendung fasst eine oder mehrere COM-Klassen zusammen und ermöglicht die gemeinsame Konfiguration dieser Klassen. Zahlreiche Einstellungen (z.B. Sicherheitseinstellungen) sind nur auf Anwendungsebene, nicht jedoch für eine einzelne Klasse konfigurierbar. Jede COM-Klasse kann nur einer COM-Anwendung angehören. Alle Einstellungen einer COM-Anwendung gelten für alle zugehörigen Klassen. Mehr Informationen zu COM-Anwendungen finden Sie in Kapitel 2.18.

COM-Anwendungen

Abbildung 2.3:
ER-Diagramm der
Bausteine von COM
(Windows XP und
Windows .NET)



Komponenten- kategorien

COM-Kategorien stellen eine Möglichkeit dar, COM-Klassen zu kategorisieren. Die Kategorisierung dient dazu, leichter feststellen zu können, ob ein COM-Client und ein COM-Server zusammenpassen. Mehr Informationen zu COM-Komponentenkategorien finden Sie in Kapitel 2.19.

Assembly

Eine Assembly ist eine feste Ansammlung von einer oder mehreren Komponentendateien, die zum Zwecke der Versionierung zusammengefasst werden. Eine Assembly besitzt eine Konfigurationsdatei, die Assembly Manifest heisst. Ein Assembly Manifest ist eine XML-Datei.

Eine Assembly soll einen hierarchischen Namen besitzen in der Form

Firmenname.Anwendungsname.Anwendungsteil.Komponentenname



Das Konzept der Assemblies wurde erst mit Windows Whistler (Windows XP und Windows .NET) eingeführt. Assemblies und Manifeste in COM sind ähnlich, aber nicht gleich den Assemblies und Manifesten im .NET-Framework.

COM-Server und COM-Client

Client und Server

Oft wird auch der Begriff *COM-Server* für eine Komponente verwendet. Dieser Begriff stellt in den Vordergrund, dass eine Komponente ein Dienstanbieter ist. In Abgrenzung dazu heißt der Nutzer *COM-Client*. Ein COM-Client kann eine »normale« Anwendung, ein Skript oder wiederum eine Komponente sein.

Die Begriffe *COM-Server* und *COM-Client* werden immer dann benutzt, wenn klar abgegrenzt werden soll, wer Dienstanbieter und wer Dienstnutzer ist.

ter ist. Ein COM-Client kann natürlich selbst wiederum ein COM-Server für einen anderen COM-Client sein. Aus der Sicht eines COM-Clients ist ein COM-Server lokal (d.h. auf dem gleichen Rechnersystem) oder entfernt (engl. remote, d.h. auf einem anderen physikalischen Rechner).

Oft werden die Begriffe COM-Server und COM-Client auch für einzelne COM-Objekte und nicht nur für ganze COM-Komponenten verwendet.

2.4.2 Global Unique Identifier (GUIDs)

Die Spezifikation von COM verlangt, dass zentrale Bausteine der Komponentenarchitektur eindeutig identifizierbar sind. Diese Kennung soll nicht nur innerhalb eines Systems, sondern über Raum und Zeit eindeutig sein. Derartige Kennungen werden in COM *Global Unique Identifier* (kurz: GUIDs) genannt. Einige Autoren sprechen auch von Universal Unique Identifier (kurz: UUID). Der Nummernraum soll theoretisch ausreichen, um in der Zukunft auch über unseren eigenen Planeten hinaus Eindeutigkeit zu gewährleisten.

GUIDs werden mit Hilfe eines im *Distributed Computing Environment* (DCE) der *Open Software Foundation* (OSF) spezifizierten Algorithmus aus dem Netzwerknamen des Computers und der genauen aktuellen Uhrzeit gebildet. Sofern der Computer über keinen Netzwerknamen verfügt, wird eine Zufallszahl gebildet, die höchstwahrscheinlich eindeutig ist. Die Eindeutigkeit ist wahrscheinlich aufgrund der Größe der GUIDs: Sie umfasst 16 Byte (128 Bit), also einen Bereich von rund $3,4028236 \times 10^{38}$ Werten (2^{128}). Im Kontext mit einem bestimmten Baustein haben die GUIDs besondere Namen (vgl. Tabelle 2.1).

Die OSF ist ein Konsortium von Soft- und Hardwareherstellern. 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 Unterroutinen.

GUIDs

DCE-Algorithmus

COM-Baustein	GUID-Name
COM-Klasse	Class Identifier (CLSID)
COM-Schnittstelle	Interface Identifier (IID) oder Interface Pointer Identifier (IPID)
COM-Typbibliothek	Library Identifier (LibID)
COM-Komponentenkategorie	Category Identifier (CatID)
COM-Anwendung	Application Identifier (AppID)

Tabelle 2.1:
Verschiedene Typen
von GUIDs in COM

GUIDGEN GUIDs werden in der Regel als eine 16-stellige hexadezimale Zahl, eingrahmt in geschweifte Klammern, dargestellt. GUIDs können mit dem Werkzeug GUIDGEN (GUIDGEN.EXE) erzeugt werden. GUIDGEN wird u.a. mit Visual Studio 6.0 ausgeliefert.

Tabelle 2.2:
Beispiele für GUIDs

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



Folgende Identifikatoren in Zusammenhang mit COM sind keine GUIDs:

- ▶ sogenannte *Local Identifier (LCID)*, die Regionen bezeichnen
- ▶ Dispatch-IDs (vgl. Kapitel 2.14)
- ▶ ProgIDs (vgl. Kapitel 2.10)

2.4.3 Moniker

Moniker Die Namen für Instanzen heißen in der COM-Welt *Moniker* (engl.: Spitzname). Moniker sind eine textliche Repräsentation der Pfadangabe zu einem Objekt. Sie sind keine GUIDs. Es ist keine Pflicht für eine Instanz, einen Moniker zu haben.

Moniker haben die Form eines im Internet gebräuchlichen Uniform Resource Locators (URL). Ein URL hat die allgemeine Form

Protokoll:Protokoll-spezifischer Teil

Ein Moniker hat die Form

Moniker-Typ:Moniker-Typ-spezifischer Teil

Ein Moniker-Typ wird auch als Moniker-ProgID bezeichnet.

Tabelle 2.3:
Ausgewählte
Beispiele für
Moniker-Typen

Moniker-Typ	Erläuterung
File://	Datei oder Ordner im Dateisystem
WinNT://	Objekt im NT4-Verzeichnisdienst
LDAP://	Objekt in einem LDAP-Verzeichnisdienst
HTTP://	Objekt in einem Web-Ordner bzw. im Exchange2000-Webstore

Objekt	Moniker-Beispiel
Moniker für eine Word-Datei	FILE://SERVER/FREIGABE/VERZEICHNIS/DATEI.XLS
Moniker für einen Registrierungs-Schlüssel	RGY:// HKEY_LOCAL_MACHINE\SOFTWARE\IT-VISIONS
Moniker für ein Verzeichnis-objekt	LDAP://SONNE2000/CN=HS,CN=USERS, DC=IT-VISIONS,DC=DE
Moniker für eine WWW-Adresse	HTTP://WWW.WINDOWS-SCRIPTING.DE/ BOOK

Tabelle 2.4:
Beispiele für
Moniker

Weitere Informationen zu Monikern finden Sie in der MSDN Library [MSD01b].

2.5 COM-Dienste

Während Dienste im Rahmen der CORBA-Architektur eine zentrale Rolle einnehmen, ist der Begriff *Dienst* in der COM-Spezifikation unscharf. Folglich kommt es zu sehr unterschiedlichen Abgrenzungen der COM-Dienste. Diese sind keineswegs NT-Dienste, sondern basieren darauf, dass COM-Klassen bestimmte COM-Standardschnittstellen implementieren. Ein COM-Objekt kann einen Dienst nutzen, wenn seine Klasse die für den Dienst notwendigen Schnittstellen implementiert. Als COM-Dienste werden in der Regel betrachtet:

Dienste

- ▶ Statischer Methodenaufruf via Standardschnittstelle `IUnknown` (siehe Kapitel 2.14)
- ▶ Automation: dynamischer Methodenaufruf über die Standardschnittstelle `IDispatch` (siehe Kapitel 2.14)
- ▶ Namensdienst: Identifizierung bestehender Instanzen (siehe Kapitel 2.17.3)
- ▶ Sicherheit: Sicherheitseinstellungen für Komponenten (siehe Kapitel 2.22)
- ▶ Ereignisse: Objekte melden das Eintreten von Zuständen an ihren Client (siehe Kapitel 2.12)
- ▶ ActiveX-Steuerelemente: visuelle Elemente
- ▶ ActiveX-Dokumente (Object Linking and Embedding): Zusammensetzung von Dokumenten aus unterschiedlichen Dokumententypen
- ▶ Structured Storage: Persistenz für Objekte (siehe Kapitel 2.21)
- ▶ Distributed COM (DCOM): Zugriff auf entfernte Komponenten (siehe Kapitel 2.16)

Der Microsoft Transaction Server und COM+ erweitern COM um einige weitere Dienste (siehe Kapitel 2.25 und 2.26).

2.6 COM-Konfigurationsdaten

COM in der Registry

Die Informationen über die Komponenten und deren Konfiguration müssen an einem zentralen Ort gespeichert werden. Konfigurationsspeicher für COM war bis Windows XP allein die Registry. Ab Windows XP können Konfigurationsdaten auch in XML-Konfigurationsdateien im Pfad der Anwendung gespeichert werden. Die ursprünglichen Pläne eines in den Active Directory-Verzeichnisdienst integrierten *Class Store* sind bisher entgegen anders lautender Vorankündigungen nicht umgesetzt worden.

2.6.1 Die Registry als Konfigurationsspeicher

Folgende Orte in der Registry enthalten für COM relevante Informationen:

- ▶ HKEY_CLASSES_ROOT (äquivalent zu HKEY_LOCAL_MACHINE\SOFTWARE\CLASSES) ist der Hauptstandort für COM-Informationen.
- ▶ HKEY_LOCAL_MACHINE\SOFTWARE\MICROSOFT\OLE enthält globale COM-Konfigurationseinstellungen.
- ▶ HKEY_LOCAL_MACHINE\SOFTWARE\MICROSOFT\RPC enthält die Konfiguration der DCOM-Netzwerkprotokolle.

Referenz

Dieses Buch dokumentiert nur ausgewählte Schlüssel und Unterschlüssel. Die MSDN Library enthält eine komplette Referenz der COM-Registry-Schlüssel [MSD01e]. Allgemeine Informationen zur Registry als Informationsspeicher für COM gibt es in [MS01f].

HKEY_CLASSES_ROOT

HKEY_CLASSES_ROOT

Der Inhalt des Registry-Wurzelschlüssels HKEY_CLASSES_ROOT ist sehr unübersichtlich, weil dort verschiedenartige Informationen abgelegt sind:

- ▶ Dateiextensionen (File Extension Keys): Da diese mit einem Punkt beginnen, stehen sie am Anfang der Liste (z.B. .WRI).
- ▶ die den Dateiextensionen zugeordneten Dateitypen (z.B. *wrifile* für die Extension .WRI)
- ▶ ProgIDs der COM-Klassen (z.B. Word.Document.8)
- ▶ Moniker-Typen, z.B. HTTP://, FTP://, LDAP://, OUTLOOK://
- ▶ Unterschlüssel, in denen andere COM-Bausteine registriert sind (CLSID, INTERFACE, APPID, CATID, TYPELIB). Es wäre besser gewesen, für alle diese Informationstypen solche Unterschlüssel zu bilden, weil die Registry dann übersichtlicher wäre.

2.6.2 XML-Konfigurationsdateien

Windows Whistler (Windows XP und Windows .NET) unterstützen auch die Ablage von COM-Konfigurationsdaten in XML-Dateien. Diese XML-Dateien heißen *Manifeste*.

Manifest

Gleichzeitig wird ein neues Konzept in COM eingeführt: Eine *Assembly* ist eine feste Ansammlung von einer oder mehreren Komponentendateien. Eine *Assembly* hat eine Konfigurationsdatei, die *Assembly Manifest* heißt. Ein *Assembly Manifest* umfasst folgende Daten:

Assembly

- ▶ Ein Name der Assembly.
- ▶ Eine Versionsnummer
- ▶ Die Typangabe »win32«
- ▶ Liste der Komponentendateien, die zu der Assembly gehören
- ▶ COM-Konfigurationsdaten, die COM für die Lokalisierung und Aktivierung dieser Komponenten benötigt
- ▶ Liste der abhängigen Assemblies – jeweils mit Versionsnummer

Eine Anwendung besitzt eine XML-Datei, die *Application Manifest* heißt. Ein *Application Manifest* umfasst folgende Daten:

Application Manifest

- ▶ Ein Name der Anwendung.
- ▶ Eine Versionsnummer
- ▶ Liste der abhängigen Assemblies – jeweils mit Versionsnummer

Ein ähnliches Konzept wird auch im .NET-Framework verfolgt. Dort werden auch die Namen *Assembly* und *Manifest* verwendet. Das kann zu Verwirrung führen, da ein .NET-Manifest anders ist als ein COM-Manifest.

2.7 Komponentenarten

Bezüglich der Art und Weise, wie und wo COM-Komponenten gestartet werden, unterscheidet man verschiedene Arten von COM-Komponenten:

- ▶ prozessintern (engl. in-process, kurz: in-proc)
- ▶ prozesseextern (engl. out-process, kurz: out-proc)
- ▶ lokal (engl. local)
- ▶ entfernt (engl. remote)

Prozess**interne** Komponenten laufen im Prozess des COM-Clients, während prozess**externe** Komponenten in einem separaten Prozess mit einem eigenen Adressraum ausgeführt werden. Da prozesseexterne COM-Komponen-

**Prozessintern
versus
prozesseextern**

ten keinen gemeinsamen Adressraum mit dem COM-Client besitzen, ist der Datenaustausch zwischen COM-Client und COM-Server sehr viel aufwendiger und damit langsamer als bei prozessinternen Komponenten. Prozessexterne Komponenten haben jedoch auch Vorteile: Ein Absturz der Komponente reißt nicht zwangsläufig auch den gesamten COM-Client in den Abgrund. Prozessexterne Komponenten sind also robuster. Außerdem können sie unabhängig von einem Client gestartet werden.

Lokal versus entfernt Eine lokale Komponente läuft auf demselben Rechner wie der COM-Client, eine entfernte Komponente auf einem anderen System. Eine Komponente, die ein entfernter COM-Server ist, muss natürlich immer in einem eigenen Prozess laufen, da ein Prozess mit einem rechnerübergreifenden Adressraum nicht möglich ist.

Arten des Prozeduraufrufs

LPC Der direkte Aufruf einer Unteroutine ist nur dann standardmäßig möglich, wenn die Unteroutine im gleichen Adressraum wie die aufrufende Routine liegt. Man spricht dann von einem *Local Procedure Call (LPC)*. Nur In-process-Komponenten erfüllen diese Anforderung. Sobald die aufzurufende Unteroutine in einem anderen Prozess liegt, läuft sie in einem anderen Adressraum und ist daher über LPC nicht mehr erreichbar.

RPC Der Aufruf einer Unteroutine in einem anderen Adressraum wird *Remote Procedure Call (RPC)* genannt. Dabei wird üblicherweise davon ausgegangen, dass diese Unteroutine sich auch auf einem anderen Rechner befindet.

LRPC In COM wird ein RPC aber auch bei der Verwendung von Objekten in Out-process-Komponenten auf dem selben Rechner und zwischen verschiedenen Threads innerhalb eines Prozesses eingesetzt. Da allerdings die Inter-process-Kommunikation noch wesentlich einfacher ist als die Inter-system-Kommunikation, spricht man in solchen Fällen von einem *Lightweight RPC (LRPC)* oder *unechtem RPC*. Die grundsätzlichen Herausforderungen sind aber ähnlich wie beim *echten RPC*. Zusätzlich gilt es beim echten RPC jedoch, die Netzwerklast möglichst gering zu halten.

Marshalling

Marshalling Der Funktionsaufruf erfolgt beim RPC/LRPC durch Nachrichtenaustausch. Der Vorgang ist aber für den Aufrufer und Empfänger transparent, d.h., er unterscheidet sich in der Form nicht von dem Aufruf der gleichen Funktion auf dem lokalen Rechner. Die Verteiltheit soll allenfalls am Performance-Unterschied erkennbar sein.

Proxy und Stub Um einen RPC/LRPC auszuführen, ist es notwendig, den Funktionsnamen und die Parameter in eine Nachricht (Paket) zu verpacken, die über das Netzwerk dem entfernten System bzw. dem anderen Prozess zugestellt wird. Ebenso müssen die Rückgabewerte nach Ende der Prozedur an den Aufrufer zurückgegeben werden. Eine besondere Herausforderung sind dabei Zeiger. Hier müssen Speicherbereiche übergeben werden, da eine Übergabe eines Zeigers zwischen Prozess- und Systemgrenzen hinweg sinn-

Komponentenarten

los wäre. Der Vorgang der Parameterübergabe wird als *Marshalling* bezeichnet und findet im sogenannten *Proxy* statt. Auf der Gegenseite erfolgt im *Stub* ein DeMarshalling. Das Format, in dem die Daten übertragen werden, heißt Network Data Representation (NDR).

Für Proxy und Stub sind auch einige andere Bedeutungen im Umlauf. So wird der Marshaller auf der Client-Seite auch *Client-Stub* oder *Server-Proxy* (Stellvertreter des Servers) genannt, während die Gegenseite jeweils analog *Server-Stub* oder *Client-Proxy* genannt wird.

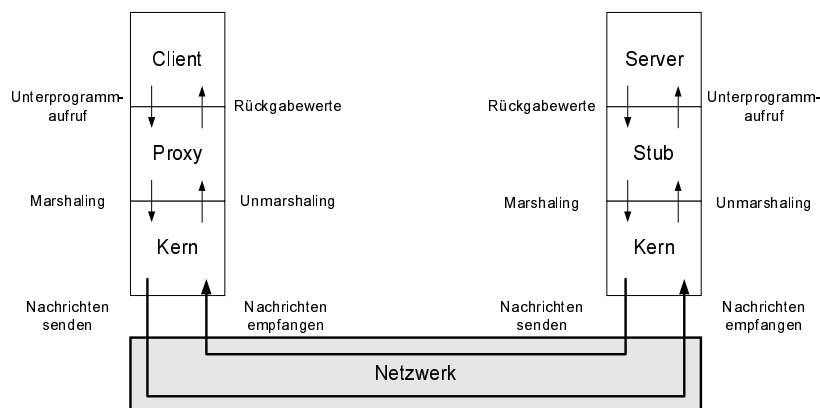


Abbildung 2.4:
Ablauf eines RPCs

Proxy- und Stub-Objekte

Proxys bzw. Stubs sind in COM/DCOM selbst COM-Objekte. Die COM-Bibliothek OLE32.DLL enthält für die meisten COM-Standardschnittstellen bereits fertige Proxy- und Stub-Objekte.

Proxys und Stubs

Es gibt drei Möglichkeiten des Marshallings für benutzerdefinierte Schnittstellen (siehe auch [GRM97]):

► Standard Marshalling

Bei diesem Verfahren wird der sogenannte MIDL-Compiler eingesetzt, um Proxy und Stub automatisch zu generieren. Dazu wird eine Schnittstellenbeschreibung in der Interface Definition Language (IDL) benötigt (zu IDL und MIDL siehe Kapitel 2.13).

IDL-basiert

► TypLib Marshalling

Für das Marshalling von IDispatch-Schnittstellenzeigern sind keine eigenen Proxy-/Stub-Objekte nötig, da der sogenannte *Automations-Marshaller* (auch: *Universal Marshaller*) von COM diese Aufgabe übernimmt. Er wird von Microsoft in Form der OLEAUT32.DLL bereitgestellt.

IDispatch-basiert

► Custom Marshaller

Für spezielle Anwendungsfälle besteht auch die Möglichkeit, das Marshalling selbst zu implementieren. Custom Marshaller können bislang al-

individuell

lerdings ausschließlich in C++ implementiert werden. Für einige wichtige Klassen liefern die Hersteller jedoch *Custom Marshaller* mit. Microsoft liefert im Rahmen der Data Access Components einen Custom Marshaller für die Klasse `ADODB.Recordset`, die relationale Tabellen speichern kann.

Proxy und Stub in Visual Basic

Hochsprachen wie Visual Basic erzeugen Proxy- und Stub-Objekte automatisch. Ein Skriptprogrammierer braucht sich um das Marshalling nicht zu sorgen, da er sowieso nur `IDispatch` nutzen kann.

2.8 Verpackungsform

- COM-Dateien** Eine sich aus COM-Klassen zusammensetzende COM-Komponente bildet eine abgeschlossene Einheit, die als solche in binärer Form in einer Datei im Dateisystem gespeichert wird.
- Eine solche Komponentendatei enthält die Definition der Klassen und den Programmcode. Hier können jedoch keine Instanzen der Klassen gespeichert werden.
- Dateiarten** Es gibt inzwischen vier Dateiformen (Physical Packaging), die COM-Komponenten enthalten können:
- ▶ Dynamic Linking Libraries (.DLL / .OCX)
 - ▶ ausführbare Windows-Dateien (.EXE)
 - ▶ Java-Klassen (.CLASS)
 - ▶ Skriptdateien (*Scriptlets* oder *Windows Script Components*) genannt (.SCT und .WSC)
- Dabei ist .DLL die häufigste Verpackungsform. Nicht jede .EXE-, .DLL- oder .CLASS-Datei ist jedoch eine COM-Komponente. Die Dateierweiterung .OCX wird dagegen nur für COM-DLLs verwendet, die ActiveX-Steuerelemente enthalten.
- Script Components** *Windows Script Components* (WSCs) sind in Skriptsprachen geschriebene COM-Komponenten. Der Begriff WSC steht jedoch weder allgemein für Komponenten, die von Skripten aus genutzt werden können, noch für die Bausteine der Windows Scripting-Architektur.
- J++** Nur mit Microsoft Visual J++ können Java-Klassen erzeugt werden, die auch COM-Komponenten sind. Da Microsoft die Weiterentwicklung von J++ nach einem Rechtsstreit mit Sun eingestellt hat, ist diese Variante von COM-Komponenten unwichtig. Microsoft bietet aber einen Migrationspfad von J++ zu Microsoft .NET unter dem Namen *JUMP* an.

In Anlehnung an den Begriff COM-Server spricht man auch von EXE-Servern für eine Komponente in einer .EXE-Datei bzw. von DLL-Servern für eine Komponente in einer .DLL-Datei.

Inhalt einer COM-DLL

Eine COM-DLL erkennt man daran, dass sie die vier folgenden DLL-Funktionen (DLL-Einsprungpunkte) exportiert:

► DllRegisterServer

Diese Funktion dient der Registrierung (d.h. der Eintrag in die Registry) der Komponente. Sie wird von der COM-Bibliothek aufgerufen. Es ist Aufgabe der Komponente, die Registrierung selbst zu implementieren.

Registrierung

► DllUnregisterServer

Diese Funktion dient der Deregistrierung (d.h. die Entfernung aus der Registry) der Komponente. Sie wird von der COM-Bibliothek aufgerufen. Es ist Aufgabe der Komponente, die Deregistrierung selbst zu implementieren.

Deregistrierung

► DllGetClassObject

Diese Funktion liefert einen Zeiger auf die Class Factory (siehe Kapitel 2.17) für eine bestimmte COM-Klasse zurück. Der Client kann danach die Class Factory aufrufen, die im Auftrag des Clients eine Instanz der COM-Klasse erzeugt.

Instanz erzeugen

► DllCanUnloadNow

Mit dieser Funktion kann die COM-Bibliothek erfragen, ob die DLL nicht mehr gebraucht wird. Dies ist dann der Fall, wenn keine Instanzen von COM-Klassen in dieser Komponente mehr benutzt werden. Dazu gibt es in jeder Instanz eine Referenzzählung. Diese wird später im Zusammenhang mit der Standardschnittstelle IUnknown erläutert (Kapitel 2.14).

Entfernbarkeit

Bei einer COM-EXE findet man diese Funktionen nicht.

2.8.1 EXE-Server im Vergleich zu DLL-Server

Ein EXE-Server hat gegenüber einem DLL-Server den Vorteil, dass die COM-Komponente unabhängig von einem Client gestartet und beendet werden kann. So kann ein EXE-Server schon gestartet werden, bevor es einen Client gibt. Bei der ersten Instanziierung eines Clients erfolgt der Aufruf dann wesentlich schneller als wenn die Komponente erst dann geladen werden müsste. Eine Komponente in Form einer COM-EXE kann auch als Windows NT-Dienst laufen. Der Nachteil von EXE-Servern ist jedoch, dass diese immer in einem eigenen Prozess laufen, was Performance-Nachteile bei jedem einzelnen Methodenaufruf mit sich bringt.

EXE vs. DLL

COM-Theorie

Abbildung 2.5:
Betrachtung einer
COM-DLL mit dem
Werkzeug »Depen-
dency Walker«
(DEPENDS.EXE)

[illegible]

Abbildung 2.6:
Betrachtung einer
COM-EXE mit dem
Werkzeug »Depen-
dency Walker«
(DEPENDS.EXE)

The screenshot shows the Dependency Walker application. The 'Loaded Modules' pane on the left lists the following modules:

- ADVAPI32.dll
- GDI32.dll
- USER32.dll
- SHL3.dll
- OLE32.dll
- RPCRT4.dll
- GDI32.dll
- RPCRT4.dll
- USER32.dll
- ADVAPI32.dll
- SHL3.dll
- USER32.dll

The 'Dependency List' pane on the right shows the following dependencies:

Ordinal	Path	Function	Entry Point
36	[D:\WIN\...]	GetProcAddress	0x00404023
37	[D:\WIN\...]	GetModuleHandle	0x00404023
38	[D:\WIN\...]	GetProcAddress	0x00404023
39	[D:\WIN\...]	GetProcAddress	0x00404023
40	[D:\WIN\...]	GetProcAddress	0x00404023
41	[D:\WIN\...]	GetProcAddress	0x00404023
42	[D:\WIN\...]	GetProcAddress	0x00404023
43	[D:\WIN\...]	GetProcAddress	0x00404023
44	[D:\WIN\...]	GetProcAddress	0x00404023
45	[D:\WIN\...]	GetProcAddress	0x00404023

The 'Module List' pane at the bottom shows the following modules:

Module	Size	Checksum	Signature	Language	Architecture	Company Name	Product Name	Version	File Size	File Date	File Time	Image Size
ADVAPI32.dll	655,360	4,194	7,155,792	OK	x86	Microsoft Corporation	Microsoft Windows Common-...	6.0.6002.18005	655,360	2006.11.16	2006.11.16	655,360
GDI32.dll	655,360	4,194	7,155,792	OK	x86	Microsoft Corporation	Microsoft Windows Common-...	6.0.6002.18005	655,360	2006.11.16	2006.11.16	655,360
USER32.dll	655,360	4,194	7,155,792	OK	x86	Microsoft Corporation	Microsoft Windows Common-...	6.0.6002.18005	655,360	2006.11.16	2006.11.16	655,360
SHL3.dll	655,360	4,194	7,155,792	OK	x86	Microsoft Corporation	Microsoft Windows Common-...	6.0.6002.18005	655,360	2006.11.16	2006.11.16	655,360
OLE32.dll	655,360	4,194	7,155,792	OK	x86	Microsoft Corporation	Microsoft Windows Common-...	6.0.6002.18005	655,360	2006.11.16	2006.11.16	655,360
RPCRT4.dll	655,360	4,194	7,155,792	OK	x86	Microsoft Corporation	Microsoft Windows Common-...	6.0.6002.18005	655,360	2006.11.16	2006.11.16	655,360
ADVAPI32.dll	655,360	4,194	7,155,792	OK	x86	Microsoft Corporation	Microsoft Windows Common-...	6.0.6002.18005	655,360	2006.11.16	2006.11.16	655,360
SHL3.dll	655,360	4,194	7,155,792	OK	x86	Microsoft Corporation	Microsoft Windows Common-...	6.0.6002.18005	655,360	2006.11.16	2006.11.16	655,360
USER32.dll	655,360	4,194	7,155,792	OK	x86	Microsoft Corporation	Microsoft Windows Common-...	6.0.6002.18005	655,360	2006.11.16	2006.11.16	655,360

2.8.2 Der Zusammenhang zwischen Komponententart und Verpackungsform

Versionsnummer
und andere Meta-
Informationen

Die Verpackungsform der Komponente hat Einfluss auf die Komponententart:

- ▶ Scriptlets sind immer **prozessintern**.
- ▶ Eine **COM-EXE** ist immer eine **prozesseexterne** Komponente.
- ▶ Eine **COM-DLL** ist normalerweise eine **prozessinterne** Komponente. Allerdings kann eine COM-DLL mit Hilfe eines sogenannten Surrogat-Prozesses auch in einem eigenen Prozess laufen. Ein *Surrogat-Prozess* ermöglicht es einem DLL-Server, in einem eigenständigen Prozess zu laufen.

	lokal	entfernt
In-Process	.DLL .OCX .WSC	nicht möglich
Out-Process	.EXE .DLL/.OCX/.WSC (wenn mit Surrogat-Prozess)	.EXE .DLL/.OCX/.WSC (wenn mit Surrogat-Prozess)

Abbildung 2.7:
Zusammenhang
zwischen Kompo-
nententart und Ver-
packungsform

Es ist Zielsetzung von Microsoft, die Verpackungsform und die Aktivierungsform zunehmend voneinander zu trennen. Die Zukunft soll den COM-DLLs gehören – COM-EXEs sind eine von Microsoft zum Aussterben verdamnte Art.

**COM-EXEs sollen
aussterben**

2.8.3 Zusatzinformationen

Es ist üblich, in den erweiterten Dateiattributen der .DLL- und .EXE-Dateien Informationen über die Datei zu speichern, z.B. den Hersteller, die Versionsnummer und weitere Kommentare. Diese Informationen werden von den Werkzeugen wie dem Microsoft COM-Viewer allerdings nicht angezeigt. Sie können diese Daten entweder über die Eigenschaften der Komponentendatei selbst oder aber über das Werkzeug COM-Explorer (siehe Kapitel 2.28.5) einsehen.

Abbildung 2.8:
Attribute einer typi-
schen COM-DLL
(Eigenschaftsfenster
der Datei)



2.9 Registrierung von Komponenten

Komponenten- registrierung

Bevor Komponenten auf einem Rechnersystem benutzt werden können, müssen sie registriert werden, d.h., es müssen die passenden Einträge für Objekte, Schnittstellen etc. in der Registry erzeugt werden. Die Registrierung erfolgt bei den verschiedenen Komponenten-Dateitypen unterschiedlich:

- ▶ COM-EXE-Dateien sollten sich beim ersten Aufruf selbst registrieren. Einige erwarten für den Registrierungsvorgang die Kommandozeilenoption /REGSERVER. Mit /UNREGSERVER wird die Registrierung aufgehoben.
- ▶ Komponenten in .DLL- und .OCX-Dateien sowie .SCT- und .WSC-Dateien werden mit dem Kommandozeilentool REGSVR32.EXE registriert. REGSVR32.EXE stößt den Aufruf der exportierten DLL-Funktionen DllRegisterServer bzw. DllUnRegisterServer in der Komponenten-DLL an.

Fehlerursachen

Abhängige Komponenten

Die erfolgreiche Registrierung der Komponente ist eine notwendige, aber nicht hinreichende Bedingung für das Funktionieren der Komponente. So

kann eine Komponente andere Komponenten benötigen, die nicht vorhanden sind. Eine häufige Ursache für das Fehlschlagen der Komponentenregistrierung besteht darin, dass DLLs, von denen die Komponenten abhängig sind, nicht vorhanden sind.

Eine Hilfe bei der Suche nach Abhängigkeitsproblemen ist das Werkzeug *Dependency Walker* (siehe Kapitel 2.28.6). Der Dependency Walker zeigt die Abhängigkeiten zwischen DLLs, wobei die Abhängigkeitsinformationen in sogenannten Importlisten innerhalb jeder DLL gespeichert sind. Zu beachten ist aber, dass die DLL-Importlisten keine COM-Technologie sind. Man kann anhand der Listen nur erkennen, welche Abhängigkeiten zu Nicht-COM-DLLs existieren. Dagegen kann man aber nicht erkennen, welche COM-DLLs oder anderen COM-Komponenten eine DLL benutzt, weil diese Funktionalität innerhalb der COM-Spezifikation nicht vorgesehen ist.



Einige Komponenten können nur funktionieren, wenn unter HKEY_CLASSES_ROOT\LICENSES eine entsprechende Lizenz eingetragen ist.

Lizenz

Zukunft

Es gab Gerüchte, dass Microsoft die Registry in Windows2000 durch einen Class Store im Active Directory ablösen wollte. Dies wurde nicht realisiert. Im .NET-Framework wird die Registry nicht mehr benötigt: ein pfadbasierter Suchmechanismus und Konfigurationsdateien im XML-Format übernehmen die Aufgaben der Registry im .NET-Komponentenmodell.

Class Store und
DOTNET

2.10 COM-Klassen

Es gibt in COM zwei Arten von Klassen. Sie unterscheiden sich darin, wer eine Instanz der Klasse erzeugen kann:

COM-Klassen

- ▶ (Öffentlich) erzeugbare Klassen (creatable classes) können direkt durch einen COM-Client erzeugt werden.
- ▶ Nicht (öffentlich) erzeugbare Klassen (non-creatable classes) können nur von Instanzen anderer COM-Klassen der gleichen Komponente erzeugt werden.

Man setzt gelegentlich *öffentlich* vor die Begriffe erzeugbare/nicht-erzeugbare Klasse, um klarzustellen, dass sich die Erzeugbarkeit nur auf den COM-Client bezieht. Selbstverständlich sollte eine Klasse zumindest immer durch eine andere Klasse derselben Komponente selbst erzeugbar sein. Eine Klasse, die weder durch den COM-Client noch innerhalb der Komponente selbst erzeugbar ist, ist nicht sinnvoll, weil sie nicht verwendet werden kann.

In der Regel verfügt eine Komponente nur über sehr wenige erzeugbare Klassen. Die meisten Klassen einer Komponente sind von bestimmten Initialisierungen abhängig, die durch die Instanziierung bzw. durch die Initialisie-

Erzeugbare
Klassen

nung in Folge der Instanziierung einer zentralen Stammklasse vorgenommen werden. Erzeugbare Klassen sind oft die Stammklassen von Objektmodellen (vgl. Kapitel 2.27).

Ein COM-Objekt ist eine Instanz einer COM-Klasse. Dabei ist in der Regel gewünscht, dass es mehr als nur eine Instanz jeder Klasse geben kann.

Bezeichner für Klassen

Es gibt drei verschiedene Arten von Bezeichnern für Klassen, wobei eine einzelne COM-Klasse alle drei Arten von Bezeichnern haben kann:

- ▶ ein Class Identifier (CLSID)
- ▶ ein Programmatic Identifier (kurz: ProgID)
- ▶ ein Friendly Class Name

Diese Konzepte werden im Folgenden beschrieben.

2.10.1 Klassen-Identifikation mit CLSIDs

CLSIDs, CoClass Ein Class Identifier (CLSID) ist eine GUID für eine Klasse. Nicht jede COM-Klasse verfügt jedoch über eine CLSID. CLSIDs werden nur für sogenannte *CoClasses* vergeben, für die in der Interface-Definition (vgl. Kapitel 2.13) eine explizite Klassendefinition (*CoClass* genannt) vorliegt. Da, wie in Kapitel 2.11 beschrieben, Schnittstellen und nicht Klassen das Element für den Zugriff auf Objekte in COM sind, ist eine explizite Klassendefinition via CoClass mit CLSID nur für zwei Arten von Klassen notwendig:

- ▶ für erzeugbare Klassen
- ▶ für Klassen, die Ereignisse aussenden

Alle anderen Klassen können, müssen aber nicht als CoClass definiert werden.

2.10.2 Programmatic Identifier

ProgIDs Ein Programmatic Identifier (kurz: ProgID) ist eine Zeichenkette, die mit einer CLSID assoziiert werden kann. Die ProgID kann ebenso wie die CLSID benutzt werden, um auf eine Klasse zuzugreifen. Die COM-Bibliothek setzt eine ProgID mit Hilfe der Funktion `CLSIDFromProgID()` in eine CLSID um. Eine ProgID hat die Form `KomponentenName.KlassenName.Version`. Die Angabe einer Versionsnummer ist dabei optional. Eine ProgID ohne Versionsnummer heißt `VersionIndependentProgID`. Die Namensvergabe der ProgID ist leider nur eine Konvention, kein Automatismus. Daher gibt es auch ProgIDs, die nur aus dem Klassennamen bestehen (z.B. `AdsSecurity`, `AdsSID`, vgl. Kapitel 13.11).

Eine ProgID kann man sich wesentlich einfacher merken als eine lange CLSID. Zweck der ProgID ist es, dem Benutzer (Programmierer) den Zugriff auf COM-Klassen zu vereinfachen. Eine ProgID ist jedoch weniger präzise als eine CLSID: Sie ist nicht weltweit eindeutig. Die ProgID kann jeder COM-Programmierer frei wählen; es besteht die Gefahr, dass mehrere Programmierer für unterschiedliche Klassen die gleiche ProgID verwenden!

Leider gibt es keine Konsistenzprüfung bezüglich des Aufbaus einer ProgID. Komponentenerzeuger sind daher nicht gezwungen, sich an die oben genannte Konvention `KomponentenName.KlassenName` zu halten.

- ▶ So kann eine Klasse in der ProgID einen Namen bekommen, der nicht dem Namen in der Typbibliothek entspricht.
- ▶ Es ist auch möglich, dass Klassen, die in einer Datei (also in einer Komponente) enthalten sind, in der ProgID verschiedene Komponentennamen erhalten.
- ▶ Manche ProgIDs bestehen auch nur aus einem Klassennamen (z.B. eine ProgID wie `ADsSIID`).

Der COM Runtime sind diese Inkonsistenzen egal, weil intern sowieso nur mit eindeutigen CLSIDs gearbeitet wird. Sie sollten sich jedoch bei selbst erstellten Komponenten an die Konvention halten, dass jede COM-Datei genau einen Komponentennamen hat und der Klassename der Typdefinition entspricht. Sie machen es damit dem Anwender leichter. Es gibt leider viele Komponentenprogrammierer, die sich nicht an diese Konvention halten.



2.10.3 Friendly Class Name

Neben der CLSID und der ProgID kann es noch eine weitere Bezeichnung für eine Klasse geben. Dieser Bezeichnung ist ein beliebiger Text und heißt *Friendly Class Name*.

**Friendly Class
Name**

Dieser *Friendly Class Name* weicht leider häufig von der ProgID ab. So ist beispielsweise die Klasse `{3FA7DEB3-6438-101B-ACC1-00AA00423326}` der ProgID `MAPI.Session` zugeordnet, als Friendly Class Name ist jedoch `Active Messaging Session Object` eingetragen. Dieser Friendly Class Name wird vom *Microsoft COM-Viewer* (siehe Kapitel 2.28.4) zur Anzeige verwendet.

2.10.4 Klassen in der Registry

CLSIDs sind im Registry-Schlüssel `HKEY_CLASSES_ROOT\CLSID` abgelegt. Es würde dem Anwender das Verständnis der Zusammenhänge erleichtern, wenn für jede COM-Klasse eine CoClass existierte.

Komponentendatei**LocalServer32 und
InProcServer32**

Wichtigste Aufgabe des CLSID-Schlüssels ist die Bindung an eine COM-Komponentendatei sowie an eine Typbibliothek. Bei In-process-Komponenten steht der Verweis auf die implementierende Datei im Unterschlüssel INPROCServer32. Bei Out-process-Komponenten steht der Verweis auf die implementierende Datei im Unterschlüssel LOCALServer32. Sofern beide Einträge angegeben sind, wird die Out-process-Komponente bevorzugt. Es ist nicht möglich, dass der Pfad zu der Datei als UNC-Pfad angegeben wird, selbst dann nicht, wenn der UNC-Pfad auf das lokale System verweist.

ProgIDs in der Registry**ProgIDs**

Die ProgIDs befinden sich direkt unterhalb von HKEY_CLASSES_ROOT und machen diesen Hauptordner daher sehr unübersichtlich. Ein ProgID-Schlüssel enthält als einzigen Pflichteintrag den Verweis auf die zugehörige CLSID. Eine *VersionIndependentProgID* (eine ProgID ohne Versionsnummer) enthält, sofern mehrere Versionen der COM-Klasse installiert sind, im Unterschlüssel CURVER die ProgID der aktuellsten Version dieser Klasse.

Friendly Class Name in der Registry**Friendly Class
Name**

Der Friendly Class Name ist als Standardattribut des CLSID-Schlüssels gespeichert.

Weitere Werte

Weiterhin enthält ein CLSID-Schlüssel die in nachstehender Tabelle aufgelisteten Werte.

Tabelle 2.5:
Unterschlüssel
einer CLSID

Schlüssel	Erläuterung
AppID	AppID der COM-Anwendung, zu der die Klasse gehört
AutoConvertTo	CLSID der Klasse, an die alle Aufrufe weitergeleitet werden sollen
AuxUserType	Kurzname der Klasse
Control	Identifiziert eine Klasse als ein ActiveX-Steuererelement
DefaultIcon	Verweis auf ein Icon zur grafischen Repräsentation von Instanzen der Klasse
ImplementedCategories	Liste der COM-Kategorien, zu denen die Klasse gehört
InprocHandler	Verweis auf In-process-Handler (16 Bit)
InprocHandler32	Verweis auf In-process-Handler (32 Bit)
InprocServer	Verweis auf In-process-Komponente (16 Bit)
InprocServer32	Verweis auf In-process-Komponente (32 Bit)

COM-Schnittstellen

Schlüssel	Erläuterung
Insertable	Zeigt an, dass diese Klasse per OLE in Dokumente eingefügt werden kann
Interface	Liste der implementierten Interfaces in Form von IIDs. Dieser Unterschlüssel wird leider in der Praxis nicht verwendet.
LocalServer	Verweis auf implementierende Out-process-Komponente (16 oder 32 Bit)
LocalServer32	Verweis auf implementierende Out-process-Komponente (32 Bit)
ProgID	Programmatic Identifier (ein Alias für eine CLSID in Form einer Zeichenkette)
RequiredCategories	Liste der COM-Kategorien, zu denen der aufrufende Client gehören muss
TypeLib	LibID der zugehörigen Typbibliothek
ToolBoxBitmap32	Verweis auf eine Bitmap zur Darstellung der Klasse in einer Toolbox (gilt hauptsächlich für visuelle, also ActiveX-Komponenten)
TreatAs	CLSID einer Klasse, die diese Klasse emulieren kann
Verb	Liste zu dieser Klasse gehörender Menüeinträge
Version	Versionsnummer

Tabelle 2.5:
Unterschlüssel
einer CLSID
(Forts.)

Schlüssel wie *Insertable* und *Control* stellen eine Kategorisierung dar, die jedoch veraltet ist. Heute ist es üblich, COM-Kategorien zu verwenden.

Für die Abbildung der n-zu-m-Verknüpfung zwischen Klassen und Schnittstellen ist unterhalb der CLSID ein Unterschlüssel mit dem Namen *Interface* vorgesehen. In der Praxis wird dieser Schlüssel jedoch nicht benutzt. Die implementierten Interfaces einer Klasse können durch `IUnknown::QueryInterface()` erfragt werden.

Interfaces

2.11 COM-Schnittstellen

Das zentrale Element in der COM-Architektur sind nicht die Klassen, sondern die Schnittstellen. Eine Schnittstelle definiert eine Menge von Attributen, Methoden und Ereignissen. Eine COM-Schnittstelle ist ein abstraktes Gebilde. Sie wird implementiert durch eine COM-Klasse. COM-Klassen implementieren eine oder mehrere COM-Schnittstellen; COM unterstützt also Mehrfachschnittstellen. Klassen sind bildlich gesehen eine Klammer um eine Menge von Schnittstellen. Man bezeichnet COM auch als *schnittstellenbasiertes Programmieren*.

Schnittstellen als zentrales Element in COM

Als Instanz einer Klasse erhält ein COM-Objekt ausnahmslos alle Schnittstellen, die die COM-Klasse implementiert. Die Funktionalität eines COM-Objekts ergibt sich also aus der Gesamtfunktionalität aller Schnittstellen.



Der *Objektkatalog* in Visual Basic 6.0 und VBA erzeugt durch die Verheimlichung der Existenz von Schnittstellen bzw. durch die Erhebung aller Nichtstandardschnittstellen zu »Klassen« Verwirrung. Der *comTLBrowser* unterscheidet konsequent zwischen Klassen und Schnittstellen. Weitere Informationen zu diesen beiden Tools finden Sie in Kapitel 2.28.7

2.11.1 Standardschnittstelle einer COM-Klasse

Jede Klasse besitzt eine Standardschnittstelle, die verwendet wird, wenn keine Schnittstelle explizit verlangt wird. Leider verbirgt der Microsoft Objektkatalog (vgl. Kapitel 2.28) diese Standardschnittstelle vor dem Entwickler. Einige Sprachen (z.B. Visual Basic) fragen bei der Objektaktivierung immer nach dieser Standardschnittstelle.



Die Bezeichnung *Standardschnittstelle einer Klasse* sollte nicht mit den sogenannten *COM-Standardardschnittstellen* (siehe Kapitel 2.15) verwechselt werden. Viele Mechanismen basieren auf der Implementierung bestimmter, von Microsoft vorgegebener Schnittstellen (z.B. *IUnknown*, *IDispatch*, *IPersist*). Diese COM-Standardschnittstellen müssen in Bezug auf eine konkrete Klasse keineswegs zwingend die Standardschnittstelle dieser Klasse sein.

2.11.2 Namensgebung

IID Jede Schnittstelle besitzt eine GUID, die Interface Identifier (IID) genannt wird, sowie einen textlichen Namen.

Namensgebung Bei der Namensgebung ist es üblich, die Namen der Schnittstellen mit einem großen I beginnen zu lassen. Sofern die Interfacenamen vor Attributen und Methoden genannt werden, werden diese durch zwei Doppelpunkte getrennt vorangestellt (z.B.: *ISchnittstelle1::Methode()*). Die Standardschnittstelle sollte den Namen der Klasse mit einem vorangestellten großen I tragen (Beispiel: die Klasse *File* verfügt über die Standardschnittstelle *IFile*).

Mit Visual Basic 6.0 erstellte Komponenten bilden den Namen der Standardschnittstelle einer Klasse mit einem vorangestellten Unterstrich aus dem Klassennamen (Bsp.: *File* hat die Standardschnittstelle *_File*). In der Entwicklungsumgebung der Visual Basic-Vollversion werden die führenden Unterstriche jedoch verborgen, so dass auch hier der Unterschied zwischen Klasse und Schnittstelle verwischt wird. Sichtbar sind alle Schnittstellen im *comTLBrowser* (siehe Kapitel 2.28.8).

2.11.3 Schnittstellen in der Registry

Schnittstellen werden ebenfalls in der Registry verzeichnet. Unterhalb von HKEY_CLASSES_ROOT\INTERFACE befindet sich eine Liste der Interface-IDs (IIDs) aller installierten Komponenten. Zu einem Interface werden in der Registry nur wenige Informationen abgelegt:

Interface-IDs

- die CLSID der zugehörigen Proxy- und Stubklasse
- optional die Anzahl der Methoden, die die Schnittstelle definiert
- optional die LibID der zugehörigen Typbibliothek

2.11.4 Virtuelle Tabellen

In der binären Form sind Schnittstellen die aus der C++-Welt bekannten »Tabellen virtueller Funktionen« (kurz: vTable oder VTBL). Eine vTable-Struktur ist eine Liste von Zeigern auf die Implementierung von Funktionen (vgl. folgende Abbildung). Zwischen dem Zeiger auf eine Schnittstelle und der vTable liegt eine weitere Zwischenstufe. Das, was der COM-Client als Schnittstellenzeiger besitzt, ist nicht der Zeiger auf die vTable, sondern ein Zeiger auf einen Zeiger auf eine vTable. Die Position einer Methode innerhalb der vTable wird über einen Funktionsoffset ermittelt, der die relative Position innerhalb der vTable angibt.

Liste von Zeigern

Die vTables in COM entsprechen den vTables des Microsoft C++-Compilers (vgl. [GRU00], Seite 262). Eine vTable ist das bestimmende Element des Binärstandards von COM. Diese vTable-Struktur macht COM programmiersprachenunabhängig, da jede Sprache COM nutzen kann, die diese vTable-Struktur verwenden kann.

Binärstandard

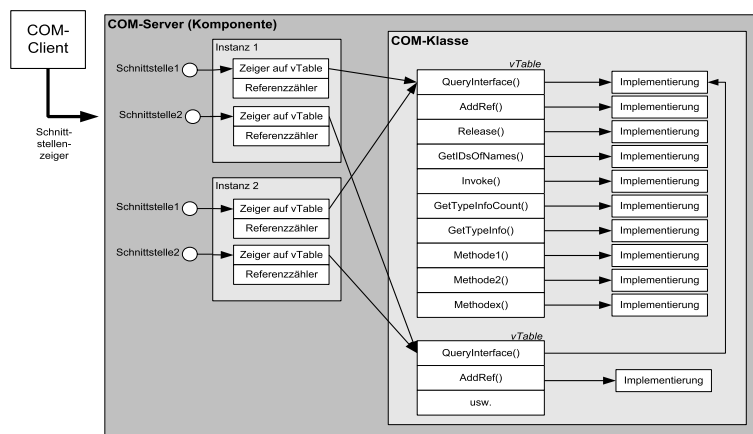


Abbildung 2.9:
Interner Aufbau
einer Komponente
aus vTables

2.11.5 Die Grauzone zwischen Klasse und Schnittstelle

Klasse versus Schnittstelle	Während es in der Komponententheorie und auch in der COM-Spezifikation eine klare Abgrenzung zwischen Klasse und Schnittstelle gibt, wird dieser Unterschied in der Praxis der COM-Programmierung stark verwischt. Dabei muss man zwei Blickrichtungen unterscheiden: die Sicht der Implementierung des COM-Servers und die Sicht der Implementierung des COM-Clients.
	Sicht des Servers
Sicht des Servers	Von der Implementierung der Komponente aus betrachtet gibt es in der Regel nur Klassen. Programmiersprachen wie C, C++, Java und Visual Basic kennen nämlich das Konzept der expliziten Schnittstellendefinition nicht. Die Sprachen verfügen nicht über ein Schlüsselwort, um eine Schnittstelle explizit zu definieren. In den Sprachen können nur Klassen definiert werden, deren Mitglieder genau eine Schnittstelle bilden. Mehrfachschnittstellen können nicht definiert werden. Dennoch existiert in jeder dieser Sprachen die Möglichkeit, im Zuge der Erzeugung von COM-Komponenten Mehrfachschnittstellen zu generieren. Schnittstellen werden dabei als abstrakte Basisklassen definiert (d.h. also als Klassen mit Funktionsrümpfen, aber ohne Implementierung). Eine Klasse x , die Mehrfachschnittstellen besitzen soll, erbt von einer oder mehreren dieser abstrakten Basisklassen (y und z). Die Klassen y und z bilden dann jeweils eine Schnittstelle in der Klasse x . Dieser umständliche Weg wurde gewählt, um Programmiersprachen COM-implementierungsfähig zu machen, unter der Bedingung, dass der Sprachumfang nicht oder nur gering erweitert werden muss.
	Sicht des Clients
Sicht des Clients	Der COM-Client dagegen sieht nichts von der Implementierung einer Komponente. Er sieht lediglich das binäre Format der Komponente und ggf. eine Typinformation (vgl. Kapitel 2.13). Diese sprachunabhängigen Darstellungsformen kennen sehr wohl das Konzept von expliziten Schnittstellen und Mehrfachschnittstellen. Programmiersprachen besitzen zur Verwaltung von Objekten Objektvariablen oder Objektzeiger. Diese zeigen bei der Arbeit mit COM aber nicht auf Objekte, sondern auf einzelne Schnittstellen. Dabei ist jeweils nur eine Schnittstelle (d.h. eine bestimmte Untermenge der Gesamtfunktionalität) zu einem bestimmten Zeitpunkt im Zugriff. Um auf andere Schnittstellen zugreifen zu können, muss die Schnittstelle gewechselt werden (<i>Interface Casting</i>). Sofern nicht die Programmiersprache ein besonderes Konstrukt für Schnittstellenzeiger anbietet, merkt der Programmierer nicht, ob er mit Objekten oder Schnittstellen arbeitet. Er merkt das allenfalls am Namen.

Schnittstellenwechsel (Interface Casting)

COM unterstützt den Wechsel der Schnittstelle mit dem Standardmechanismus `IUnknown::QueryInterface()` (vgl. Kapitel 2.15). Einige Programmiersprachen können diesen Mechanismus bedienen, andere – z.B. Skriptsprachen – nicht. Eine Klasse hat immer die Möglichkeit, neben dem vorgeschriebenen Standardmechanismus zum Schnittstellenwechsel auch einen benutzerdefinierten Mechanismus anzubieten. Dabei bietet die Klasse ein Attribut oder eine Methode an, die einen Zeiger auf die andere Schnittstelle liefert (aus der Sicht des Programmierers ist der Zeiger – wie oben erläutert – ein Zeiger auf ein Objekt, nicht auf eine Schnittstelle). Dieses Verfahren kann selbstverständlich jeder COM-Client – also auch eine Skriptsprache – nutzen, der überhaupt in der Lage ist, auf Attribute und Methoden von COM-Objekten zuzugreifen. Für den Programmierer der Komponente bedeutet dies jedoch zusätzlichen Aufwand. Die Komponente *Collaboration Data Objects* (CDO) in der Version 3.0 besitzt einen solchen benutzerdefinierten Mechanismus zum Schnittstellenwechsel.

Schnittstellenwechsel

2.11.6 Mangelnde Selbsterkenntnis bei den Schnittstellen

Den Namen der Klasse, zu der das Objekt gehört, kennt eine COM-Schnittstelle nicht. Die Abbildung der 1-zu-n-Beziehung zwischen Klasse/Objekt und Schnittstellen kann nicht von einer Komponente erfragt werden. Wenn ein COM-Client also von einem bestehenden Objekt einen Schnittstellenzeiger bekommt, dann kann der Client nicht ohne weiteres erkennen, ob es sich dabei um den Zeiger auf eine andere Schnittstelle des gleichen Objekts oder um den Zeiger auf eine Schnittstelle eines anderen Objekts handelt. Der Client, der den `IUnknown`-Standardmechanismus zum Schnittstellenwechsel verwendet, könnte höchstens den Typ des neuen Schnittstellenzeigers dazu verwenden, auf einem anderen Schnittstellenzeiger mit `IUnknown::QueryInterface()` nach diesem Typ zu fragen. Sofern der dann ermittelte Schnittstellenzeiger mit dem zuvor von einem Objekt gelieferten Schnittstellenzeiger identisch ist, weiß der COM-Client, dass es sich um zwei Schnittstellen ein und desselben Objekts handelt. Dies ist ein komplizierter Weg, der zudem voraussetzt, dass der COM-Client den Typ des neuen Schnittstellenzeigers kennt.

Schnittstellen ohne Zusammenhang

Sie werden sich jetzt fragen, warum es denn nachteilig ist, dass eine Schnittstelle nichts über die Klasse, die sie implementiert, weiß. Das Problem liegt nicht auf technischer Ebene, denn COM selbst benötigt den Zusammenhang zwischen Klasse und Schnittstelle nicht. Das Problem liegt allein bei dem Menschen, der einen COM-Client für diese Komponente schreiben will. Ein Programmierer muss wissen, über welche COM-Klasse eine bestimmte COM-Schnittstelle erreichbar ist, sonst erhält er nämlich keinen Schnittstellenzeiger zu dieser Schnittstelle.

Der Mensch ist der Leidtragende

Typinformationen

Typinformationen Die COM-Schöpfer haben mit den Typinformationen, die in Kapitel 2.13 vorgestellt werden, eine Möglichkeit vorgesehen, den Zusammenhang zwischen Klassen und Schnittstellen auf formelle Art zu beschreiben. Leider ist es weder zwingend, alle Klassen und Schnittstellen vollständig zu beschreiben, noch überhaupt Typinformationen zu liefern. So beschränken sich viele Typinformationen darauf, die Schnittstellen der instanziiierbaren Klassen zu listen; alle anderen Schnittstellen sind oft ohne die Klassennamen definiert. Leidtragender ist der Komponentennutzer, der im Zweifel durch Ausprobieren herausbekommen muss, wie er einen Zeiger zu einer bestimmten Schnittstelle bekommt.



Spätestens an diesem Punkt ist eine absolut saubere Dokumentation einer COM-Komponente notwendig, da es sonst zu großen Begriffsverwirrungen kommt. Leider findet man solche Dokumentationen heute noch sehr selten mit der Konsequenz, dass die Einarbeitung in die meisten Komponenten unnötig verlängert wird.

Registry

Die Registry hilft nicht weiter

Die Registry stellt zwar mit dem Unterschlüssel `INTERFACES` einer `CLSID` grundsätzlich eine Möglichkeit bereit, die implementierten Schnittstellen einer Klasse zu listen, jedoch wird von dieser Möglichkeit in der Praxis kein Gebrauch gemacht, zumal dies ja auch nur für die CoClasses von Nutzen wäre, da andere COM-Komponenten nicht in der Registry verzeichnet werden. Ein Client könnte auch versuchen, alle in der Registry gelisteten IIDs durch Ausprobieren mit `IUnknown::QueryInterface()` gegen eine Instanz einer bestimmten Klasse zu testen. Dies ist jedoch eine sehr langwierige Aufgabe (zu `QueryInterface()` siehe Kapitel 2.15.1).

Schnittstellen in Skriptsprachen

Skriptsprachen sehen nur Klassen

Das Ganze wird etwas einfacher, wenn man Sprachen betrachtet, die den Wechsel der Schnittstelle mit dem COM-Standardmechanismus gar nicht unterstützen. Zu diesem Typus von Programmiersprachen gehören die bislang vorhandenen ActiveX-Skriptsprachen. Diese Sprachen können immer nur das Standardinterface einer COM-Klasse sehen. Da sie keine weiteren Schnittstellen sehen können, ist aus der Blickrichtung dieser Sprachen ein Objekt das gleiche wie eine Schnittstelle. Die Standardschnittstelle muss zudem eine bestimmte Form haben; sie muss nämlich eine direkte Implementierung der Standardschnittstelle `IDispatch` sein oder aber von `IDispatch` abgeleitet sein. Der dahinterstehende Mechanismus heißt *COM-Automation* (vgl. Kapitel 2.14 und 2.15.2).

Interface Casting in Skriptsprachen

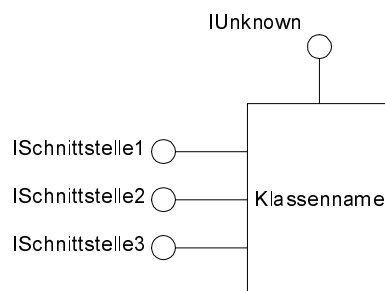
Interface Casting in Skriptsprachen

Sofern alle beteiligten Schnittstellen `IDispatch` unterstützen, können auch Skriptsprachen dazu gebracht werden, die Schnittstelle zu wechseln. Es gibt dazu zwei Möglichkeiten:

- Das Objekt selbst stellt einen eigenen Mechanismus bereit, der dem Aufrufer einen Zeiger auf eine andere Schnittstelle liefert. Aus der Sicht der Skriptsprache besteht kein Unterschied darin, ob der Zeiger ein Zeiger auf ein neues Objekt oder ein Teil des alten Objekts ist. Wie bereits erwähnt ist dies in CDO 3.0 genau auf diese Weise implementiert.
- Active Scripting-Sprachen können mit einer einzigen Zusatzkomponente durchaus dazu gebracht werden, bei allen COM-Klassen mit Mehrfachschnittstellen die Schnittstelle zu wechseln (vgl. [WES99c]). Diese Zusatzkomponente muss dann aber auf allen Systemen, auf denen ein derart geschriebenes Skript eingesetzt werden soll, existieren.

2.11.7 Bullet-and-Stick-Diagramme

Wenn die Zuordnung zwischen Klassen und Schnittstellen grafisch dargestellt werden soll, werden sogenannte Bullet-and-Stick-Diagramme (siehe folgende Abbildung) verwendet. Solche Bullet-and-Stick-Diagramme können beispielsweise mit der Diagramming-Software Microsoft Visio erstellt werden. Leider gibt es in der üblichen grafischen Notation keine Unterscheidung zwischen Klasse und Objekt: Beide werden als ein Kasten dargestellt. Da – wie oben geschildert – aus der Sicht des Scriptings die Schnittstellen eine untergeordnete Rolle spielen, wird in der Darstellung der Objektmodelle auf die explizite Darstellung der Schnittstellen verzichtet.



Grafische Darstellung

Abbildung 2.10:
Bullet-and-Stick-Diagramm einer COM-Klasse mit Standardschnittstelle IUnknown und drei benutzerdefinierten Schnittstellen

2.12 Klassen-Mitglieder

Eine Klasse bzw. eine Schnittstelle besitzt Mitglieder. In COM gibt es drei verschiedene Typen: Attribute, Methoden und Ereignisse.

Auch wenn COM-Clients einen Unterschied zwischen Attributen, Methoden und Ereignissen sehen, so sind doch alle diese Mitglieder intern in Form von Methoden implementiert. So ist der Attributzugriff aus der Sicht von COM der Zugriff auf ein Paar von zwei Methoden, wobei die eine den Attributwert ausliest (die »Get«-Methode) und die andere den Attributwert setzt (die »Set«-Methode). Dies ist die einzig mögliche Implementierung, da sonst

Die Klassenmitglieder näher betrachtet

der COM-Client immer die Möglichkeit haben müsste, direkt in den Adressraum des COM-Servers zu schreiben.

Attribute werden in Methoden umgesetzt

Entwicklungsumgebungen wie Visual Studio bieten dem Programmierer dennoch eine Attribut-Sicht auf Basis von Typinformationen (vgl. Kapitel 2.13). Vom Programmierer kodierte Attribute werden automatisch in ein passendes Paar von Methoden umgewandelt. Wenn im Folgenden also nur noch von Methoden die Rede ist, sind die Attribute keineswegs vergessen worden.

Attribute mit Parametern

Item-Attribut mit Parametern

Verwirrend ist es für den Entwickler, Attribute mit Parametern zu sehen, denn es gibt in der objektorientierten Lehre normalerweise keine Attribute mit Parametern. In COM ist dies jedoch möglich und an einer Stelle auch sehr gebräuchlich: Collections (Objektmengen) enthalten ein Attribut `Item` für den direkten Zugriff auf die enthaltenen Objekte (im Einzelfall auch auf die elementaren Datentypen). `Item` hat dabei einen Parameter zur Spezifikation des gewünschten Objekts anhand eines Schlüssels. Vorzufinden sind sowohl numerische als auch alphanumerische Schlüssel. Sehen Sie `Item` eher wie ein Array, dann können Sie das Vorgehen besser mit dem objektorientierten Weltbild vereinen.



Reduzierung der Round Trips

Bei prozessexternen Komponenten besteht eine enorme Optimierungsmöglichkeit darin, die Anzahl der Methodenaufrufe zu minimieren: Da auch jeder Zugriff auf ein Attribut in einen Methodenaufwurf umgesetzt wird, ist es günstiger, eine einzige Methode mit n Parametern aufzurufen, anstatt zunächst n Attribute zu setzen und dann eine Methode ohne Parameter aufzurufen. Dieser Aspekt ist jedoch irrelevant, wenn COM-Client und COM-Server im gleichen Adressraum liegen.

Ereignisse

Events

COM unterstützt das Aussenden von Ereignissen durch COM-Server und die Behandlung von Ereignissen durch COM-Clients. COM-Events werden beispielsweise eingesetzt zur Meldung von Zwischenständen bei länger andauernden Operationen oder zur Realisierung asynchroner Methodenaufrufe, bei denen der COM-Client direkt nach dem Aufruf einer Methode die Kontrolle zurückerhält. Über das Ergebnis der von ihm aufgerufenen Methode wird er nach Abschluss der Verarbeitung per Ereignis informiert.

Publisher, Subscriber und Event Sink

In Zusammenhang mit COM-Events wird folgende Terminologie verwendet:

- *Event Publisher* (Ereignisanbieter) ist ein Objekt, das ein Ereignis aussendet. Man kann den Publisher auch den Event Server nennen.
- *Event Subscriber* (Event Client, Ereigniskonsument) ist ein Objekt, das auf das Eintreten eines Ereignisses in einem Event Publisher wartet.

Typinformationen

- *Event Sink* ist die Ereignisbehandlungsroutine im Event Subscriber, die in Reaktion auf das Ereignis Programmcode ausführt.

Die Realisierung von Ereignissen basiert ebenfalls auf Methoden und der Unterstützung bestimmter Standardschnittstellen im COM-Client und im COM-Server. Der COM-Client liefert dem COM-Server einen Zeiger auf eine bestimmte Schnittstelle. Wenn im COM-Server ein Ereignis ausgelöst wird, ruft der COM-Server eine Methode der ihm gelieferten Schnittstelle auf. In diesem Zusammenhang sind auch Verbindungspunkte und die Standardschnittstellen `IConnectionPoint` und `IConnectionPointContainer` zu nennen; sie sollen hier nicht näher betrachtet werden.

Connection Points

COM-Clients, die Events abfangen wollen, müssen selbst COM-Komponenten sein. Scripting Hosts unterstützen in der Regel nur Ereignisse ihrer Intrinsic Objects. Eine Ausnahme bilden bisher nur der Windows Script Host (WSH) und der System Scripting Host (SSH), die Ereignisse beliebiger automationsfähiger Komponenten abfangen können.

**Event Subscriber
beim Scripting**

Der *Connection Point*-Mechanismus ist ein Ereignismechanismus mit fester Kopplung, bei dem sowohl COM-Client als auch COM-Server zu allen Zeiten ausgeführt werden müssen. Lose gekoppelte Ereignisse werden erst durch COM+ in Windows2000 möglich.

COM+-Events

2.13 Typinformationen

Typinformationen sind eine formale Beschreibung der Klassen mit ihren Schnittstellen und deren Mitgliedern. Typinformationen umfassen insbesondere:

Typinformationen

- Namen der Klassen
- Namen der Schnittstellen
- die von einer Klasse implementierten Schnittstellen
- Namen der Mitglieder einer Schnittstelle
- Attribute mit ihrem Datentyp
- Methoden und Ereignisse mit den zugehörigen Parametern sowie dem Rückgabewert (jeweils inkl. Datentyp)

Für das statische Binden werden Typinformationen benötigt, um die Einsprunganadressen zu ermitteln. Das dynamische Binden kommt dagegen ohne Typinformationen aus. Gleichwohl bilden Typinformationen auch beim dynamischen Binden die Grundlage für die Eingabehilfen, welche von modernen Entwicklungsumgebungen (z.B. Visual Studio 6.0) zur Verfügung gestellt werden. Schließlich kann ein COM-Client eine Komponente durch Typinformationen auch zur Laufzeit erforschen.



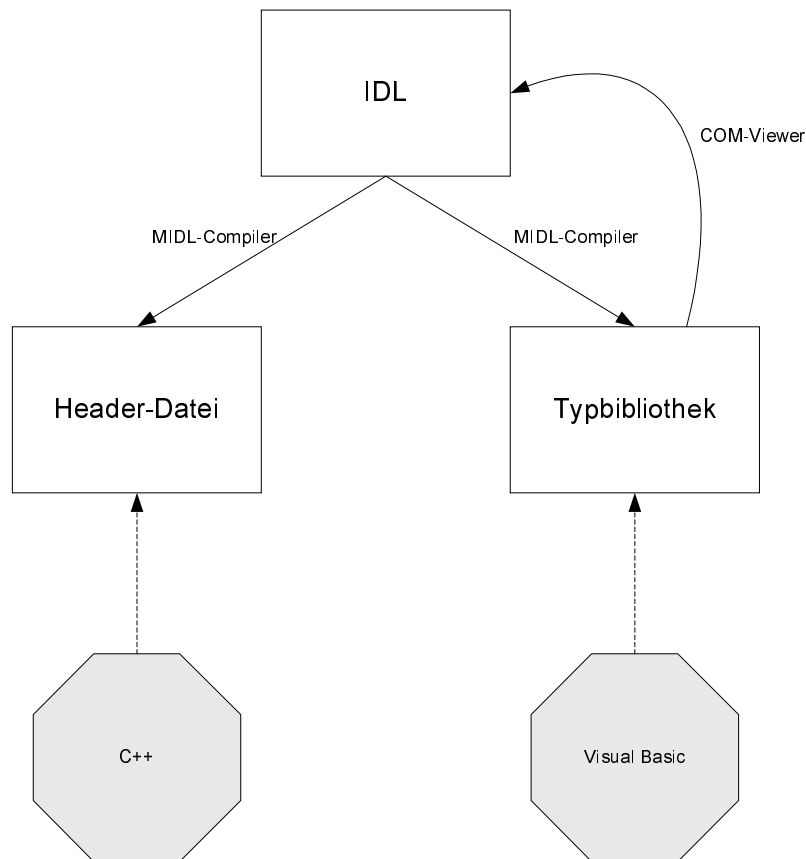
Leider sind aus der Sicht von COM selbst nicht alle oben genannten Typinformationen zwingend notwendig. Sie sind allesamt hilfreich für den Menschen, der eine Komponente nutzen will. Daran denken jedoch viele Komponentenentwickler nicht, die nur unvollständige Typinformationen zu ihren Komponenten liefern.

Speicherformen

Speicherformen Es gibt drei Formen der Speicherung von Typinformationen: IDL-Dateien, Header-Dateien und Typbibliotheken. Ein COM-Client bedient sich entweder einer Header-Datei oder einer Typbibliothek, um an Typinformation zu kommen. Eine IDL-Datei kann er nicht verwenden. IDL-Dateien werden eingesetzt, um daraus Header-Dateien und Typbibliotheken automatisch zu erzeugen.

Header-Dateien werden von C/C++ verwendet und enthalten die notwendigen Deklarationen in Form von Quellcode im C/C++-Stil. Visual Basic und Skriptsprachen verwenden Typbibliotheken.

Abbildung 2.11:
Speicherformen für
Typinformationen



2.13.1 Interface Definition Language (IDL)

Es gibt eine formale Sprache zur Beschreibung von Typinformationen – die Interface Definition Language (IDL); sie wurde von der Open Software Foundation (OSF) entwickelt und garantiert eine programmiersprachenunabhängige Definition von Schnittstellen. IDL basiert auf der Syntax der Programmiersprache C. Microsoft verwendet eine erweiterte Form, die Microsoft Interface Definition Language (MIDL). Eine ältere Bezeichnung dafür ist auch Object Definition Language (ODL). Eine IDL-Datei kann Informationen über alle kompletten Komponenten enthalten. Sie hat die Dateierweiterung .IDL. IDL-Dateien können problemlos mit Visual C++ erstellt werden.

IDL

Die Ausdrucksfähigkeit von IDL reicht aus, um das Schema einer Komponente komplett zu beschreiben. Allerdings werden in der COM-Praxis die Typinformationen in der Regel nur dazu benutzt, die Schnittstellen zu beschreiben. Die Typinformationen beschreiben nur das, was der Client von der Komponente sehen soll. Die in der Komponente enthaltenen Objektmodelle sind nicht immer anhand der Typinformationen erkennbar. Grundsätzlich jedoch ermöglichen Typinformationen unter Einsatz eines entsprechenden Tools wie dem Microsoft Objektkatalog die Navigation in den Objektmodellen einer Komponente.

Typinformation,
Schema und
Objektmodell

Beispiele

Es folgen einige IDL-Beispiele. Das erste Beispiel zeigt die Definition einer von IDispatch abgeleiteten Schnittstelle mit Namen IWSHShell.

Beispiel 1

```
[
    odl,
    uuid(F935DC21-1CF0-11D0-ADB9-00C04FD58A0B),
    helpstring("Shell Object Interface"),
    hidden,
    dual,
    oleautomation
]
interface IWSHShell : IDispatch {
    [id(0x00000064), propget]
    HRESULT SpecialFolders([out, retval] IWSHCollection**
out_Folders);
    [id(0x000000c8), propget]
    HRESULT Environment(
        [in, optional] VARIANT* Type,
        [out, retval] IWSHEnvironment** out_Env);
    [id(0x000003e8)]
    HRESULT Run(
        [in] BSTR Command,
        [in, optional] VARIANT* WindowStyle,
        [in, optional] VARIANT* WaitOnReturn,
```

COM-Theorie

```
[out, retval] int* out_ExitCode);
[id(0x000003e9)]
HRESULT Popup(
    [in] BSTR Text,
    [in, optional] VARIANT* SecondsToWait,
    [in, optional] VARIANT* Title,
    [in, optional] VARIANT* Type,
    [out, retval] int* out_Button);
[id(0x000003ea)]
HRESULT CreateShortcut(
    [in] BSTR PathLink,
    [out, retval] IDispatch** out_Shortcut);
[id(0x000003ee)]
HRESULT ExpandEnvironmentStrings(
    [in] BSTR Src,
    [out, retval] BSTR* out_Dst);
[id(0x000007d0)]
HRESULT RegRead(
    [in] BSTR Name,
    [out, retval] VARIANT* out_Value);
[id(0x000007d1)]
HRESULT RegWrite(
    [in] BSTR Name,
    [in] VARIANT* Value,
    [in, optional] VARIANT* Type);
[id(0x000007d2)]
HRESULT RegDelete([in] BSTR Name);
};
```

Listing 2.1: IDL-Beispiel 1: Eine IDL-Schnittstellendefinition

Beispiel 2 Das zweite Beispiel demonstriert Versionierung: IWSHShell2 ist eine neuere Version von IWSHShell. IWSHShell2 erbt daher von IWSHShell.

```
[
    odl,
    uuid(24BE5A30-EDFE-11D2-B933-00104B365C9F),
    helpstring("Shell Object Interface"),
    hidden,
    dual,
    oleautomation
]
interface IWSHShell2 : IWSHShell {
    [id(0x00000bb8)]
    HRESULT LogEvent(
        [in] VARIANT* Type,
        [in] BSTR Message,
        [in, optional] BSTR Target,
```


Typinformationen

```
        [out, retval] VARIANT_BOOL* out_Success);
[id(0x00000bc2)]
HRESULT AppActivate(
    [in] VARIANT* App,
    [in, optional] VARIANT* Wait,
    [out, retval] VARIANT_BOOL* out_Success);
[id(0x00000bc3)]
HRESULT SendKeys(
    [in] BSTR Keys,
    [in, optional] VARIANT* Wait);
};
```

Listing 2.2: IDL-Beispiel 2: Eine abgeleitete IDL-Schnittstellendefinition

Im dritten Beispiel wird eine Klasse `WSHShell` definiert, die genau eine Schnittstelle `IWSHShell2` hat. **Beispiel 3**

```
[
    uuid(72C24DD5-D70A-438B-8A42-98424B88AFB8),
    helpstring("Shell Object")
]
coclass WSHShell {
    [default] interface IWSHShell2;
};
```

Listing 2.3: IDL-Beispiel 3: Eine IDL-Klassendefinition mit einer Schnittstelle

Eine Klasse mit mehreren Schnittstellen sieht man im vierten Beispiel. Eine der Schnittstellen ist dabei die Standardschnittstelle. **Beispiel 4**

```
[
    uuid(DFC33154-69A3-43EC-9EE7-BB12C9609E1F),
    version(1.0)
]
coclass AllInOne {
    [default] interface _AllInOne;
    interface _IPrinter;
    interface _IFax;
    interface _IScanner;
};
```

Listing 2.4: IDL-Beispiel 4: Eine IDL-Klassendefinition mit vier Schnittstellen

Mit Hilfe des Werkzeugs COM-Viewer können die im Folgenden beschriebenen Typbibliotheken in IDL zurückgewandelt werden.



2.13.2 Typbibliotheken

TypeLibs Eine Typbibliothek (engl. Type Library, kurz: TypeLib) ist die kompilierte Version einer IDL-Datei, die binär gespeicherte Typinformationen enthält. Eine Typbibliothek kann als eigenständige Datei (Dateiextensionen .TLB und .OLB, zum Teil auch .OCA oder .RLI) oder als Bestandteil einer .DLL, .OCX oder .EXE-Datei realisiert werden. Eine derartige Datei kann mehrere Typbibliotheken enthalten, allerdings unterstützen nicht alle Umgebungen dieses Feature. Der Microsoft Objektkatalog unterstützt dieses Feature nicht.

Anwendungsgebiete Typbibliotheken sind für das Funktionieren von COM nicht unbedingt notwendig, bieten jedoch wertvolle Zusatzdienste. Sie wurden für Visual Basic geschaffen, um dieser Sprache, die keine Header-Dateien einbinden kann, das frühe Binden zu ermöglichen. Ebenso basieren viele Eingabehilfen und Werkzeuge (wie der Microsoft Objektkatalog und der in diesem Buch vorgestellte comTLBrowser – siehe Kapitel 2.28.8) auf Typbibliotheken. Leider können Typbibliotheken nicht den kompletten IDL-Sprachumfang wiedergeben.

Konvertierungen **Werkzeuge** Header-Dateien und Typbibliotheken können mit Hilfe des *Microsoft IDL-Compilers (MIDL-Compiler)* aus IDL-Dateien erzeugt werden. Typbibliotheken können mit Hilfe des Microsoft COM-Viewers in IDL zurückverwandelt werden. Der COM-Viewer enthält dazu eine spezielle Funktion mit dem Namen »View TypeLib«, die ein Fenster mit dem Titel »TypeLib Viewer« öffnet.

Abbildung 2.12:
Anzeige einer
Typbibliothek mit
dem TypeLib Viewer
des COM-Viewers



Konstantenlisten

Eine Typbibliothek kann neben den Definitionen von Klassen, Schnittstellen und deren Mitgliedern auch symbolische Konstanten enthalten. Symbolische Konstanten (auch *Konstantenbezeichnungen* genannt) sind Zeichenketten, die einen beliebigen Wert repräsentieren (in der Regel eine Zahl) und als Alias für diesen Wert benutzt werden. Während der eigentliche (numerische) Konstantenwert aus der Sicht des Systems auf das Wesentliche fokussiert, nämlich sich von anderen Werten eindeutig zu unterscheiden, dabei aber wenig Speicherplatz zu verbrauchen, richtet sich eine symbolische Konstante an das Erinnerungsvermögen des Programmierers. Ein Mensch kann sich einen sprechenden Namen besser merken als Zahlen oder Abkürzungen. So repräsentiert beispielsweise die symbolische Konstante `vbYes` die Zahl 6, die die Visual Basic-Funktion `MsgBox()` zurückliefert, wenn der Benutzer auf den Button JA geklickt hat. Der Programmcode wird durch symbolische Konstanten lesbarer.

Symbolische
Konstanten

Symbolische Konstanten sind in Konstantenlisten (*Constant Enumerations*) angeordnet, was sinnvoll ist, da es in der Regel einen Block zusammenhängender Konstantenwerte gibt, die in einer bestimmten Methode oder in einem bestimmten Attribut verwendet werden. Dabei sind die Konstanten aus einer Konstantenliste nicht immer als alternative Werte zu sehen. Wenn ein Methodenparameter bzw. ein Attribut Mehrfachwerte erlaubt, spricht man von einem Flag. Ein solches Flag verwendet beispielsweise die VB-Funktion `MsgBox()` für den Parameter `Buttons`.

Constant
Enumerations

Voraussetzung für die Nutzung symbolischer Konstanten ist, dass die jeweilige Umgebung Typbibliotheken auslesen kann. Leider verfügen noch nicht alle Scripting Hosts über diese Fähigkeit. In diesem Fall bleibt dem Entwickler nichts anderes übrig, als die symbolischen Konstanten selbst im Quellcode zu definieren. Während Sie mit dem Microsoft Objektkatalog jede symbolische Konstante nur einzeln in die Zwischenablage übernehmen können, ermöglicht der `comTLBrowser` die Übernahme ganzer Konstantenlisten.

Konstanten-
definitionen in
Typbibliotheken

Typbibliotheken in der Registry

Die Informationen zu den Typbibliotheken befinden sich unterhalb des Schlüssels `HKEY_CLASSES_ROOT/TYPELIB`. Dort sehen Sie eine Liste der LibIDs der installierten Typbibliotheken. Es können mehrere Versionen einer Typbibliothek auf einem System parallel bestehen, wobei die Versionsnummer die erste Ebene unterhalb der LibID bildet. Unterhalb der jeweiligen Versionsnummer finden Sie im Standardattribut des Schlüssels `/Win32` den Pfad zur Typbibliothek. Die zweite wichtige Information ist das Standardattribut der Versionsnummer. Hier steht eine kurze textliche Beschreibung der Typbibliothek, die als Friendly Name (d.h., es ist eine für einen Menschen bestimmte Information, vgl. Kapitel 2.10.4) der Typbibliothek dient. Dieser String wird in der IDL als sogenannter *Helpstring* definiert.

TypeLib-Schlüssel

Der Mechanismus der COM-Automation (siehe Kapitel 2.14 und 2.15.2) stellt Methoden bereit, um Typinformationen aus Typbibliotheken zur Laufzeit zu ermitteln. Dadurch kann ein COM-Client zur Laufzeit den COM-Server erforschen und »interessante« Methoden aufrufen.

Dynamische
Typinformationen

Abbildung 2.13:
Aufbau der TypeLib-
Informationen in der
Registry anhand der
Word-Komponente;
die interne Versions-
nummer der Typbib-
liothek entspricht
nicht immer der Pro-
duktversionsnum-
mer (hier 8.1 = 9.0)



2.14 Statischer Aufruf versus Automation

COM kennt – wie andere Verteilungsplattformen auch – zwei Formen des Methodenaufrufs:

Statischer Aufruf versus dynamischer Aufruf

- **Statischer Aufruf** (engl. Static Method Invocation): Bereits zur Zeit der Kompilierung ist festgelegt, welche Methoden aufgerufen werden.
- **Dynamischer Aufruf** (engl. Dynamic Method Invocation, COM-Automation): Die Entscheidung über den konkreten Methodenaufruf fällt erst zur Laufzeit. Dies ist immer dann nötig, wenn zur Zeit der Kompilierung nicht bestimmt werden kann, welche Methode aufgerufen werden soll. Dies ist insbesondere beim Einsatz von Polymorphismus gegeben und bei interpretierten Sprachen (wie Skriptsprachen), die nicht kompiliert werden.

Begriffe

Automation Server und Clients

Der dynamische Aufruf wird in COM *Automation* genannt. Ein COM-Server, der Automation unterstützt, ist ein *Automation Server*. Man spricht auch von automationsfähigen Klassen und automationsfähigen Komponenten, sofern eine Komponente automationsfähige Klassen unterstützt. Ein COM-Client, der Automation nutzt, heißt Automation Client oder Automation Controller. Frühere Begriffe, die Sie aber nicht mehr verwenden sollten, waren OLE Automation, OLE Server und OLE Client/OLE Controller.

Automation versus Automatisierung

Statt des Begriffs *Automation* werden Sie in anderen Büchern auch *Automatisierung* lesen. Im Englischen heißt die Technologie *Automation*. In diesem Buch wird durch Verwendung des Begriffs *Automation* eine Abgrenzung zum allgemeinen Begriff *Automatisierung* vollzogen. *Automatisierung* steht hier allgemein für die programmgesteuerte Ausführung von administrativen Aufgaben.

Der Begriff *Automation* stammt aus der Zeit, in der *Dynamic Data Exchange* (DDE) und die erste Version von OLE zum Datenaustausch bzw. zur Fernsteuerung von Office-Anwendungen wie Word und Excel verwendet wurden. Die heutige COM-Automation ist eine Weiterentwicklung der Techniken, die bereits bei OLE eingesetzt wurden. Der Name ist geblieben, obwohl COM-Automation längst in anderen Bereichen als nur in der Fernsteuerung von Office-Anwendungen eingesetzt wird.

Geschichte des Begriffs Automation

Binden

Binden bedeutet in COM die Zuordnung eines Methodenaufrufs zu einer Adresse in der vTable einer Schnittstelle (zum Begriff »Binden« siehe auch Anhang A). Der Aufruf ist der folgende Schritt. Ein statischer Aufruf erfordert statisches Binden (frühes Binden, Early Binding), ein dynamischer Aufruf erfordert dynamisches Binden (spätes Binden, Late Binding). Die Unterscheidung zwischen dynamischem und statischem Binden ist Standard in objektorientierten Sprachen.

Statisches und dynamisches Binden in COM

Beim *statischen Binden* kennt der COM-Client zur Entwicklungszeit die Position (*Offset*) einer Funktion innerhalb der vTable. Der Offset ist im Binärcode des COM-Clients festgelegt. Der statische Aufruf einer Methode erfolgt zur Laufzeit also über die Startadresse der vTable und den hinterlegten Offset. Die Startadresse der vTable der gewünschten Schnittstelle muss freilich auch beim statischen Binden zur Laufzeit bei der Komponente erfragt werden (vgl. den Abschnitt zu *IUnknown* im folgenden Kapitel). Aufgrund des direkten Zugriffs auf die vTable heißt das statische Binden auch *VTBL Binding*. Beim *dynamischen Binden* ist dagegen der Offset nicht bekannt. Die Bindung erfolgt durch verschiedene Mechanismen erst zur Laufzeit (vgl. den Abschnitt zu *IDispatch* in Kapitel 2.15).

Allerdings bildet COM durch die Konzeption der Mehrfachschnittstellen eine Ausnahme. Hier ist ein dynamischer Methodenaufruf mit dem Ziel, Polymorphismus zu realisieren, auch durch statisches Binden möglich.

Ausnahme in COM

Wenn durch einen Methodenaufruf auf zwei Instanzen unterschiedlicher COM-Klassen, die die gleiche Schnittstelle unterstützen, zugegriffen werden soll, muss der Bindungsvorgang nicht dynamisch erfolgen, da der Funktionsoffset (relativ zur vTable der Schnittstelle) zur Entwicklungszeit ermittelt werden kann. Die Bindung ist also statisch möglich; der Methodenaufruf ist dennoch als dynamisch anzusehen, da zur Entwicklungszeit noch nicht feststeht, welche Implementierung aus welcher der beiden Klassen aufgerufen werden wird. Sobald zwei Klassen zwar die gleichen Methoden, diese aber in unterschiedlichen Schnittstellen unterstützen, ist immer dynamisches Binden notwendig, da die Funktionsoffsets verschieden sind.

Hiermit haben Sie wieder ein Beispiel kennen gelernt, wie komplex die Welt der Objekte und Komponenten in COM sein kann.

Bewertung

Der Vorteil des statischen Aufrufs durch dynamisches Binden ist eindeutig die sehr viel höhere Geschwindigkeit, da der COM-Client die Funktion direkt aufrufen kann. Allerdings ist der statische Aufruf durch statisches Binden unflexibel: Ein COM-Client muss bereits zur Entwicklungszeit wissen, welchen Objekttyp er vor sich hat. Wenn Polymorphismus eingesetzt werden soll, ist statisches Binden normalerweise nicht möglich.

2.15 COM-Standardschnittstellen**Standard-schnittstellen**

Microsoft hat inzwischen rund 200 Standardschnittstellen definiert, einen Teil davon bereits im Rahmen der COM-Spezifikation. Standardschnittstellen sind die Basis zur Erbringung von COM-Diensten: Ein COM-Dienst kann nur dann funktionieren, wenn die daran beteiligten Komponenten einheitliche Schnittstellen bieten. Alle Nichtstandard-Schnittstellen heißen *benutzerdefinierte Schnittstellen*.

COM liefert für die Mehrheit der Standardschnittstellen (noch) keine Implementierung. Standardschnittstellen sind in der Regel nur abstrakte Schnittstellendefinitionen; sie müssen also in jedem COM-Client neu implementiert werden. Nur für ausgewählte Standardschnittstellen enthält die COM-Bibliothek Standardimplementierungen. Entwicklungsumgebungen und programmiersprachenspezifische Laufzeitumgebungen stellen auf verschiedene Weisen (z.B. die Visual Basic-Laufzeitumgebung, die Active Template Library in Visual C++) Implementierungen bereit, die übernommen werden können, so dass der Programmierer diese Schnittstellen nicht selbst kodieren muss.

IUnknown und IDispatch

Die wichtigste Standardschnittstelle ist IUnknown. Ob etwas eine COM-Klasse ist, kann daran festgemacht werden, ob sie IUnknown unterstützt oder nicht. Auf dem zweiten Platz dieser Hitliste liegt IDispatch. Danach kommt lange erst einmal nichts mehr, weshalb dieses Buch sich auf die Darstellung dieser beiden zum Verständnis von COM wichtigsten Schnittstellen beschränkt. Mehr über diese und andere Standardschnittstellen erfahren Sie bei [BOX99] und [EDD00]. Die Typinformationen zu den Standardschnittstellen IUnknown und IDispatch müssen nicht in jede IDL-Datei aufgenommen werden. Dazu kann die Typbibliothek STDOLE32.TLB importiert werden. Sie können die Datei im COM-Viewer (vgl. Kapitel 2.28) öffnen, um sich die »rohe« Definition der beiden Standardschnittstellen anzusehen.

2.15.1 Statisches Binden mit IUnknown**IUnknown**

IUnknown (IID= 00000000-0000-0000-C000-000000000046) ist die wichtigste Schnittstelle in COM. Mit Hilfe dieser Schnittstelle können die Zeiger auf die vTables anderer Schnittstellen ermittelt werden. Dabei ist IUnknown die Basis für den statischen Methodenaufruf (*VTBL Binding*). Im Bullet-and-Stick-Diagramm wird die Schnittstelle IUnknown üblicherweise an exponierter

COM-Standardschnittstellen

Stelle (oben) auf der Komponente dargestellt, um der Besonderheit dieser Schnittstelle Ausdruck zu verleihen. Es gilt:

- Jede COM-Klasse muss eine `IUnknown`-Schnittstelle besitzen.
- Alle COM-Schnittstellen sind von `IUnknown` abgeleitet. Das gilt sowohl für Standard- als auch für benutzerdefinierte Schnittstellen. Daher kann auf die Methoden von `IUnknown` von jeder Schnittstelle aus zugegriffen werden.

Die `IUnknown`-Methoden

Die drei von `IUnknown` definierten virtuellen Methoden müssen daher von jeder einzelnen COM-Schnittstelle bereitgestellt werden. Sie stehen am Anfang der vTable einer jeden Schnittstelle. Üblicherweise erfolgt die Implementierung aber nur einmal pro Klasse. Die Funktionszeiger jeder einzelnen Schnittstelle verweisen dann auf diese Implementierung.

- `QueryInterface()` ermöglicht die Anfrage an ein COM-Objekt, ob das Objekt eine bestimmte Schnittstelle unterstützt. Wenn das Objekt das Interface anbietet, liefert `QueryInterface()` einen Zeiger auf die Schnittstelle zurück, sonst den Wert 0.
- Jede Schnittstelle benötigt einen Referenzzähler, der angibt, wie oft eine Schnittstelle zum aktuellen Zeitpunkt benutzt wird. Der Aufruf von `AddRef()` erhöht diesen Referenzzähler um eins. `AddRef()` wird automatisch aufgerufen, wenn `QueryInterface()` Erfolg hat. Ein manueller Aufruf von `AddRef()` ist nur notwendig, wenn der COM-Client selbst einen Schnittstellenzeiger dupliziert und die Laufzeitumgebung der betreffenden Sprache den Aufruf von `AddRef()` nicht automatisch vornimmt. Der COM-Server kann die Zählung in diesem Fall nicht vornehmen, da er an der Duplizierung nicht beteiligt ist.
- Der Aufruf von `Release()` vermindert den Referenzzähler der Schnittstelle um eins. `Release()` entfernt nicht automatisch das Objekt aus dem Speicher. Ein Objekt kann erst gelöscht werden, wenn der Referenzzähler der Schnittstelle, für die `Release()` aufgerufen wurde, auf 0 steht und die Referenzzähler aller anderen Schnittstellen auch auf 0 stehen.

QueryInterface()

AddRef()

Release()

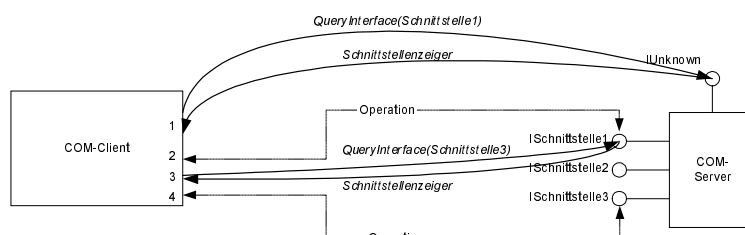


Abbildung 2.14:
Darstellung des
`QueryInterface()`-
Mechanismus

vTable-Fähigkeit Einschränkungen

Ergebnis der Instanziierung einer Klasse ist in COM immer, dass der COM-Client einen Zeiger auf die `IUnknown`-Schnittstelle des erzeugten Objekts erhält. Für C++- und Java-Programmierer ist `IUnknown` der zentrale Dreh- und Angelpunkt der COM-Programmierung. Das große Visual Basic und VBA nutzen zwar `IUnknown` (diese Fähigkeit hat Visual Basic seit Version 4.0), verbergen dies jedoch vor dem Programmierer. Skriptsprachen können `IUnknown` nicht benutzen, weil sie üblicherweise nicht »vTable-fähig« sind. vTable-Fähigkeit bedeutet die Unterstützung des komplizierten Mechanismus, um von einem Schnittstellenzeiger über einen Zeiger auf einen weiteren Zeiger zu den Methodenimplementierungen zu gelangen. Nicht vTable-fähig ist gleichbedeutend mit der Unfähigkeit, verschiedene Schnittstellen eines Objekts voneinander unterscheiden zu können und damit auch der Unfähigkeit, Mehrfachschnittstellen zu nutzen.



Das große Visual Basic und VBA können `IUnknown` aber nur nutzen, wenn es für die Klasse und ihre Schnittstellen eine Typbibliotheksdefinition gibt. In der Regel kompilieren Komponentenentwickler eine Typbibliothek leider nur für automationsfähige Komponenten.

2.15.2 Automation mit IDispatch

IDispatch Die bisherigen Ausführungen haben gezeigt, dass Skriptsprachen COM gar nicht nutzen könnten, wenn es nur `IUnknown` gäbe. Die Lösung heißt `IDispatch` (IID = 00020400-0000-0000-C000-000000000046). Diese Standardschnittstelle ist die Basis für die COM-Automation, auf der das Active Scripting (siehe Kapitel 2.23) beruht. `IDispatch` ermöglicht den Aufruf einer Methode über ihren Namen in Textform. Ein Automation-Client muss keinen anderen Schnittstellenzeiger als den von `IDispatch` kennen und keine anderen Funktionsoffsets als diejenigen der wenigen `IDispatch`-Methoden. Diese Informationen sind fest in der Laufzeitumgebung der jeweiligen Sprache hinterlegt. `IDispatch` ist eine Form des dynamischen (späten) Bindens.

Der IDispatch-Mechanismus

Genauer betrachtet funktioniert der Mechanismus folgendermaßen:

- ▶ Der Automation-Client beschafft sich zunächst einen Zeiger auf die Standardschnittstelle der Klasse. Dieser Mechanismus ist fest hinterlegt. Der Automation-Client kann nur mit diesem Schnittstellenzeiger arbeiten, sofern die Schnittstelle `IDispatch` entspricht oder zumindest von `IDispatch` abgeleitet ist.
- GetIDsOfNames()** ▶ Der Automation-Client ruft zunächst die `IDispatch`-Methode `GetIDsOfNames()` auf. Dabei übergibt der Client den Namen der gewünschten Methode in Form eines Strings. `GetIDsOfNames()` liefert dem Automation-Client eine sogenannte *Dispatch-ID* (*DispID*) zurück. Eine *DispID* ist eine

eindeutige Zahl für eine Methode innerhalb einer COM-Schnittstelle. DispIDs sind jedoch keine GUIDs und daher nicht über Schnittstellen hinweg eindeutig.

- Der Methodenaufruf erfolgt dann mit `IDispatch::Invoke()`, wobei die DispID und die Parameter für die Methode übergeben werden müssen. Die Parameter werden in Form eines *Arrays of Variants* übergeben.

Invoke()

Der Datentyp *Variant* geht auf die Sprache Visual Basic zurück. Ein Variant ist ein universeller Behälter für prinzipiell alle Datentypen. Eine Variant-Variable kann zur Laufzeit zu jedem beliebigen Zeitpunkt ihren Inhaltstyp ändern. Sie trägt auch dazu bei, COM für (Skript-) Sprachen zu vereinfachen, die keine strengen Datentypen kennen.

Variant

Typinformationen zur Laufzeit

Weiterhin bietet `IDispatch` zwei Methoden, um zur Laufzeit Typinformationen auszulesen:

- Mit der Methode `IDispatch::GetTypeInfoCount()` kann ermittelt werden, ob für die Klasse Typinformationen verfügbar sind. Der Rückgabewert 1 bedeutet, dass Typinformationen zur Verfügung stehen. 0 bedeutet, dass keine Typinformationen verfügbar sind.
- `IDispatch::GetTypeInfo()` liefert bei vorhandenen Typinformationen einen Zeiger auf die Schnittstelle `ITypeInfo()`, die ihrerseits dazu dient, Namen und Parameter der implementierten Mitglieder dieser Klasse zu ermitteln.

GetTypeInfoCount()

GetTypeInfo()

`GetTypeInfoCount()` und `GetTypeInfo()` sind für die Funktionsfähigkeit des `IDispatch`-Ablaufs nicht notwendig. Diese beiden Methoden werden nur dazu benutzt, dem Entwickler zur Entwicklungszeit Informationen über die verfügbaren Methoden und Attribute bereitzustellen. Die Eingabehilfen verschiedener Entwicklungsumgebungen greifen auf `GetTypeInfoCount()` und `GetTypeInfo()` zurück.

Da alle Schnittstellen von `IUnknown` abgeleitet sein müssen, besitzt natürlich auch eine `IDispatch`-Schnittstelle die drei `IUnknown`-Methoden. Diese sind aber für einen Automation-Client ohne Bedeutung. Natürlich können auch vTable-fähige Sprachen `IDispatch` verwenden. Sie sollten es jedoch nicht tun: Der geschilderte Weg des Methodenaufrufs über die `IDispatch`-Schnittstelle macht deutlich, dass COM-Automation wesentlich langsamer ist als VTBL Binding.



Binden mit bekannter DispID

Es gibt einen Weg, die Verwendung von `IDispatch` etwas zu beschleunigen: Wenn der COM-Client die DispID einer Methode bereits kennt, dann entfällt der Aufruf von `GetIDsOfNames()`, und `Invoke()` kann direkt verwendet werden. Interpretersprachen, die nur zur Entwicklungszeit binden können, sind

ID Binding

dazu angehalten, einmal zur Laufzeit ermittelte DispIDs in einem Cache zu halten und somit eine zweite Anfrage an `GetIDsOfNames()` zu vermeiden. Kompilierte Sprachen können eine DispID auch schon zur Entwicklungszeit aus einer Typbibliothek ermitteln und in die Binärform eines Clients hineinkompilieren. Dieses Verfahren wird auch als *ID Binding* bezeichnet. Da ID Binding eine Form des frühen Bindens ist, wird das VTBL Binding zur Abgrenzung vom ID Binding auch als *sehr frühes Binden* bezeichnet.

Grenzen von IDispatch

Ein COM-Server kann natürlich neben `IDispatch` auch benutzerdefinierte Schnittstellen unterstützen. Über die COM-Automation sind jedoch in jedem Fall nur jene Methoden erreichbar, die in `IDispatch` unterstützt werden. Sofern die benutzerdefinierten Schnittstellen weitere Methoden unterstützen, sind diese nur via `IUnknown` und VTBL Binding erreichbar. `IDispatch` konterkariert also den Vorteil der Mehrfachschnittstellen – das ist der Tribut, den man beim Windows Scripting zahlen muss.

Automations-MINFU

Viele verwirrende Begriffe

Leider betreibt Microsoft auch an dieser Stelle wieder ein **MI**crosoft **N**omenclature **F**oul-Up (MINFU, vgl. Erläuterungen in Anhang B) und zeigt sich sehr einfallsreich und wenig konsistent darin, den Unterschied zwischen automationsfähigen und nichtautomationsfähigen Schnittstellen einer Komponente zu dokumentieren.

So findet man in der MSDN Library für automationsfähige Schnittstellen z.B. die Begriffe:

- ▶ *ActiveX Programming Objects* (XML Reference im Web Workshop)
- ▶ *Scripting API* (WMI Reference)
- ▶ *Visual Basic Object Model* (ASP Reference)

Dagegen heißen die nichtautomationsfähigen Schnittstellen oft:

- ▶ *COM Programming Interfaces* (XML Reference)
- ▶ *COM API* (WMI Reference)
- ▶ *Non-Automation Interfaces* (ADSI Reference)

Grundsätzlich können Sie als Anhaltspunkt nehmen: Die Verwendung des Begriffs *Interface* deutet eher auf nichtautomationsfähige Schnittstellen hin, während Objekte über `IDispatch` angesprochen werden können. Dies entspricht der eher auf Klassen bzw. Objekte gerichteten Sicht der Automation. Aber auch das ist nicht allgemein gültig, denn oft fehlt jeglicher Hinweis auf die `IDispatch`-Unterstützung. Als wäre es so schwer, jeder Komponentenreferenz `IDispatch: Ja | Nein` voranzustellen!

2.15.3 Duale Schnittstellen

Ein guter Ansatz ist die Implementierung aller Schnittstellen als sogenannte *duale Schnittstellen*. Eine duale Schnittstelle ist eine benutzerdefinierte Schnittstelle, die nicht direkt von `IUnknown`, sondern von `IDispatch` abgeleitet wird. Da `IDispatch` von `IUnknown` abgeleitet ist, verfügt eine duale Schnittstelle natürlich auch über die nötige `IUnknown`-Implementierung. Die `vTable` einer dualen Schnittstelle besteht also aus mindestens sieben Methoden (drei von `IUnknown`, vier von `IDispatch`). Die weiteren Positionen der `vTable` enthalten dann direkte Zeiger auf die Methoden der Schnittstelle. Eine Klasse kann über mehrere duale Schnittstellen verfügen. Die `IDispatch`-Implementierung sollte jedoch jeweils dieselbe sein, und natürlich sollten über jedes `IDispatch::Invoke()` alle benutzerdefinierten Methoden aller Schnittstellen zur Verfügung stehen.

Duale Schnittstellen

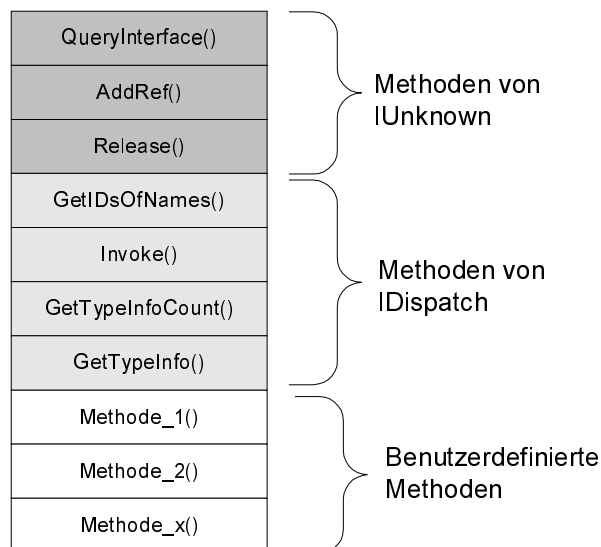


Abbildung 2.15:
Table einer dualen
Schnittstelle

COM-Clients haben dann die Auswahl, über die `vTable` direkt auf die benutzerdefinierten Methoden zuzugreifen oder aber über `IDispatch::Invoke()` den dynamischen Aufruf zu benutzen. Natürlich sollte ein entsprechend mächtiger Client immer den benutzerdefinierten Teil der Schnittstelle verwenden!

In der Regel werden heute COM-Objekte mit dualen Schnittstellen generiert. Beispielsweise erzeugt Visual Basic immer duale Schnittstellen für in VB erstellte COM-Klassen.



2.15.4 IDispatchEx

Erweiterte Version von IDispatch

IDispatchEx ist eine verbesserte Version von IDispatch, die zusätzlich folgende Features unterstützt:

- ▶ Ermittlung der Namen der von der Klasse bereitgestellten Methoden zur Laufzeit – auch ohne Typbibliothek
- ▶ Hinzufügen von Methoden zur Laufzeit
- ▶ Entfernen von Methoden zur Laufzeit

IDispatchEx wird bisher leider nur von wenigen COM-Klassen implementiert.

2.16 Distributed COM

DCOM Das *Distributed Component Object Model* (DCOM) ist ein Zusatzdienst zum Component Object Model (COM), der es ermöglicht, COM-Komponenten nicht nur auf dem lokalen Rechner, sondern auch auf entfernten Systemen zu benutzen. Ein früherer Name von DCOM ist *Network OLE*. DCOM wurde mit NT4 ausgeliefert, später aber auch als Add-on zu Windows 95 verfügbar gemacht. Seitdem gehört DCOM zum Standard jeder Betriebssystemversion bei Microsoft.

OSF DCE Dem Component Object Model und dem DCOM-Dienst liegt die Distributed Computing Environment (DCE) der Open Software Foundation (OSF) zu Grunde. DCE definiert eine Umgebung für verteilte Systeme mit Werkzeugen und Diensten für verteilte Anwendungen. Zentraler Dienst im DCE ist der Remote Procedure Call (RPC).

**Eigenschaften
Fernaktivierung** Jede COM-Komponente kann ohne Veränderungen sofort mit DCOM eingesetzt werden. Es müssen lediglich die Sicherheitseinstellungen entsprechend konfiguriert werden. DCOM unterscheidet:

- ▶ ausgehende Fernaktivierung – ein lokaler Client nutzt eine entfernte Komponente
- ▶ eingehende Fernaktivierung – ein entfernter Client benutzt eine lokale Komponente

2.16.1 DCOM-Protokoll

DCOM-Protokoll DCOM ist ein Dienst, der mit Hilfe des *DCOM-Protokolls* realisiert wird. Das DCOM-Protokoll basiert auf den im Distributed Computing Environment (DCE) spezifizierten Remote Procedure Call (RPC) und wird auch Object RPC (ORPC) genannt. Es ist im ISO/OSI-Referenzmodell auf der Schicht 7 (Anwendungsebene) angesiedelt und transportprotokollunabhängig; es kann auf verschiedene Transportprotokolle (z.B. TCP/IP, UDP/IP, IPX/SPX, AppleTalk), aber auch auf Anwendungsprotokollen wie HTTP aufsetzen.

Distributed COM

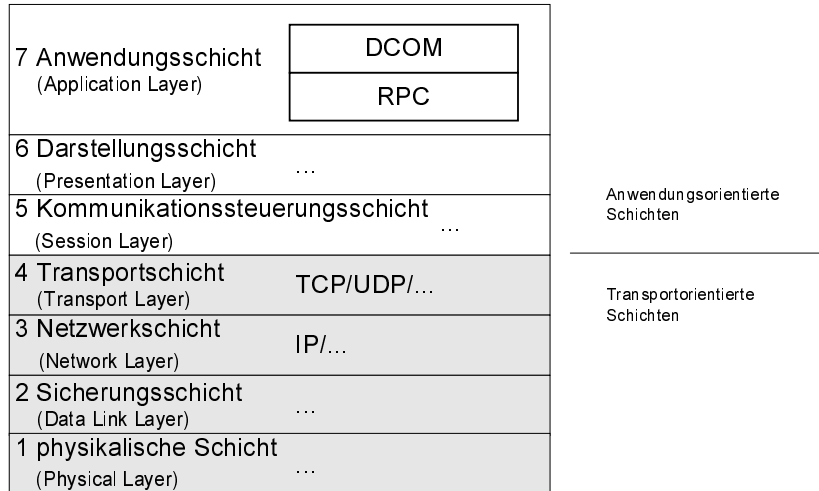


Abbildung 2.16:
Einordnung von
DCOM in das ISO/
OSI-Referenzmodell

Eigenschaften

Zu den Features von DCOM gehört eine Distributed Garbage Collection, die über einen *Ping*-Mechanismus realisiert wird. Ein COM-Client muss regelmäßig eine kurze Nachricht an den Computer senden, der den COM-Server hostet. Wenn dieser Ping mehrmals nicht eintrifft, dann reduziert COM den Referenzzähler der verwendeten Instanz des COM-Servers. Wenn der Referenzzähler den Wert 0 erreicht hat, verwirft COM die Instanz des COM-Servers. Der DCOM-Ping-Mechanismus ist nicht mit dem auf dem ICMP-Protokoll basierenden Netzwerk-Ping im TCP/IP-Protokollstack zu wechseln.

Ping

Methodenaufrufe sind in COM/DCOM immer synchron. Zwar gibt es auch eine asynchrone Form des RPC, diese wird von DCOM jedoch nicht verwendet.

Synchron

2.16.2 DCOM-Installation und -Konfiguration

DCOM gehört zum Basisumfang von NT4, Windows2000, Windows 98 und Windows ME. Auf Windows 95 kann DCOM durch Installation des Internet Explorers ab Version 4.0 oder durch ein spezielles DCOM-Add-on für Windows 95 ermöglicht werden. Für das in Windows 98 enthaltene DCOM gibt es ein Update.

**Installation und
Konfiguration**

Alte 16-Bit-Windows-Betriebssysteme können über die alte Remote Automation-Technik auf COM-Komponenten zugreifen.

DCOM-Konfiguration

DCOM zulassen Damit DCOM auf einem PC möglich ist, muss der Registry-Schlüssel `HKEY_LOCAL_MACHINE \ SOFTWARE \ MICROSOFT \ OLE \ ENABLED COM` auf den Wert »Y« (für »Yes«) gesetzt sein. Mit einem anderen Wert sind weder eingehende noch ausgehende Fernaktivierungen möglich. Lokale Aktivierungen sind dennoch erlaubt, sofern die der Klasse zugeordneten Startberechtigungen dies für den Benutzerkontext des Clients erlauben.

Auf einem Windows 95/98/ME-System muss zusätzlich der Schlüssel `HKEY_LOCAL_MACHINE \ SOFTWARE \ MICROSOFT \ OLE \ ENABLEREMOTE CONNECTIONS` auf »Y« gesetzt werden, damit COM-Objekte angesprochen werden können.

Die für die DCOM-Kommunikation zu verwendenden Protokolle befinden sich im Registry-Schlüssel `HKEY_LOCAL_MACHINE \ SOFTWARE \ MICROSOFT \ RPC`.

Ports Jeder Prozess, der ein COM-Objekt enthält, erhält dynamisch eine Portnummer zwischen 1024 und 65535. Das nachfolgende REGEDIT4-Listing zeigt die notwendigen Registry-Einträge, um die Ports einzuschränken. Im folgenden Beispiel werden nur die Ports 4000-6000 erlaubt.

```
REGEDIT4
[HKEY_LOCAL_MACHINE\Software\Microsoft\Rpc\Internet]
"Ports"="4000-6000"
"PortsInternetAvailable"="Y"
"UseInternetPorts"="Y"
```

Listing 2.5: Registry-Einstellungen zur Einschränkung der DCOM-Ports
[CD: /install/com/portbeschraenkung.reg]



Alle vorgenannten Einstellungen können auch über das DCOM Configuration Utility (DCOMCNFG) vorgenommen werden.

2.16.3 DCOM im Internet

DCOM kann nicht nur im LAN und WAN, sondern prinzipiell auch im Internet genutzt werden, da TCP/IP als Netzwerkprotokoll unterstützt wird. Dem stehen jedoch Firewalls entgegen, welche die für das DCOM-Protokoll notwendigen TCP-Ports sperren oder IP-Adressen aus Sicherheitsgründen umsetzen (DCOM-Stubs speichern die IP-Adressen). Oft kann oder will man aber die Konfiguration einer Firewall nicht ändern, um DCOM den Weg freizumachen.

COM Internet Service (CIS) Microsofts Lösung dafür heißt *COM Internet Service (CIS)* (in der deutschen Version: *COM-Internetdienste*). CIS meint das Tunneling des DCOM-Protokolls im Hypertext Transfer Protocol (HTTP). Dabei wird jedoch nur die erste DCOM-Nachricht in eine HTTP-Nachricht verpackt. Alle weiteren DCOM-Nachrichten werden direkt via Port 80 versendet. Auch das wird

von einigen Firewalls nicht erlaubt. Eine weitere Einschränkung ist, dass CIS nur in Verbindung mit einem Internet Information Server ab Version 4.0 auf dem CIS-Server funktioniert. Ein Windows 9x/ME-Rechner kann also kein CIS-Server sein.

Die COM-Komponente, die auf CIS aufsetzt, heißt *Remote Data Service* (RDS). RDS wird in diesem Buch nicht weiter behandelt. Nähere Informationen hierzu finden Sie in der MSDN Library [MSL00].

RDS

CIS-Installation

CIS ist in folgenden Systemen enthalten:

CIS-Installation

- ▶ NT4 durch Installation von Service Pack 4
- ▶ Windows 95 ab DCOM Version 1.2
- ▶ Windows 98 ab DCOM Version 1.3
- ▶ Windows ME
- ▶ Windows2000 (Installationsoption)
- ▶ Windows XP (Installationsoption)
- ▶ Windows.NET (Installationsoption)

CIS-Konfiguration unter Windows2000

Auf einem Windows2000-Server erfolgt durch Auswahl der Installationsoption NETZWERKDIENTE/COM INTERNETDIENSTEPROXY nicht nur die Installation von CIS, sondern auch die korrekte Konfiguration des IIS. Auf einem NT4-Server müssen Sie selbst sicherstellen, dass

CIS-Konfiguration

- ▶ der ISAPI-Filter RPCPROXY.DLL aktiviert ist;
- ▶ unterhalb der Standardwebsite ein virtueller Pfad mit Namen RPC existiert, der eine Kopie der RPCPROXY.DLL enthält und auf dem Ausführungsrechte aktiviert sind;
- ▶ CIS aktiviert ist (in DCOMCNFG müssen Sie die Option INTERNETDIENSTE AUF DIESEM RECHNER GRUNDSÄTZLICH AKTIVIEREN auf der Registerkarte STANDARDEIGENSCHAFTEN aktivieren);
- ▶ CIS den Standardprotokollen hinzugefügt ist (der Eintrag »TUNNELING TCP/IP« auf der Registerkarte STANDARDPROTOKOLLE muss hinzugefügt sein, siehe folgende Abbildung).

Auf dem CIS-Client müssen nur die letzten beiden Schritte durchgeführt werden. Unter Windows 95/98/ME gibt es dafür das Kommandozeilentool CISCNFG.EXE.

ciscnfg.exe

Abbildung 2.17:
Aktivierung von
CIS mit Hilfe von
DCOMCNFG auf
einem NT-Client



Simple Object Access Protocol (SOAP)

SOAP und XML RPC

Die Zukunft der Inter-process-Kommunikation im Internet wird aber nicht DCOM gehören, sondern wahrscheinlich einem Protokoll mit dem Namen *Simple Object Access Protocol* (SOAP). SOAP realisiert einen RPC auf Basis des Austausches von XML-Daten via HTTP. SOAP will unabhängig von Plattformen, Komponentenarchitekturen, Sprachen und Firewalls sein. SOAP wird im .NET-Framework verwendet. Weitere Informationen zu SOAP finden Sie bei [SOA01a], [SOA01b] und [DVM00]. XML RPC ist ein alternativer, wenn auch sehr ähnlicher Ansatz zu SOAP [XRP00], der sich nicht durchgesetzt hat.

2.17 Objektaktivierung

Aktivierung

Wesentliche Voraussetzung für die Ausführung eines Methodenaufrufs in einer Komponente ist die ordnungsgemäße Aktivierung des Objekts. Das ist notwendig, um einem COM-Client einen Schnittstellenzeiger auf einen bisher von ihm nicht benutzten COM-Server zu liefern. Diese Vorgänge sind sehr komplex und sollen hier nur in Grundzügen erläutert werden.

COM unterstützt drei Formen der Aktivierung:

- ▶ Erzeugung neuer Objekte durch Instanziierung von Klassen
- ▶ Zugriff auf bestehende Instanzen im Speicher
- ▶ Zugriff auf persistente Objektinstanzen

2.17.1 Service Control Manager (SCM)

Der *Service Control Manager* (SCM) ist derjenige Teil der COM-Implementierung, der dafür zuständig ist, Anfragen von COM-Clients nach Schnittstellen entgegenzunehmen und mit einem Zeiger auf die entsprechende Schnittstelle zu antworten. COM-Client und COM-Server kommunizieren aber nicht direkt mit dem SCM; sie rufen Methoden in der COM-Bibliothek auf, die ihrerseits den SCM ansprechen.

SCM

Der SCM ist kein *Object Trader*, der in der Lage wäre, eine Anforderung der Form »Liefere mir irgendeine Komponente, die eine Textdatei lesen und schreiben kann und Unicode unterstützt« zu bearbeiten. Die Auswahl der Komponente muss der Client treffen. Das Component Object Model stellt bislang neben der Kategorisierung von Klassen in Component Categories (siehe Kapitel 2.20) keinen Trading-Mechanismus bereit. Auch der Standort der Komponente muss dem Client bekannt sein. Oft werden daher selbst programmierte Trader-Objekte zwischen Client und Server geschaltet.



Der SCM ist in RPCSS.EXE implementiert und im Task-Manager sichtbar, da er automatisch gestartet wird. Der SCM verwendet als festen TCP-/UDP-Port den Port Nummer 135. Nur so weiß ein SCM immer, wie er den SCM-»Kollegen« auf einem entfernten Rechner erreichen kann. Sofern ein Client eine Remote-Instanziierung fordert, hat diese Anforderung für den SCM Vorrang vor den in der Registry konfigurierten lokalen Komponenten.

RPCSS.EXE

2.17.2 Erzeugung neuer Instanzen

Die Funktion der COM-Bibliothek, mit der eine neue Instanz angefordert werden kann, heißt `CoCreateInstance()`. Wichtigste Parameter von `CoCreateInstance()` sind die CLSID der gewünschten Klasse und die IID der gewünschten Schnittstelle; optional ist der Name des Computers, von dem das Objekt angefordert werden soll. Sofern ein Client eine ProgID statt einer CLSID übergibt, muss diese vorher umgewandelt werden. Skriptsprachen fordern natürlich immer `IDispatch` an und erledigen die Umwandlung von der ProgID in die CLSID automatisch. Der Computernamen kann sowohl der NetBIOS-Name des Rechners als auch der DNS-Name oder die IP-Adresse sein.

`CoCreateInstance()`

Eine wichtige Fragestellung im Rahmen von DCOM ist stets, auf welchem Computer eine COM-Klasse instanziiert werden soll. Es gibt zwei Möglichkeiten, den Aktivierungsort zu spezifizieren:

- **Über einen Eintrag in der Registry:** In der Registry kann in einem AppID-Schlüssel `REMOTESERVERNAME` der DNS-Name des Rechners eingetragen werden, auf dem die Instanziierung erfolgen soll.
- **Als expliziter Parameter beim Aufruf der Instanziierungsroutine innerhalb des COM-Clients:** Die Angabe in der Registry ist ein Instru-

`RemoteServerName`

Parameter bei der Instanziierung

ment, um Ortstransparenz herzustellen; ein COM-Client muss nicht wissen, wo sich das aufzurufende Objekt befindet. Dies ist in den Fällen von Nachteil, in denen der Client einen bestimmten Rechner ansprechen will. Der Client müsste dann vor der Instanziierung die Registry ändern. Dies birgt die Gefahr von Wechselwirkungen mit anderen Programmen, die ebenfalls dieses Objekt benutzen.

Viele Hochsprachen kapseln `CoCreateInstance()` hinter eigenen Funktionen:

- ▶ VBScript kapselt den Aufruf in `CreateObject()`.
- ▶ Visual Basic kapselt den Aufruf in `new` oder `CreateObject()`.
- ▶ JScript verwendet `new ActiveXObject()`.

```
Dim Objektvariable
set Objektvariable = CreateObject("Komponente.Klasse")
```

Listing 2.6: Instanziierung einer COM-Klasse in allen Visual Basic-Dialekten

```
Dim Objektvariable As Komponente.Klasse
set Objektvariable = new Komponente.Klasse
```

Listing 2.7: alternative Möglichkeit zur Instanziierung einer COM-Klasse in Visual Basic 6.0/VBA

```
var Objektvariable;
Objektvariable = new ActiveXObject("Komponente.Klasse");
```

Listing 2.8: Instanziierung einer COM-Klasse in JScript

Dabei stehen jeweils weniger Optionen zur Verfügung als bei dem ursprünglichen `CoCreateInstance()`. Die verfügbaren Optionen reichen aber für die Skriptprogrammierung aus und verbergen viel von der Komplexität einer Instanziierung in COM.

Aktivierungsvorgang

Ablauf Durch einen Aufruf von `CoCreateInstance()` führt der SCM folgende Schritte aus:

1. Der SCM sucht in der Registry nach einem entsprechenden Eintrag für die gewünschte CLSID.
2. Der SCM stellt fest, ob die Komponente auf dem lokalen System oder auf einem entfernten System instanziiert werden soll. Dabei hat ein bei `CoCreateInstance()` übergebener Rechnername das größte Gewicht. Fehlt dieser, prüft der SCM in der Registry, ob dort für die AppID, zu der die Klasse gehört, ein `RemoteServer`-Eintrag besteht. Fällt auch diese Überprüfung negativ aus, erfolgt die lokale Instanziierung. Sofern eine Fernaktivierung gewünscht ist, leitet der SCM die Anfrage über DCOM an den SCM des entfernten Systems weiter.

Objektaktivierung

3. Der betroffene (lokale oder entfernte) SCM stellt fest, welche Datei die Komponente implementiert. Er lädt diese Datei und startet – wenn nötig – einen Prozess für die Komponente.
4. Jede COM-Komponente besitzt für jede Klasse eine sogenannte *Class Factory* in Form eines *ClassFactory*-Objekts (auch *ClassObject* genannt) mit einer Standardschnittstelle *IClassFactory*. Einen Zeiger auf die Class Factory erhält der SCM über die DLL-Funktion *DllGetClassObject()*, die von jeder COM-DLL bereitgestellt wird. Der SCM ruft die Methode *IClassFactory::CreateInstance()* auf. Die Class Factory erzeugt daraufhin eine neue Instanz der gewünschten COM-Klasse.
5. Das *ClassFactory*-Objekt übergibt dem SCM nach erfolgreicher Instanziierung einen Zeiger auf die gewünschte Schnittstelle des neuen Objekts.
6. Der SCM leitet diesen Schnittstellenzeiger an den Client weiter.

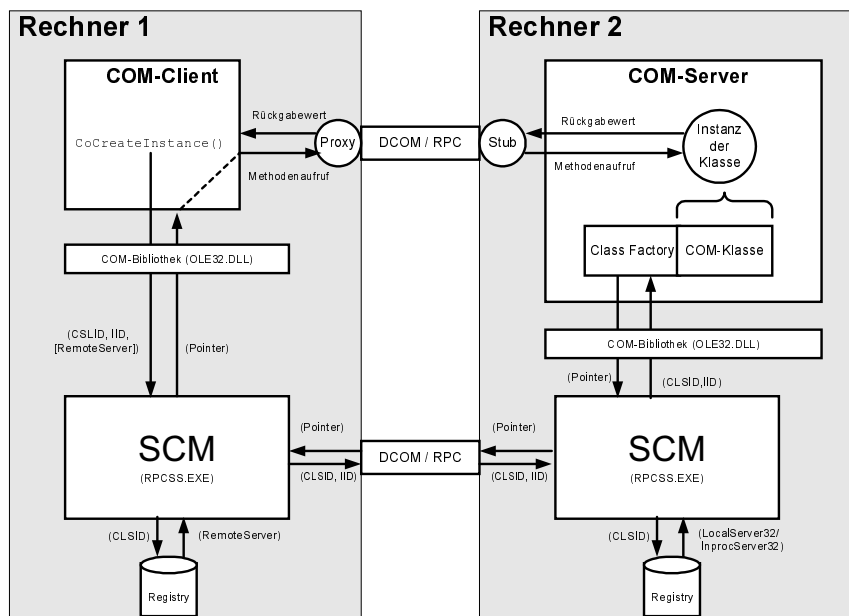


Abbildung 2.18:
Ablauf der Aktivierung
auf einem entfernten Rechner

Danach kann der Client ohne weitere Einschaltung des SCMs direkt – über Proxy- und Stub-Objekte – mit der Komponente kommunizieren, da jede Schnittstelle `IUnknown::QueryInterface()` unterstützt und so zur weiteren Anfrage nach Schnittstellen verwendet werden kann.

Surrogat-Prozesse

Eine EXE-Datei wird entfernt aufgerufen, indem sie auf dem entfernten Rechner gestartet wird. Dies ist mit DLLs nicht möglich.

Damit eine prozessinterne Komponente von einem anderen Rechner aus aufgerufen werden kann, ist ein sogenannter *Surrogat-Prozess* (engl. *Surro-*

**Ausführung von
DLLs in einem
eigenen Prozess**

gate Process) notwendig, der die prozessinterne Komponente aufnimmt und ausführt.

Microsoft liefert zwei Standard-Surrogat-Prozesse: DLLHOST.EXE und MTS.EXE. Dabei wird MTS.EXE nur innerhalb des Microsoft Transaction Servers unter NT4 verwendet. Unter COM+ in Windows2000/XP/.NET ist DLLHOST.EXE der Standard-Surrogat-Prozess. DLLHOST.EXE stand auch schon in NT4 zur Verfügung, wenn eine Aufnahme in den MTS nicht gewünscht war. Sie konfigurieren eine Klasse einer In-process-Komponente für die Out-process-Verwendung, indem Sie die Klasse in ein MTS-Package (vgl. Kapitel 2.25) oder in eine COM+-Anwendung (vgl. Kapitel 2.26) integrieren. C++-Programmierer können einen solchen Surrogat-Prozess auch selbst schreiben.

Windows 95/98

Komponenten, die auf Windows 9x/ME-Plattformen laufen, können nicht fernaktiviert werden. Diese Plattformen machen es lediglich möglich, eine Fernverbindung zu bereits laufenden Objekten herzustellen. COM-Server müssen also auf diesen Systemen manuell gestartet werden.

2.17.3 Zugriff auf bestehende Instanzen

COM-Namensdienst

Die Basis für den Zugriff auf bestehende Instanzen ist der COM-Namensdienst. Mit diesem Namensdienst können einzelne Instanzen einer Klasse von anderen Instanzen unterschieden werden.

Moniker

Instanzen haben in COM keine GUIDs, sondern Moniker (siehe Kapitel 2.4). Moniker werden intern selbst in Objekten (Moniker-Objekten mit der Schnittstelle `IMoniker`) gespeichert. Ein COM-Server kann einem COM-Client ein Moniker-Objekt übergeben, um dem Client zu ermöglichen, zu einem späteren Zeitpunkt wieder eine Verbindung zu dieser Instanz aufzunehmen, nachdem die Objektreferenz zwischenzeitlich gelöscht wurde.

Running Objects Table (ROT)

ROT Die Running Objects Table ist eine globale Tabelle auf einem Rechnersystem, in der instanziierte Objekte verzeichnet werden können. Es gibt nur genau eine ROT pro Rechner. COM-Clients können Instanzen aus der ROT mitverwenden. So kann beispielsweise erreicht werden, dass neu zu öffnende Dateien einer Anwendung mit Multi-Document-Interface (MDI; z.B. Word 97) in einem bestehenden Fenster geöffnet werden können. Die Aufnahme einer Instanz in die ROT erfolgt keineswegs automatisch, sondern ist eine Option, die das Objekt selbst wahrnehmen oder eben nicht wahrnehmen kann. Ein Objekt meldet sich beim SCM über ein Moniker-Objekt an oder ab. Natürlich kann ein COM-Server die Entscheidung für oder gegen eine Anmeldung dem COM-Client überlassen. Die ROT kann mit Hilfe des in Visual Studio enthaltenen ROT-Viewers (siehe Kapitel 2.28.12) eingesehen werden.

Als Skriptprogrammierer haben Sie ebenso wenig wie ein Visual Basic-Programmierer die Möglichkeit, Instanzen an die ROT zu melden. Zumindest scheint sich noch niemand die Arbeit gemacht zu haben, eine COM-Komponente zu schreiben, die die entsprechenden API-Aufrufe kapselt.



Aktivierung persistenter Instanzen aus Dateien

Mit Hilfe von File Monikern der Form

```
file://pfad/dateiname.extension
```

File Moniker

besteht eine Zugriffsmöglichkeit auf im Dateisystem persistent gemachte Objektinstanzen. Dazu dient die COM-Bibliotheksfunktion `CoGetInstanceFromFile()`.

Die Dokumente der Microsoft Office-Produktfamilie sind beispielsweise persistente Instanzen von COM-Klassen in Form von Compound Documents.



Wenn ein Client mit Hilfe der COM-Bibliotheksfunktion `CoGetInstanceFromFile()` (in Visual Basic und JScript: `GetObject()`) auf eine Datei zugreift, werden folgende Schritte vom SCM ausgeführt:

CoGetInstanceFromFile()

1. Anhand des Headers der Datei bzw. der Dateierweiterung bestimmt der SCM die CLSID der COM-Klasse, die zur Verwaltung der angeforderten Datei fähig ist. Es ist auch möglich, bei `CoGetInstanceFromFile()` bzw. `GetObject()` eine CLSID (oder ProgID) für eine Klasse anzugeben, die zur Aktivierung verwendet werden soll.
2. Anschließend überprüft der SCM zunächst in der ROT, ob diese Objektinstanz bereits aktiviert wurde. Ist der zugehörige Moniker in der ROT eingetragen, so liefert der SCM den Schnittstellenzeiger an den Client.
3. War die Instanz nicht schon in Benutzung, so aktiviert der SCM eine neue Instanz der Klasse und fordert die Schnittstelle `IPersistFile` an.
4. Der SCM übergibt `IPersistFile::Load()` die Anforderung zum Laden der Datei und überlässt das weitere Vorgehen (also die Übernahme der Daten aus den Streams des Compound Documents in die internen Speicherstrukturen) der Objektinstanz der Klasse selbst. Dazu gehört auch der Eintrag in die ROT nach erfolgreichem Laden der Datei.

Moniker oder nicht Moniker

Die Frage, ob irgendetwas als eine bestehende Instanz angesehen wird oder erst eine Instanz erzeugt werden muss, ist eine philosophische Frage, die jeder Komponentenprogrammierer für sich selbst beantworten muss. Während viele Komponenten den Zugriff via Moniker ermöglichen, wird in

Auslegungssache

anderen Fällen zunächst eine »leere« Instanz einer Klasse erzeugt und diese dann per Methodenaufruf an eine Entität gebunden. Wie die folgende Tabelle zeigt, gibt es keine konsistente Sichtweise darauf, ob etwas direkt per Moniker angesprochen werden kann oder instanziiert werden muss.

Tabelle 2.6:
Unterschiedliche
Semantik hinsicht-
lich der Persistenz
von Objekten in ver-
schiedenen Kompo-
nenten

Instanz wird als persistent angesehen (Moniker-Zugriff via GetObject())	Instanz muss erzeugt werden (Indirekter Zugriff via CreateObject())
▶ Word-Datei ▶ Verzeichnisobjekt ▶ WMI-Objekt ▶ Registry-Schlüssel in der Kompo- nente »ADsSecurity«	▶ Access-Tabelle ▶ SQL-Server-Tabelle ▶ Ereignisprotokoll ▶ Klasse im COM+-Katalog ▶ Registry-Schlüssel in der Kompo- nente »REGCOL«

Teilweise gibt es auch für gleiche Entitäten der Realwelt zwischen verschiedenen Komponenten unterschiedliche Vorgehensweisen: So wird in der *ADsSecurity-Komponente* ein Registry-Schlüssel via Moniker angesprochen, in der Komponente *REGCOL* jedoch indirekt per Methodenaufruf.

2.17.4 Aktivierungsfehler

Mögliche Fehlerquellen

Es gibt eine Reihe möglicher Ursachen, warum eine Aktivierung fehlschlägt:

- ▶ Die Klasse ist gar nicht installiert.
- ▶ Die Klasse ist nicht richtig registriert.
- ▶ Die angegebene ProgID ist nicht mit der entsprechenden CLSID verbunden.
- ▶ Der Benutzer besitzt keine Rechte für die Nutzung der Klasse.

Bei einer Fernaktivierung kommen weitere Gründe hinzu:

- ▶ Der angesprochene Rechner existiert nicht oder ist nicht erreichbar.
- ▶ Auf dem angesprochenen Rechner ist DCOM deaktiviert.
- ▶ Für eine COM-DLL gibt es keinen Surrogat-Prozess.
- ▶ Die benötigten lokalen Registry-Einträge sind nicht vorhanden.

Leider differenzieren viele Sprachen bzw. Hosts nicht genau zwischen all diesen Fehlerursachen. Mit Vorliebe melden sie einfach »Objekterstellung durch ActiveX-Komponente nicht möglich« (VB6) oder »ActiveX-Komponenten kann kein Objekt erstellen« (WSH). Ganz abgesehen davon, dass die letzte Meldung grammatikalisch falsch ist, sind beides unglückliche Formulierungen, die einen Anfänger verwirren. Wieso will eine ActiveX-Komponente ein Objekt erstellen? Gemeint ist wohl, dass innerhalb einer Komponente eine bestimmte Klasse nicht instanziiert werden kann.

COM-Anwendungen

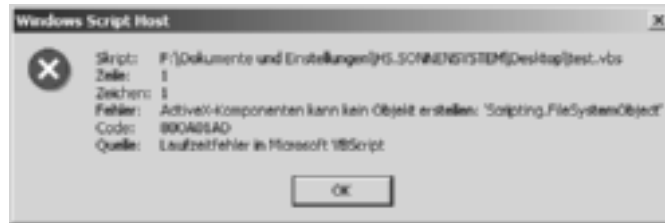


Abbildung 2.19:
Wenig aussagekräftige Fehlermeldung
beim Windows
Scripting Host

2.18 COM-Anwendungen

Eine COM-Anwendung fasst eine oder mehrere COM-Klassen zusammen und ermöglicht die gemeinsame Konfiguration dieser Klassen. Zahlreiche Einstellungen (z.B. Sicherheitseinstellungen) sind nur auf Anwendungsebene, nicht jedoch für eine einzelne Klasse konfigurierbar. Jede COM-Klasse kann nur einer COM-Anwendung angehören. Alle Einstellungen einer COM-Anwendung gelten für alle zugehörigen Klassen.

COM-Anwendungen

Jede COM-Anwendung verfügt über eine sogenannte Anwendungsken-
nung, kurz AppID. AppIDs können mit einer CLSID identisch sein. Oft wird
die erste CLSID einer Komponente auch als AppID benutzt.

AppIDs

Eine COM-Anwendung ist nicht mit einer COM-Komponente gleichzu-
setzen. Eine COM-Anwendung kann Klassen aus mehreren COM-Kom-
ponenten umfassen. Allerdings müssen alle COM-Klassen, die in einer
COM-EXE enthalten sind, auch zur gleichen COM-Anwendung gehören.
Diese Beschränkung gilt nicht für COM-DLLs.



COM-Objekte, die Instanzen von Klassen mit derselben AppID sind, werden
im gleichen Prozess ausgeführt. Die AppIDs befinden sich in der Registry
unter HKEY_LOCAL_MACHINE\SOFTWARE\CLASSES\APPID\. Unter-
halb einer AppID können die in der folgenden Tabelle genannten Einstellun-
gen festgelegt werden.

Schlüssel	Erläuterung
AuthenticationLevel	Individuelle Authentifizierungsstufe für diese COM-Anwendung
RemoteServerName	Hier kann ein Rechner oder eine IP-Adresse des Computers eingegeben werden, auf die die COM-Klassen instanziiert werden sollen (DCOM). Dieser Wert wird vom Service Control Manager (SCM) ausgelesen, wenn der Client bei den Instanzierungsaufforderungen keinen expliziten RemoteServer angegeben hat. Ist auch RemoteServerName nicht gesetzt, wird die Klasse vom SCM lokal installiert.

Tabelle 2.7:
Einstellungen für
COM-Anwen-
dungen

Tabelle 2.7:
Einstellungen für
COM-Anwen-
dungen
(Forts.)

Schlüssel	Erläuterung
AccessPermission	Zugriffsrechte in Form einer ACL
RunAs	Identitätseinstellung
LaunchPermission	Startberechtigungen in Form einer ACL
Endpoints	Zu verwendende Protokolle und Portnummern bzw. Pfade

Globale Einstellungen

Ein Wertname ist in der Registry nur vorhanden, wenn ein Wert existiert. Ist ein Wert nicht belegt, gelten die unter HKEY_LOCAL_MACHINE\SOFTWARE\MICROSOFT\OLE festgelegten Standardeinstellungen.

AppID-Konfiguration

Tools

Kompliziert aufgebaute ACLs direkt in der Registry zu manipulieren ist kaum möglich. Die einfachste Möglichkeit der Konfiguration von bestehenden COM-Anwendungen ist das DCOM Configuration Utility (DCOMCNFG, vgl. Kapitel 2.28.9). Auch der Microsoft COM-Viewer (Kapitel 2.28.4) bietet viele Einstellungsmöglichkeiten für COM-Anwendungen. Beim Anlegen neuer COM-Anwendungen bzw. der Änderung der zu einer Anwendung gehörenden Klassen lassen Sie obige Tools weitgehend im Stich: Sie können mit dem DCOMCNFG keine COM-Anwendungen erzeugen oder Klassenzuordnungen vornehmen.



Mit dem COM-Viewer können Sie nur COM-Anwendungen erzeugen, die aus genau einer Klasse bestehen. Es bleibt also nur der manuelle Eingriff in die Registry: Dazu nehmen Sie die CLSID einer der Klassen, die zur Anwendung gehören soll, und erzeugen einen neuen AppID-Schlüssel mit der gleichen GUID. Danach tragen Sie diese AppID bei allen Klassen, die zur Anwendung gehören sollen, als AppID-Wert in deren CLSID-Schlüssel ein.

Konfigurationsberechtigungen

DCOMCNFG ermöglicht auch eine Einstellung KONFIGURATIONSBERECHTIGUNGEN, die Sie nicht als Unterschlüssel einer AppID finden werden. Diese Berechtigungen sind die Rechte auf den AppID-Schlüssel selbst, die sonst nur mit dem älteren RegEdt32-Registry-Editor eingesehen werden können.

2.19 COM-Assemblies

COM hatte lange mit der DLL-Hölle zu kämpfen. Die DLL-Hölle entsteht im klassischen COM durch zwei Faktoren:

- Es kann nur eine Version einer bestimmten Komponente in der Registry registriert sein.

Dies bezieht sich insbesondere auf Unter-Versionen (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.

- 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 Nutzungsende vom System gesperrt.

Side-by-Side-Execution

Grundsätzlich wurde in Windows2000 die DLL-Hölle dadurch entschärft, dass nach DLLs zunächst im gleichen Verzeichnis gesucht wird, aus dem auch die EXE gestartet wurde. Allerdings funktionierte das Verfahren nicht mit COM-Komponenten.

Windows2000

Assemblies

In Windows XP und Windows .NET gibt es das neue Konzept der *Assemblies*, das es erlaubt, mehrere verschiedene Versionen einer Komponente parallel auf einem System zu verwenden, auch wenn die Komponenten den gleichen Dateinamen und die gleichen GUIDs verwenden. Weil Assemblies diese ermöglichen, spricht Microsoft auch von *Side by Side Assemblies*. Die Installation einer neuen Version einer Assembly beeinflusst die vorhandenen Anwendungen nicht mehr, so dass Anwendungen besser als bisher voneinander isoliert sind.

Windows XP und
Windows .NET

Manifest

Eine Assembly ist eine feste Ansammlung von einer oder mehreren Komponentendateien. Eine *Assembly* hat eine Konfigurationsdatei, die *Assembly Manifest* heißt. Ein *Assembly Manifest* umfasst folgende Daten:

Assembly
Manifest

- Ein Name der Assembly.
- Jede Assembly hat eine viergliedrige Versionsnummer (a.b.c.d), z.B. 1.8.19.72.
- Die Typangabe »win32«
- Liste der Komponentendateien, die zu der Assembly gehören
- COM-Konfigurationsdaten, die COM für die Lokalisierung und Aktivierung dieser Komponenten benötigt

► Liste der abhängigen Assemblies – jeweils mit Versionsnummer

Das folgende Listing zeigt ein Beispiel für ein Windows XP mitgeliefertes Manifest für die MFC-Bibliothek (Microsoft Foundation Classes – die Laufzeitumgebung für Visual C++) in der Version 6.0. Die Assembly umfasst die Dateien MFC42.DLL, ATL.DLL und MFC42U.DLL.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<assembly xmlns="urn:schemas-microsoft-com:asm.v1"
manifestVersion="1.0">
  <assemblyIdentity type="win32"
name="Microsoft.Tools.VisualStudio.Runtime-Libraries"
version="6.0.0.0" processorArchitecture="x86"
publicKeyToken="6595b64144ccf1df"/>

  <dependency optional="yes">
    <dependentAssembly>
      <assemblyIdentity type="win32"
name="Microsoft.Tools.VisualStudio.Runtime-Libraries.Resources"
version="6.0.0.0" processorArchitecture="x86"
publicKeyToken="6595b64144ccf1df" language="*/>
    </dependentAssembly>
  </dependency>

  <file name="mfc42u.dll"
hash="d9357dbe54a1f754cd8d662323acdeb100a0b0d1" hashalg="SHA1">
    <comClass description="Font Property Page" clsid="{0BE35200-
8F91-11CE-9DE3-00AA004BB851}"/>
    <comClass description="Color Property Page" clsid="{0BE35201-
8F91-11CE-9DE3-00AA004BB851}"/>
    <comClass description="Picture Property Page"
clsid="{0BE35202-8F91-11CE-9DE3-00AA004BB851}"/>
  </file>
  <file name="mfc42.dll"
hash="138a2057b090678d865720ed22276b00ede39168" hashalg="SHA1"/>
  <file name="atl.dll"
hash="60f116cba40bf191e78dd71177de8de79d79c50b" hashalg="SHA1">
    <comClass description="Registrar Class" clsid="{44EC053A-400F-
11D0-9DCD-00A0C90391D3}" progid="ATL.Registrar"/>
    <typelib tlbid="{44EC0535-400F-11D0-9DCD-00A0C90391D3}"
version="1.0" helpdir=""/>
  </file>
  <file name="msvcp60.dll"
hash="0d48860c3fdc649067ae29ef95635443d9d7064d" hashalg="SHA1"/>
    <comInterfaceExternalProxyStub name="IAxWinAmbientDispatch"
iid="{B6EA2051-048A-11D1-82B9-00C04FB9942E}"
```

COM-Assemblies

```
proxyStubClsid32="{00020424-0000-0000-C000-000000000046}"
numMethods="35" baseInterface="{00000000-0000-0000-C000-000000000046}" />

</assembly>
```

Listing 2.9: Beispiel für ein Manifest

Anwendungen

Eine Anwendung ist ein COM-Client, der eine oder mehrere Assemblies verwendet. Eine Anwendung besitzt im neuen Konzept ebenfalls eine XML-Datei, die *Application Manifest* heißt. Ein *Application Manifest* umfasst folgende Daten:

- ▶ Ein Name der Anwendung
- ▶ Eine viergliedrige Versionsnummer
- ▶ Liste der abhängigen Assemblies – jeweils mit Versionsnummer

Finden einer Komponente

Die Versionsnummer und Abhängigkeitsdaten sind die zentrale Neuerung. COM sucht für den Client eine Komponente, die die angegebenen Bedingungen erfüllt. Gesucht wird im Pfad der Anwendung und in einem globalen Komponentenverzeichnis mit dem Namen *Global Side-By-Side-Store* (WinSXS). Der Global Side-By-Side-Store liegt unter \WINDOWS\WINSXS. Dieses Verzeichnis gab es vor Windows XP nicht.

Application Manifest

Global Side-By-Side-Store

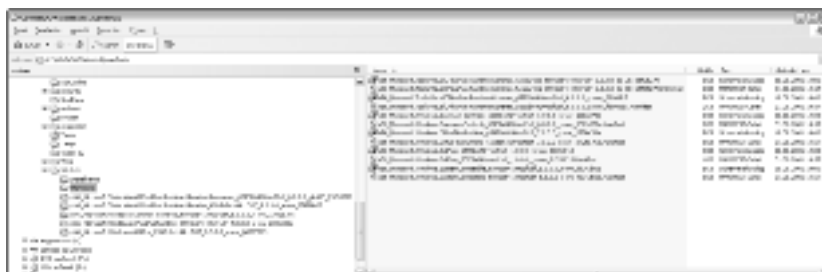


Abbildung 2.20: Verzeichnis WinSXS unter Windows XP

Das Assembly-Konzept erfordert keine Änderung an der Implementierung der Komponente selbst. Es ist lediglich das Erstellen von Manifesten erforderlich. So kann eine Anwendung auch zu einem späteren Zeitpunkt durch einen Administrator umkonfiguriert werden, so dass die Anwendung andere Versionen bestimmter Komponenten verwendet.



Name

Eine Assembly soll einen hierarchischen Namen besitzen in der Form

Firmenname.Anwendungsname.Anwendungsteil.Komponentenname

Ein Beispiel dafür ist

PearsonEducation.AddisonWesley.BuecherVerwaltung.Buch

Assembly-Typen

Windows XP/.NET unterscheiden zwei Typen von Assemblies:

Private Assemblies	▶ Private 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 installiert werden. Microsoft spricht in diesem Zusammenhang von XCOPY-Deployment.
Shared Assemblies	▶ Eine Shared Assembly kann – wie jede COM-Komponente bisher – von mehreren Anwendungen genutzt werden. Sie müssen digital signiert werden und müssen im Global Side-By-Side-Store (WinSXS) liegen. Shared Assemblies müssen mit Windows Installer 2.0 oder höher installiert werden. Im WinSXS gibt es für jede Assembly ein Manifest im Unterverzeichnis /MANIFESTS und ein Unterverzeichnis, das den Namen der Assembly zusammen mit dem öffentlichen Schlüssel trägt (z.B. x86_MICROSOFT.TOOLS.VISUALCPLUSPLUS.RUNTIME-LIBRARIES_6595B64144 CCF1DF_6.0.0.0_X-WW_FF9986D7). In diesem Verzeichnis werden die Komponentendateien dieser Assembly gespeichert. So sind die einzelnen Assemblies voneinander isoliert und gleichnamige Dateien sind kein Problem mehr.

COM und .NET



Assemblies und Manifeste in COM sind ähnlich, aber nicht gleich den Assemblies und Manifesten im .NET-Framework. So heißt zum Beispiel der Global Side-By-Side-Store dort Global Assembly Cache (GAC). Der Aufbau der XML-Dateien ist anders.

2.20 COM-Kategorien

Komponenten-kategorien

COM-Kategorien stellen eine Möglichkeit dar, COM-Klassen zu kategorisieren. Die Kategorisierung dient dazu, leichter feststellen zu können, ob ein COM-Client und ein COM-Server zusammenpassen. Obwohl es Klassen betrifft, die kategorisiert werden, heißt dieses Feature *Komponentenkategorien* (*Component Categories*) – ein typischer Fall von großem MINFU (vgl. Vorwort).

COM-Kategorien

Ein Client, der eine bestimmte Schnittstelle benötigt, aber keine COM-Klasse kennt, die diese implementiert, müsste normalerweise jede Klasse instanzieren und mit Hilfe von `QueryInterface()` anfragen, ob die betreffende Schnittstelle implementiert wird. Sofern der Client nicht nur eine einzelne Schnittstelle, sondern einen Satz von Schnittstellen benötigt, müsste er `QueryInterface()` bei jeder Klasse entsprechend oft aufrufen.

Durch die Zuordnung einer COM-Klasse zu einer Kategorie kann in der Registry hinterlegt werden, dass eine COM-Klasse einen bestimmten Satz von Schnittstellen implementiert. Ein Client muss dann lediglich in der Registry nach der entsprechenden Kategorie suchen.

Weitere Informationen finden Sie in der MSDN Library [MSD01c].

Benötigte und implementierte Kategorien

Eine COM-Klasse kann zu beliebig vielen Kategorien gehören; diese Zuordnung wird *Implemented Categories* genannt. Eine Komponente kann aber auch selbst mit Hilfe von *Required Categories* fordern, dass ein COM-Client zu einer bestimmten Komponentenkategorie gehört. Dies ist eine wichtige Funktion, wenn COM-Server Funktionen innerhalb eines COM-Clients aufrufen sollen.

**Required
Categories and
Implemented
Categories**

Jede Komponentenkategorie besitzt eine GUID, *CategoryID* (CATID) genannt. Eine Liste der definierten COM-Kategorien befindet sich in der Registry unter dem Schlüssel `HKEY_CLASSES_ROOT\COMPONENT CATEGORIES`.

Einsatzgebiete

COM-Kategorien können eingesetzt werden, wenn Clients nicht auf eine bestimmte COM-Klasse festgelegt sind, sondern jede COM-Klasse akzeptieren würden, die eine bestimmte Funktionalität erbringt. Damit ließe sich ein einfacher Trader für COM-Komponenten realisieren (vgl. [NNI00]). Leider werden COM-Kategorien so heute noch nicht benutzt. Sie werden bislang nur verwendet, um eine grobe Einteilung in verschiedene Komponententypen herzustellen. Entwicklungsumgebungen, die eine Liste von verfügbaren Komponenten anbieten, haben auf diese Weise einen schnellen Zugriff auf die Komponenten eines Typs. Denn es sollten in einer Dialogbox, die die verfügbaren ActiveX-Steuerelemente zeigt, keine nichtvisuellen COM-Komponenten angezeigt werden.

Einsatzgebiete

Im Standard installierte Komponentenkategorien auf einem Windows-System sind beispielsweise:

**Standard-
kategorien**

- ▶ Controls
- ▶ Controls that are safely scriptable
- ▶ Document Objects
- ▶ Embeddable Objects

- Java Classes
- Active Scripting Engines
- Automation Objects.

Persistenz Durch Komponentenkategorien kann auch beschrieben werden, welchen Mechanismus eine Klasse verwendet, um Objekte persistent zu machen (siehe Tabelle). Ein Objektcontainer (z.B. ein Web-Browser oder ein Windows-Fenster) sollte eine Klasse, die einen Persistenzmechanismus verlangt, der von dem Container und seiner Umgebung nicht unterstützt wird, nicht erzeugen.

Tabelle 2.8:
Komponentenkategorien für Objektpersistenz in COM

Name der Kategorie	CATID
RequiresDataPathHost	0de86a50-2baa-11cf-a229-00aa003d7352
PersistsToMoniker	0de86a51-2baa-11cf-a229-00aa003d7352
PersistsToStorage	0de86a52-2baa-11cf-a229-00aa003d7352
PersistsToStreamInit	0de86a53-2baa-11cf-a229-00aa003d7352
PersistsToStream	0de86a54-2baa-11cf-a229-00aa003d7352
PersistsToMemory	0de86a55-2baa-11cf-a229-00aa003d7352
PersistsToFile	0de86a56-2baa-11cf-a229-00aa003d7352
PersistsToPropertyBag	0de86a57-2baa-11cf-a229-00aa003d7352

Als Skriptentwickler sollten Sie sich besonders die verschiedenen, mit dem Begriff Active Scripting beginnenden Komponentenkategorien anschauen. Dort können Sie die installierten Scripting Engines entnehmen! Weiterhin ist natürlich die Liste der Automation Objects interessant. Benutzen Sie zur Betrachtung den Microsoft COM-Viewer.

2.21 Persistenz und Structured Storage

Compound Files Unter dem Begriff *Structured Storage* bietet COM ein Verfahren an, um Objekte und Objektmengen in strukturierter Form in Dateien zu speichern (Objektpersistenz). Die Persistenz erfolgt jedoch nicht automatisch oder durch eine einfache Deklaration. Sowohl der Programmierer des COM-Servers als auch der Programmierer des COM-Clients müssen die Persistenz explizit programmieren. *Structured Storage* definiert eine Reihe von Standardschnittstellen und damit Regeln, wie die Speicherung erfolgen kann.

Ein sogenanntes *Compound File* ist eine Datei, die eine Sammlung von *Storages* und *Streams* enthält. Ein Compound File besteht aus einem Storage, der beliebig viele Sub-Storages enthalten kann. Jedes Storage kann aus Streams und weiteren Sub-Storages bestehen. Ein Compound File ist daher vergleichbar mit einem Dateisystem: Storages sind Verzeichnisse, Streams sind Dateien.

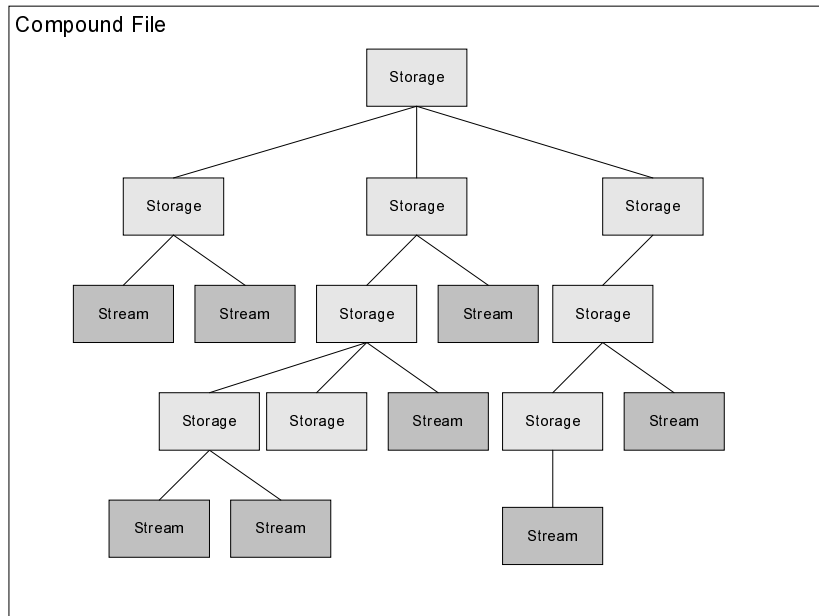


Abbildung 2.21:
Eine durch
Structured Storage
gegliederte Datei

Der große Vorteil von Structured Storage besteht in der Möglichkeit, unterschiedliche Arten von Daten in einer Datei zu speichern und einzelne Teile zu ändern, ohne das gesamte Compound File ändern zu müssen. COM definiert für Structured Storage die Standardschnittstellen `IPersist`, `IStorage`, `IStream` und `IRootStorage` und liefert Standardimplementierungen im Rahmen der COM-Bibliothek.

**Persistenz-
Schnittstellen**

Viele Windows-Anwendungen (z.B. Word-Dateien (.DOC) ab Version 6.0, Data Transformation Service-Dateien (.DTS)) speichern ihre Daten in Compound Files. Die Structured Storage-Technologie ist auch die Basis für die Speicherung von Zusatzattributen zu Dateien (z.B. Autorennamen, Firmenname, Kategorien, Versionsnummer).

**Anwendungs-
gebiete**

Structured Storage ist nicht zu verwechseln mit NTFS-Streams. Compound Files können auch im FAT-Dateisystem gespeichert werden.



Dateitypen

Ein Stream kann auch die CLSID der COM-Klasse enthalten, mit der die Datei geöffnet werden soll. In diesem Fall öffnet Windows eine Word-Datei auch dann mit Microsoft Word, wenn diese keine oder eine andere Dateierweiterung als .DOC hat. Windows schaut bei einer Datei, die einen Stream mit einer CLSID besitzt, gar nicht in der Registry nach der Dateierweiterung nach.

2.22 COM-Sicherheit

Sicherheitsfunktionen

COM benutzt für die Realisierung von Sicherheitsfunktionen die RPC-Sicherheitsinfrastruktur. Diese basiert auf den sogenannten Security Support Providern (SSP). Ein SSP ist in Form einer DLL installiert.

NT4 unterstützt standardmäßig nur den NT LAN-Manager Security Support Provider (NTLMSSP), implementiert in SECUR32.DLL. Windows2000 unterstützt zusätzlich Kerberos. Kerberos ist allerdings nur für entfernte, nicht für lokale Aufrufe verfügbar. Die Installation zusätzlicher SSPs ist jeweils möglich.

Weitere Informationen zur COM-Sicherheit finden Sie in der MSDN Library [MSD01a].



Das DCOM Configuration Utility (DCOMCNFG) ermöglicht eine einfachere und komfortablere Anzeige und Einstellung der Sicherheitsinformationen, als dies über eine direkte Manipulation der Registry möglich wäre (siehe Kapitel 2.28.9).

Konfiguration der COM-Sicherheit

Im Rahmen der COM-Sicherheit sind folgende Punkte konfigurierbar:

- ▶ Die Authentifizierungsstufe legt fest, wie stark die Kommunikation zwischen COM-Client und COM-Server gesichert wird. Die Authentifizierungsstufe umfasst in COM nicht nur die Authentifizierung, sondern auch den Schutz der Integrität und Vertraulichkeit.
- ▶ Die Identität einer Klasse bestimmt, unter welchem Benutzerkontext (»Principal«) die Instanzen einer Klasse agieren. Mit einem Benutzerkontext ist eine bestimmte Menge von Benutzerrechten verbunden. Ein COM-Objekt kann alle Aktionen ausführen, die den Benutzerrechten des Principals entsprechen.
- ▶ Die Impersonifizierungsstufe legt fest, inwiefern die Benutzerrechte des COM-Clients auf den COM-Server übertragen werden können.
- ▶ Im Rahmen der Zugriffskontrolle ist konfigurierbar, wer eine COM-Klasse aktivieren, nutzen oder konfigurieren kann.

All diese Einstellungen werden auf der Ebene einer COM-Anwendung konfiguriert und gelten damit für alle zu der COM-Anwendung gehörenden Klassen. Die Einstellungen können nicht direkt auf Klassenebene vorgenommen werden.

Mit Ausnahme der Identität können die Sicherheitseinstellungen auch global vorgegeben werden. Diese Standardeinstellungen gelten für alle nicht näher konfigurierten Komponenten. Die Standardeinstellungen liegen in der Registry unter HKEY_LOCAL_MACHINE\SOFTWARE\MICROSOFT\OLE\.

Neben der deklarativen Sicherheit, also der Vorkonfiguration der Sicherheitseinstellungen, unterstützt COM auch programmatische Sicherheit, also das Lesen und Ändern von Sicherheitseinstellungen zur Laufzeit eines Programms. Diese Möglichkeit wird jedoch nur von wenigen Komponenten genutzt. Einen kompletten Zugriff auf die DCOM-Sicherheit bietet derzeit nur die WMI-Komponente.

Programmatische Sicherheit

2.22.1 Authentifizierung

Die folgende Tabelle zeigt die verfügbaren Authentifizierungsstufen.

Name	Bedeutung	Wert
Default (Standard)	Es wird die gegenwärtige Verbindungsauthentifizierung verwendet.	0
None (kein)	Keine Authentifizierung	1
Connect (Verbinden)	Authentifizierung beim ersten Methodenaufruf, später kein Austausch mehr. Ein Angriff durch Nachrichtenwiederholung ist möglich.	2
Call (Aufruf)	Authentifizierung bei jedem RPC	3
Packet (Paket)	Zusätzlich: Verschlüsselung jedes einzelnen Netzwerkpakets (verbesserter Schutz gegen Nachrichtenwiederholung)	4
Packet Integrity (Paketintegrität)	Zusätzlich: Prüfsumme über Paketinhalt schützt vor Verfälschung der Nachrichten	5
Privacy (Paketvertraulichkeit)	Zusätzlich: Verschlüsselung des Paketinhalts (schützt Vertraulichkeit)	6

Tabelle 2.9:
COM-Authentifizierungsstufen

Die Stufen *Packet Integrity* und *Privacy* bieten zwar deutlich mehr Sicherheit, führen aber zu einer Verschlechterung der Performance und einer Erhöhung der Netzwerklast, da jedes übertragene Byte von dem Security Support Provider (SSP) verarbeitet werden muss.

2.22.2 Identität

Für jede einzelne COM-Anwendung kann in dem zugehörigen APPID-Schlüssel im Unterschlüssel RUNAS spezifiziert werden, unter welchem Benutzerkontext die COM-Anwendung ausgeführt werden soll.

Identitätseinstellungen

Für die Identität gibt es drei Möglichkeiten:

- **Interaktiver Benutzer:** Interaktiver Benutzer ist der Benutzer, der sich gerade an dem jeweiligen System angemeldet hat. Für den Client ist also nicht determinierbar, unter welchem Benutzer die Komponente ausgeführt wird. RUNAS muss auf »InteractiveUser« gesetzt werden. Wenn sich kein Benutzer angemeldet hat, kann die Komponente nicht ausgeführt werden.

- **Benutzer, der die Anwendung startet:** Dies ist die Standardeinstellung (RUNAS ist nicht gesetzt). Das COM-Objekt benutzt während der Ausführung den Benutzerkontext, unter dem auch der aufrufende Client ausgeführt wird. Für jeden Benutzer, der sich mit einer COM-Anwendung verbindet, muss Windows NT eine neue *Interactive Window-Session* eröffnen.
- **Dezidiierter Benutzer:** Das COM-Objekt wird unter einem bestimmten Benutzerkonto ausgeführt, unabhängig davon, wer das Objekt nutzt. Eine Sonderform gibt es für NT-Dienste, die auch unter dem Systemkonto laufen dürfen.

Auch in diesem Fall wird für jeden Benutzer eine interaktive Window-Session benötigt. Allerdings ist hier die Anzahl der nötigen Window-Sessions auf dem COM-Server bestimmbar und unabhängig von der Anzahl der aufrufenden Clients.

Abbildung 2.22:
Identitätseinstellung in DCOM
CNFG für die
WMI-Komponente



Diese Einstellung ermöglicht es auch, Zugriffsrechte zu kapseln. Indem die Komponente durch Zuweisung eines entsprechenden dezidierten Benutzerkontos mehr Rechte erhält, kann ein Benutzer über wohldefinierte Schnittstellen einzelne Aktionen ausführen, die über seine eigenen Rechte hinausgehen.

Einem Benutzerkonto, das als dezidiertes Benutzerkonto für COM-Anwendungen verwendet wird, muss im Benutzermanager bzw. in den Windows2000-Sicherheitsrichtlinien das Recht ANMELDEN ALS STAPEL VERARBEITUNGS-AUFTRAG zugewiesen werden.



2.22.3 Impersonifizierung

Die Impersonifizierungsstufe bestimmt, inwiefern ein Objekt Informationen über den Aufrufer erhält und inwiefern es unter dessen Benutzerrechten agieren kann.

Benutzerkontextwechsel

Impersonifizierung (engl. Impersonation) bezeichnet die Fähigkeit einer Softwareroutine, den Benutzerkontext, unter dem sie agiert, zu wechseln. Einige Autoren benutzen statt des Kunstwortes »Impersonifikation« auch die Begriffe »Imitation« oder »Identitätswechsel«.

Die nachfolgende Tabelle zeigt die möglichen Impersonifizierungsstufen. Standard ist die Stufe *Identify*. *Delegate* wird erst ab Windows2000 unterstützt, sofern Kerberos als SSP verwendet wird. Das Problem mit diesem Modus ist, dass die Rechte über eine endlose Kette weitergegeben werden können, so dass Aktionen, die unter dem Recht des Benutzers aufgerufen werden, kaum mehr kontrollierbar sind.

Name	Bedeutung	Wert
Anonymous	Das COM-Objekt hat keinen Zugriff auf die Anmeldedaten des Aufrufers. Es hat keine Möglichkeit, in Erfahrung zu bringen, wer das Objekt aufgerufen hat.	1
Identify	Das COM-Objekt kann die Anmeldedaten abfragen.	2
Impersonate	Das COM-Objekt kann die Anmeldedaten ermitteln und auf Betriebssystemebene Operationen unter Verwendung des Benutzerkontextes des Aufrufers ausführen.	3
Delegate	Das COM-Objekt kann auf alle Ressourcen unter Verwendung des Benutzerkontextes des Aufrufers zugreifen.	4

Tabelle 2.10:
Impersonifizierungsstufen

Eine Aufrufkette kann beliebig lang werden, da COM-Klassen wiederum COM-Klassen aufrufen können.

2.22.4 Zugriffsberechtigungen

Im Rahmen der Zugriffskontrolle ist konfigurierbar, wer die zu einer COM-Anwendung gehörenden COM-Klassen aktivieren, nutzen oder konfigurieren

ren kann. COM unterscheidet folgende Sicherheitseinstellungen zu einer COM-Anwendung:

Startberechtigungen	► Die Startberechtigungen legen fest, wer Instanzen der Klassen einer COM-Anwendung erzeugen darf. Mögliche Rechte sind <i>Starten zulassen</i> und <i>Starten verweigern</i> .
Zugriffsberechtigungen	► Die Zugriffsberechtigungen bestimmen, wer auf die laufenden Instanzen der Klassen einer COM-Anwendung zugreifen darf. Mögliche Rechte sind <i>Zugriff erlauben</i> und <i>Zugriff verweigern</i> .
Konfigurationsberechtigungen	► Die Konfigurationsberechtigungen regeln, wer die Sicherheitseinstellungen einer COM-Anwendung verändern darf. Mögliche Rechte sind <i>Lesen</i> und <i>Uneingeschränkter Zugriff</i> .

Start- und Zugriffsberechtigungen werden für jede COM-Anwendung in der Registry unterhalb ihres AppID-Eintrags in Form von drei Access Control Lists (ACLs) abgespeichert. Die Konfigurationsberechtigungen entsprechen der ACL des AppID-Eintrags in der Registry selbst. Diese Zugriffsrechte sind über die erweiterten Sicherheitseinstellungen RegEdt32 in feinerer Granularität konfigurierbar als über DCOMCNFG, das nur die Stufen *Lesen* und *Uneingeschränkter Zugriff* (in RegEdt32 *Vollzugriff* genannt) zulässt.

Zugriffsrechte auf Schnittstellen- und Methodenebene

Programmgesteuert können die Zugriffsrechte auch auf Schnittstellenebene geprüft werden. Dazu dienen die Standardschnittstellen *IClientSecurity* und *IServerSecurity*. In COM+ unter Windows2000 und Folgeversionen ist die Konfiguration von Zugriffsrechten auf Methodenebene möglich.

Abbildung 2.23:
Sicherheitseinstellungen in DCOMCNFG



Sicherheit unter Windows 95/98/ME

Unter Windows 95/98/ME gibt es keine COM-Sicherheit bei der Nutzung lokaler Komponenten. Die Sicherheitsfunktionen greifen nur für die Fernnutzung und auch nur dann, wenn sich das Betriebssystem im Sicherheitsmodus »Zugriffskontrolle auf Benutzerebene« befindet. Die Zugriffskontrolle auf Ressourcenebene wird nicht unterstützt. Sofern die Zugriffskontrolle auf Ressourcenebene eingestellt ist, besteht nur die Möglichkeit, die Fernaktivierung von Komponenten grundsätzlich zu erlauben oder zu verbieten. Auch im Modus »Zugriffskontrolle auf Benutzerebene« sind nur die Standardzugriffsberechtigungen konfigurierbar. Die Authentifizierungs-Unterstützung in Windows 95/98/ME ist eingeschränkt, da eingehende Aufrufe höchstens bis zur Stufe »Connect« entgegengenommen werden können. Aufrufe mit einer höheren Stufe werden abgewiesen. Weitere Informationen dazu finden Sie im COM Security FAQ [Q158508].

Windows 9x/ME

2.23 Active Scripting

Die Windows Scripting-Architektur heißt bei Microsoft *Active Scripting* oder *ActiveX Scripting*. Manchmal spricht Microsoft auch von *Windows Script*.

Active Scripting

Die Active Scripting-Architektur besteht aus folgenden drei Bausteinen:

Bausteine

- ▶ **Active Scripting Hosts** sind die Ablaufumgebungen für Skripte.
- ▶ **Active Scripting Engines** stellen einen Sprachinterpreter für eine bestimmte Skriptsprache bereit.
- ▶ **COM-Komponenten** ermöglichen den Zugriff auf Systemkomponenten oder stellen in gekapselter Form bestimmte Funktionalitäten bereit.

Ein Skript, das innerhalb der Active Scripting-Architektur ausgeführt wird, heißt *Active Script*.

Active Script

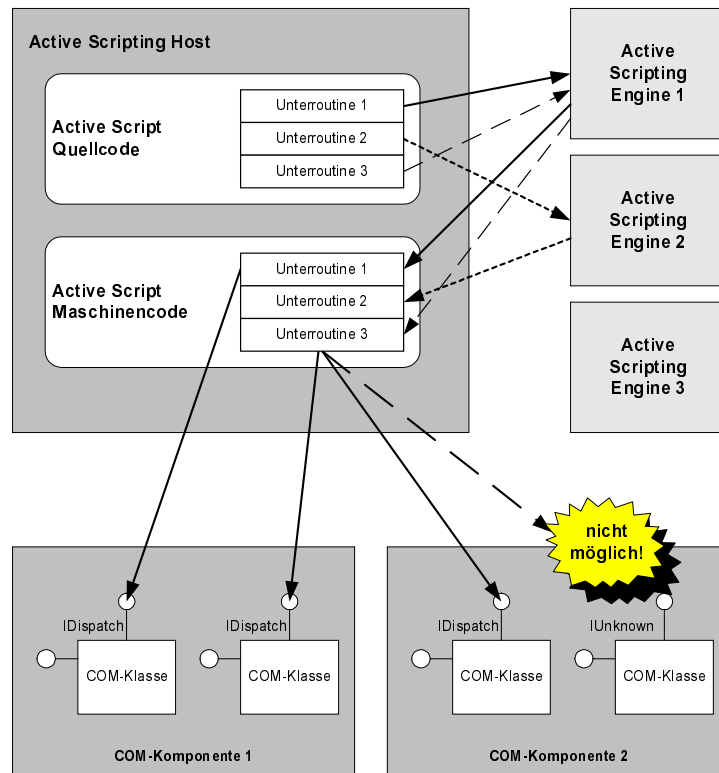
Die Active Scripting-Architektur ist so konzipiert, dass die einzelnen Bausteine untereinander austauschbar sind: Jeder Scripting Host kann jede Scripting Engine verwenden. Jede automationsfähige COM-Komponente kann von jedem Scripting Host und jeder Skriptsprache benutzt werden. Dies wird über wohldefinierte Schnittstellen sichergestellt. Damit ist Plug&Play zwischen Hosts und Engines verschiedener Hersteller möglich. Die Skriptsprache kann auch innerhalb eines einzigen Skripts variieren; so können etwa einzelne Unterrouinen in einer anderen Sprache als das Hauptprogramm geschrieben werden, wenn die besten Features der jeweiligen Sprache genutzt werden sollen.

Plug&Play

Auch Scripting Hosts und Scripting Engines sind COM-Komponenten, die spezielle Schnittstellen implementieren. Dass die Architektur *Active Scripting* und nicht *COM Scripting* heißt, beruht darauf, dass Microsoft aus Marketinggründen den Begriff ActiveX gefördert hat. ActiveX wurde eine Zeit lang mit dem Begriff COM sogar völlig gleichgesetzt.

COM versus
ActiveX

Abbildung 2.24:
Die Active Scripting-Architektur



(C) Holger Schwichtenberg 2001

Die COM-Komponenten müssen COM-Automation unterstützen, damit sie in dieser Architektur verwendet werden können.

2.23.1 Active Scripting Hosts

Ablaufumgebung

Ein Scripting Host ist die Ablaufumgebung für ein Skript und insofern vergleichbar mit den Shells unter Unix. Der Internet Explorer war der erste Scripting Host überhaupt; mit dem Windows Script Host (WSH) gibt es inzwischen einen eigenständigen Scripting Host für die Windows-Plattform. Der Windows Script Host sollte keineswegs mit dem allgemeinen Begriff Scripting Host verwechselt werden. Der WSH ist nur einer von vielen Active Scripting Hosts. In Version 1.0 hieß der WSH noch Windows Scripting Host. Wohl aus Gründen der besseren namentlichen Abgrenzbarkeit nennt Microsoft diesen Scripting Host seit Version 2.0 *Windows Script Host*.

Active Scripting

Microsoft integriert Active Scripting Hosts in immer mehr Produkte, insbesondere in solche aus der BackOffice-Reihe:

Verfügbare Scripting Hosts

- ▶ Active Server Pages im Internet Information Server (IIS) ab Version 3.0
- ▶ Event Scripting Agent im Exchange Server ab Version 5.5
- ▶ Server-Agent im SQL Server 7.0
- ▶ Data Transformation Scripts im SQL 7.0-Data Transformation Service
- ▶ Dynamic HTML-Scripting im Internet Explorer ab Version 3.0
- ▶ Outlook Forms in Microsoft Outlook ab Version 8.0
- ▶ XSL-Scripting im Microsoft XSL-Processor
- ▶ Installer Scripts im Windows Installer
- ▶ Transformations-Skripte im Microsoft BizTalk Server
- ▶ Scriptor Component im Microsoft Commerce Server
- ▶ Skripte im Microsoft Operations Manager (MOM)

Scripting Hosts werden inzwischen auch von anderen Anbietern bereitgestellt, z.B.

Script Control

- ▶ der System Scripting Host (SSH) der Firma Netal [NET00]
- ▶ der Hyperhost der Firma Dundas [DUN00]
- ▶ das Text-Transformationswerkzeug Textpipe [TXP00]

Mit dem ActiveX-Steuerelement *Microsoft Script Control* bietet Microsoft zudem die Möglichkeit, auf einfache Weise in eigene Anwendungen einen Scripting Host zu integrieren. Das Script Control kann zum Beispiel in Visual Basic 6.0 und Visual C++ 6.0 verwendet werden.



Zwar sind die VBA-Umgebungen (VBA steht für *Visual Basic for Applications*) (noch) nicht nach der Windows Scripting-Architektur konstruiert, in der Praxis sind sich die Architekturen aber sehr nahe: Ein Skriptprogrammierer, der auf Visual Basic Script (VBScript) setzt, kann mit Cut&Paste des Quelltexts sowie mit ein paar einfachen Änderungen seine Skripte auch in VBA laufen lassen. Mit anderen Skriptsprachen geht das allerdings nicht.

VBA

2.23.2 Active Scripting Engines

Eine Active Scripting Engine ist ein Sprachinterpret für eine Skriptsprache mit der Nebenbedingung, dass der Interpreter

Sprachinterpret

- ▶ in Form einer COM-Komponente vorliegt,
- ▶ bestimmte Schnittstellen implementiert und
- ▶ für eine der entsprechenden Komponentenkategorien registriert ist.

COM-fähige
Sprache versus
ActiveX-Skript-
sprache

Diese Anforderung ist abzugrenzen von einer COM-fähigen Programmiersprache (Microsoft spricht von »COM-enabled Languages« – ins Deutsche zum Teil mit »COM-aktivierten Sprachen« übersetzt). Eine COM-fähige Sprache unterstützt die Nutzung von COM-Komponenten. Nicht jede COM-fähige Sprache ist auch eine Active Scripting-fähige Sprache. Beispielsweise unterstützen auch Delphi und PHP4 die Nutzung von COM-Komponenten. Dennoch sind beide Sprachen nicht im Rahmen des Active Scripting als Skriptsprache einsetzbar, weil die Sprachinterpreter selbst keine COM-Komponenten sind.

Verfügbare Skriptsprachen

Microsoft selbst hat bislang zwei Active Scripting Engines veröffentlicht:

- VBScript** ▶ VBScript (eine abgespeckte Version der Programmiersprache Visual Basic)
- JScript** ▶ JScript (eine Erweiterung der auf Netscape JavaScript basierenden Sprachspezifikation ECMA 262, die auch ECMAScript genannt wird).

Es gibt weitere Sprachen von anderen Anbietern (zum Großteil als Free- oder Shareware):

- Perl** ▶ *PerlScript*, Active Scripting-fähige Perl-Implementierung der Firma ActiveState (siehe [ACT00])
- ▶ *PScript*, Active Scripting-fähige Perl-Implementierung der Firma MKS (siehe [MKS00])
- REXX** ▶ IBM unterstützt im Rahmen seiner REXX-Implementierung unter dem Namen *Object REXX* Active Scripting. Object REXX ist seit Version 2.1 eine Active Scripting Engine (siehe [IBM01] und [CAW01]).
- Python** ▶ *PythonScript*, Active Scripting-fähige Version von *Python* (siehe [PYT00a] und [PYT00b])
- Haskell** ▶ *HaskellScript*, Active Scripting-fähige Version der funktionalen Skriptsprache Haskell (siehe [HAS00])
- Ruby** ▶ *ActiveScriptRuby*, Active Scripting-fähige Implementierung der objekt-orientierten Skriptsprache *Ruby* (siehe [RUB01a] und [RUB01b])
- LUA** ▶ *LUAScript*, Active Scripting-fähige Implementierung der in Brasilien entwickelten Sprache LUA (siehe [LUA01a] und [LUA01b])

Gerüchte um die Active Scripting-fähigen Implementierungen von Lisp und TCL konnten zum Zeitpunkt des Redaktionsschlusses dieses Buchs nicht bestätigt werden.

VBScript

In diesem Buch wird nur VBScript benutzt. VBScript ist die am häufigsten verwendete Sprache beim Windows Scripting. Auch aufgrund der weitgehenden Kompatibilität mit der Vollversion von Visual Basic ist VBScript die erste Wahl bei den Skriptsprachen unter Windows.

Entwicklung von Scripting Engines

Die Implementierung einer Skriptsprache ist dann kompatibel mit der Active Scripting-Architektur, wenn folgende Voraussetzungen erfüllt sind:

- ▶ Die Sprache ist in Form einer COM-Klasse implementiert.
- ▶ Die COM-Klasse unterstützt die COM-Schnittstelle `IActiveScriptParse`.
- ▶ Mit Sprachen, die COM-Komponenten mit Mehrfachschnittstellen implementieren können, kann man eigene Active Scripting Engines erstellen. Dies ist jedoch nicht Thema dieses Buches.

IActiveScriptParse

Installierte Skriptsprachen

Skriptsprachen werden bei der Installation in der Registry unter einer speziellen Kategorie von Komponenten registriert. Am einfachsten kann man die auf einem System installierten Skriptsprachen mit Hilfe des Tools COM-Viewer (siehe Kapitel 2.28.4) einsehen. Es gibt dort mehrere Einträge unter ACTIVE SCRIPTING ENGINE, die die Sprachen nach ihren Fähigkeiten kategorisieren.

Ansicht der installierten Skriptsprachen

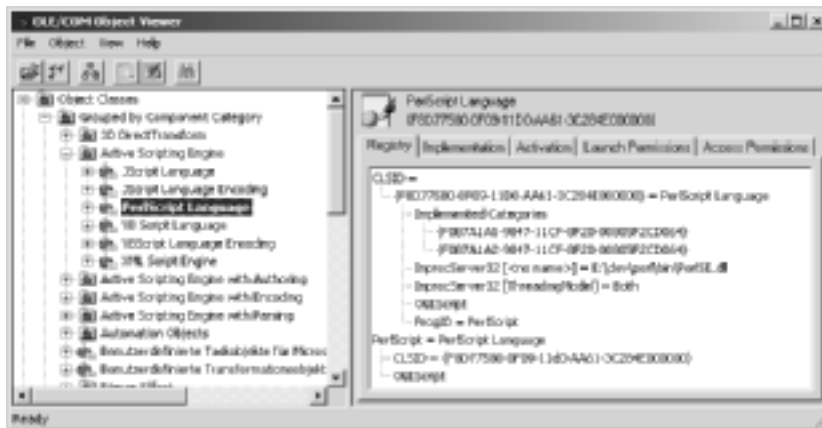
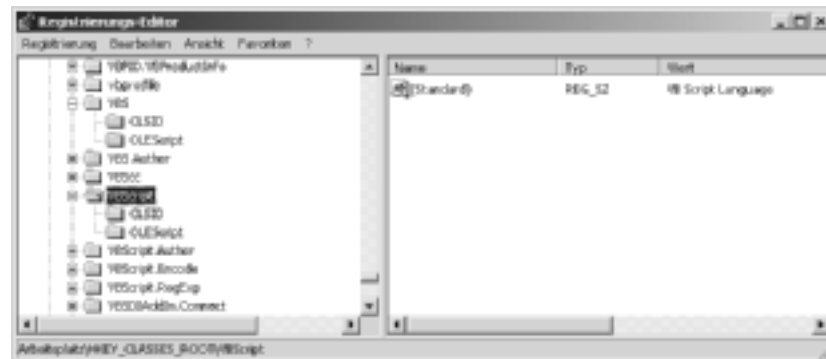


Abbildung 2.25:
Anzeige der installierten Skriptsprachen im COM Viewer

Zu jeder Sprache werden Sie einen Eintrag PROGID finden. Diese ProgID (Programmatic Identifier) ist in einigen Scripting Hosts wichtig zur Identifizierung der Skriptsprache. Visual Basic Script hat die ProgID `VBScript` und die Sprache JScript die ProgID `JScript`. Über einen Registry-Schlüssel der Form `HKEY_CLASSES_ROOT\PROGID` wird der Zusammenhang zwischen der Sprache und der Binärdatei hergestellt, die die Sprache implementiert.

ProgID der Sprache

Abbildung 2.26:
VBScript-ProgID in
der Registry



2.23.3 COM-Komponenten beim Active Scripting

Nur COM-Automation

Die Active Scripting-Architektur ermöglicht allen Active Scripting-Sprachen, auf alle COM-Komponenten zuzugreifen, die COM-Automation unterstützen. COM-Klassen, die keine IDispatch-Schnittstelle haben, können nicht genutzt werden. Auch wenn viele COM-Komponenten COM-Automation unterstützen, gibt es Komponenten, die den Dienst nicht anbieten und daher im Windows Scripting nicht verwendbar sind.

Auf einen Zugriff auf Nicht-COM-APIs (z.B. DLLs, die keine COM-Komponenten sind) hat Microsoft ausdrücklich verzichtet. Es gibt jedoch inzwischen Ansätze, dies zu ermöglichen (DynaWrap, siehe [BOR00]).

Intrinsic Objects

Eingebaute Objekte

Viele Scripting Hosts besitzen sogenannte *Intrinsic Objects* (auch: *Built-In Objects* oder *Internal Objects*) für den Zugriff auf den Scripting Host und dessen direkte Umgebung. Intrinsic Objects sind COM-Objekte, die zu Beginn beim Start des Scripting Hosts oder beim Start eines konkreten Skripts automatisch durch den Scripting Host instanziiert werden. Dem Skript steht die so erzeugte Instanz in Form eines bestimmten Bezeichners als Intrinsic Object zur Verfügung.

Beispiele für Intrinsic Objects in verschiedenen Scripting Hosts

- Die *Active Server Pages (ASP)* stellen eine Reihe von Intrinsic Objects zur Kommunikation mit dem Webserver zur Verfügung (z.B. Request, Response, Server).
- Der *Windows Scripting Host (WSH)* liefert ein WScript-Objekt zum Zugriff auf den Host und Informationen über das aufgerufene Skript.
- Der *Exchange Event Agent* bekommt über das Objekt EventDetails Zugriff auf die E-Mail, die seinen Aufruf getriggert hat, und auf den Ordner, in dem die Nachricht steht.
- Ein *Transformation Script* im Data Transformation Service (DTS) hat über die Objekte DTSSource und DTSDestination die nötigen Informationen über die an der Transformation beteiligten Daten.

Jeder Active Scripting Host hat andere Intrinsic Objects. Oft stehen die dazugehörigen Klassen nur diesem Scripting Host zur Verfügung, so dass eine Instanziierung durch andere Scripting Hosts oder Anwendungen unmöglich ist. Es kann aber natürlich auch vorkommen, dass eine Klasse, welche in einem Scripting Host ein Intrinsic Object ist, in einem anderen Scripting Host durch explizite Instanziierung erzeugt wird.

2.24 Windows Script Components (WSCs)

Windows Script Components (WSC) sind COM-Komponenten, die aus Skriptcode bestehen. Sie werden im Quellcode weitergegeben und interpretiert. *Windows Script Components* sind White-Box-Komponenten im Gegensatz zu kompilierten Komponenten (Black-Box-Komponenten), die ihre Implementierung verbergen (vgl. [BÜC97]).

Quellcodekomponenten

Interpretierte Scriptlets sind natürlich viel langsamer als kompilierte COM-Objekte, eignen sich jedoch hervorragend für das Rapid Application Development (RAD), da eine Kompilierung in einer Entwicklungsumgebung nicht notwendig ist.

Scriptlet

Eine alternative Bezeichnung für Windows Script Components ist der Begriff *Scriptlet*. Es gibt aber neben den WSCs auch sogenannte *DHTML-Scriptlets*. Diese Scriptlets bestehen aus HTML und Skriptcode. Die Implementierung beruht auf JavaScript. Ein DHTML-Scriptlet entspricht einer HTML-Seite. DHTML-Scriptlets sind keine COM-Komponenten. Sie wurden mit dem Internet Explorer 4.0 eingeführt. WSCs sind eine später als Add-on eingeführte Verallgemeinerung der DHTML-Scriptlets.

Manchmal wird auch von XML-Scriptlets oder COM-Scriptlets synonym zum Begriff WSC gesprochen.

XML-Struktur

Eine WSC wird in Form einer XML-Datei implementiert, die Skriptcode enthält. Eine Windows Script Component ist eine Datei mit der Extension .WSC (es gibt auch noch die ältere Extension .SCT, die aus der Zeit stammt, als eine Script Component noch Scriptlet hieß). Eine WSC-Datei ist XML-strukturiert. Sie kann mehrere Klassen enthalten, wobei diese Klassen ganz verschiedene ProgIDs haben können. Jede Klasse besteht aus genau einer Schnittstelle und einer beliebigen Anzahl von Methoden, Attributen und Ereignissen. Die Klasse besitzt eine CLSID und eine ProgID. Auch diese Informationen sind in XML-Form abgelegt.

Dateiformat

COM-Zusatzdienst *Script Component Runtime*

Windows Script Components sind ein COM-Zusatzdienst, der durch die *Script Component Runtime* (SCR.DLL) bereitgestellt wird. Die Script Component Runtime ist selbst eine COM-Komponente, was für das Funktionieren von Windows Script Components auch zwingend notwendig ist.

ScrObj.dll Eine WSC-Datei wird nicht kompiliert. Die CLSID wird wie eine normale COM-Komponente registriert. Im Schlüssel INPROCSERVER32 wird jedoch nicht der Pfad zu der WSC-Datei, sondern der zu SCR.DLL eingetragen. Die Script Component Runtime nimmt den Aktivierungsauftrag eines COM-Clients für die Script Component entgegen und bildet den Scripting Host für die Skripte in der WSC-Datei. Dazu muss die Script Component Runtime natürlich wissen, wo die WSC-Datei liegt. Diese Information enthält der besondere Unterschlüssel SCRIPTLETURL unterhalb des CLSID-Eintrags der Script Component. Anhand der XML-Informationen der WSC-Datei kann die Script Component Runtime die Methodenaufrufe an die entsprechenden Skriptblöcke der WSC-Datei weiterleiten. Da die Script Component Runtime ein In-process-Server ist, sind Script Components normalerweise In-process-Komponenten. Sie können jedoch genau wie die anderen COM-DLLs auch mit Hilfe des MTS bzw. COM+ als Out-process-Komponenten verwendet werden.



Die Erzeugung eigener Windows Script Components ist ausführlich in [SCH01c] beschrieben.

2.25 Microsoft Transaction Server (MTS)

MTS Den *Microsoft Transaction Server* (MTS) hätte man besser *Microsoft Component Server* nennen sollen, denn den wesentlichen Mehrwert bietet er in der Bereitstellung einer Umgebung für COM-Komponenten und erst in zweiter Linie für die Transaktionsverarbeitung. Marketingtechnisch war der Name MTS aber sicherlich erste Wahl, denn Microsoft wollte das imageträchtige Feld der Transaktionsverarbeitung mit einem eigenen Produkt besetzen.

Der MTS Version 2.0 kann unter NT4 im Rahmen des Option Packs installiert werden. Unter Windows 95/98/ME erhalten Sie den MTS durch Installation des Personal Webserver. In Windows2000/XP/.NET ist die MTS-Funktionalität ein Teil von COM+ und damit fest im Betriebssystem verankert.



Zur Verwendung der MTS-Funktionalität unter Windows2000 beachten Sie bitte auch die Ausführungen zu COM+ im nächsten Kapitel, da sich einige Aspekte des MTS in Windows2000 geändert haben.

DNA Microsofts *Distributed InterNet Applications Architecture* (DNA) sieht eine moderne Unterteilung in Präsentationsschicht, Anwendungsschicht und

Datenzugriffsschicht vor. Diese Architektur heißt auch Three-Tier-Architecture, die Anwendungsschicht, auf der der MTS zuhause ist, wird auch Middle-Tier genannt. Die Anwendungsschicht beherbergt Geschäftsobjekte, die die von der Präsentationsschicht kommenden Benutzereingaben verarbeiten. Dabei sind in der Regel Zugriffe auf Datenbanken über die Datenzugriffsschicht notwendig.

Die Transaktionssteuerungsfunktion des MTS soll hier nicht thematisiert werden, da dies vor allem die Programmierung von eigenen COM-Servern betrifft. Zum Thema Transaktionsprogrammierung mit dem MTS sei auf [SCH00a] verwiesen. Der MTS wird hier aus drei Gründen behandelt:

- ▶ Der MTS übernimmt für COM-DLLs die Funktion eines Surrogat-Prozesses, was die Möglichkeit eröffnet, COM-DLLs als Out-process- oder Remote-Komponenten zu nutzen.
- ▶ Mit Hilfe des MTS sind erweiterte, rollenbasierte Sicherheitseinstellungen für Komponenten möglich.
- ▶ Der MTS kann per Programmcode administriert werden. Dies wird in Kapitel 15 behandelt.

2.25.1 MTS-Administration

Die Administration des MTS erfolgt unter NT4 über ein Snap-In für die Microsoft Management Console (MMC) mit dem Namen Transaction Server Explorer. Den MTS Explorer gibt es auch als Stand-alone-Anwendung (MTXEXP.EXE) für Windows 95/98/ME – allerdings mit anderer Oberfläche und eingeschränkten Möglichkeiten.

MTS Explorer

Der MTS-Explorer wird in Kapitel 2.28.10 behandelt.

2.25.2 Interception

Der MTS arbeitet nach dem Interception-Prinzip: COM-Client und COM-Server kommunizieren nicht direkt, sondern über ein Interceptor-Objekt miteinander. Das Interceptor-Objekt ist wie ein Mantel um ein COM-Objekt gelegt, der die Methodenaufrufe des COM-Clients entgegennimmt und an das angesprochene COM-Objekt weiterleitet. Interception wird auch *Objekt-kontext* genannt, das Interceptor-Objekt heißt auch *Kontextobjekt* oder *Context-Wrapper-Objekt*. Interception erfolgt auf Klassenebene, nicht auf Komponentenebene.

Interception

Der MTS fügt dazu zwischen den Server-Stub und die Klasse ein Context-Wrapper-Objekt ein, wobei dies deutliche Spuren in der Registry hinterlässt: Der LOCALSERVER32-Eintrag unterhalb der CLSIDs der Klassen weist nicht mehr auf die DLL, sondern auf einen Eintrag der Form `C:\WINDOWS\SYSTEM\mtx.exe /p:{8FBA079C-B9A4-11D2-978B-0008C73ADEC0}`. Hinter MTX.EXE verbirgt sich der Context-Wrapper, der gleichzeitig ein Surrogat-Prozess ist. Der MTS startet pro Package eine Instanz von MTX.EXE; alle Objekte eines Packages laufen also im gleichen Prozess.

mtx.exe

Zusätzliche Dienste Neben den bereits erwähnten Transaktionsdiensten, dem Surrogat-Prozess und den erweiterten, rollenbasierten Sicherheitseinstellungen bietet der MTS auf Basis der Interception auch noch folgende Dienste an:

- ▶ **Connection Pooling für Datenbankverbindungen:** Wenn ein MTS-Objekt eine Datenbankverbindung verwirft, wird diese durch den Objektkontext nicht wirklich geschlossen, sondern für die weitere Verwendung vorgehalten.
- ▶ **Shared Properties:** Der Shared Property Manager ermöglicht gemeinsame Datenbereiche für alle Instanzen (auch verschiedener Klassen) innerhalb eines Packages.
- ▶ **Just-in-Time-Activation:** Der MTS 2.0 kennt einen Object Caching-Mechanismus (Just-in-Time-Activation) – Komponenten verbleiben auch nach Deaktivierung der letzten Objektinstanz noch für eine bestimmte, definierbare Zeit im Speicher, um bei einer erneuten Anfrage schneller neue Instanzen bilden zu können. Dieser Mechanismus ist auch der Grund dafür, dass Entwickler oft vergeblich versuchen, COM-DLLs zu ersetzen. Solange die pro Package einstellbare Caching-Zeit seit der letzten Verwendung nicht verstrichen ist, muss das Package im MTS Explorer manuell – mit Hilfe des Kontextmenüeintrags HERUNTERFAHREN – entladen werden. Object Caching ist kein Object Pooling, bei dem die einzelnen Instanzen erhalten bleiben. Dies ist erst in COM+ implementiert. Just-in-Time-Activation kann seine Wirkung erst in Zusammenhang mit zustandslosen Komponenten voll entfalten.

2.25.3 Packages

MTS-Packages Der MTS unterscheidet zwischen nichtkonfigurierten und konfigurierten Klassen. Erstere sind Klassen, die lediglich installiert, also in der Registry als COM-Klasse registriert sind. Sie sind jedoch nicht Teil eines MTS-Packages. Konfigurierte Klassen sind in einem sogenannten MTS-Package enthalten.

Ein Package ist die zentrale Organisationseinheit des MTS. Es besteht aus einer oder mehreren COM-Klassen. Unterstützt werden nur Klassen aus COM-In-process-Komponenten (d.h. COM-DLLs). Jede Klasse gehört zu höchstens einem Package. Nur Klassen, die in einem Package eingetragen sind, können die MTS-Zusatzdienste nutzen.

Es gibt zwei Arten von Packages:

- Library Packages** ▶ Bei **Library Packages** laufen die Klassen im Prozess des Aufrufers, was voraussetzt, dass sich der aufrufende Prozess auf dem gleichen Computer befindet.
- Server Packages** ▶ Der Normalfall sind jedoch die **Server Packages**, die in einem eigenen Prozess laufen. Nur dieser Typ unterstützt alle Features.

Der gesamte Datenspeicher des MTS heißt *MTS-Katalog*.

2.25.4 Programmierung

Die Instanzen einer Klasse, die im MTS bzw. als COM+-Anwendung laufen, können Zugriff auf die Interception-Umgebung erhalten: Nach Einbindung der Microsoft Transaction Server Type Library (MTXAS.DLL) steht die globale Funktion `GetObjectContext()` zur Verfügung, die einen Zeiger auf ein `ObjectContext`-Objekt mit einer `IObjectContext`-Schnittstelle liefert.

GetObjectContext()

Die Schnittstelle bietet hauptsächlich Funktionen zur Transaktionssteuerung (`SetComplete`, `SetAbort`, `EnableCommit`, `EnableAbort`, `IsInTransaction`) an. Während `SetComplete` und `SetAbort` das endgültige Ende einer Transaktion festlegen, kann mit `EnableCommit` und `EnableAbort` ein Zwischenstatus gesetzt werden. Zusätzlich kann der Komponentenprogrammierer Sicherheitsinformationen auslesen.

Die zuvor beschriebene Form der Sicherheitseinstellung wird in COM auch als deklarative Sicherheit bezeichnet, bei der programmatische Sicherheit wird dagegen innerhalb der COM-Objekte auf Sicherheitsfunktionen zugegriffen. MTS stellt das Rollenkonzept auch für die programmatische Sicherheit bereit. Innerhalb einer Klasse kann der Komponentenprogrammierer über die `IObjectContext`-Schnittstelle auf die Sicherheitsinformationen zugreifen. `IObjectContext` bietet die Methoden `IsSecurityEnabled()` und `IsCallerInRole()` sowie das Unterobjekt `Security` vom Typ `SecurityProperty`.

2.26 COM+

COM+ ist die Weiterentwicklung von COM, die mit Windows2000 eingeführt wurde. COM+ umfasst COM und den Microsoft Transaction Server sowie weitere Dienste. Weitere Informationen zu COM+ finden Sie in [SCH00a] und [PLA99].

COM+ in Windows2000

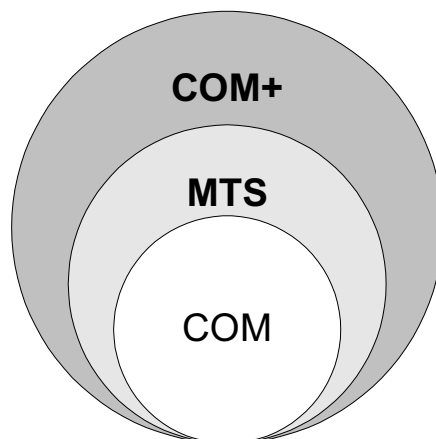


Abbildung 2.27:
Verhältnis von
COM, MTS und
COM+

2.26.1 Änderungen gegenüber dem MTS

MTS versus COM+	Die Funktionen des MTS 2.0 wurden aber nicht eins zu eins in COM+ übernommen. Folgende Dinge sind anders:
Application	► Das, was im MTS <i>Package</i> genannt wurde, heißt nun <i>COM+-Anwendung</i> . Der Name Application ist natürlich sehr ungünstig gewählt, da es dafür in der COM-Welt mindestens schon zwei Bedeutungen gibt.
Interception	► In COM+ unterliegt jede Klasse automatisch dem Interception-Modell. COM+ benötigt daher keine zwischengeschaltete MTX.EXE mehr. Die Interception findet hier bereits vor dem Client-Proxy und vor dem Server-Stub statt. Die Erhebung einer DLL in einen eigenen Prozess leistet das bereits seit NT4 Service Pack 2 ausgelieferte Standard-Surrogat DLLHOST.EXE. Für Anwendungen, die drei GB Speicher unterstützen, wird DLLHST3G.EXE verwendet. Für den Komponentenprogrammierer vereinfachen sich durch die automatische Interception einige Funktionen.
Datenspeicher	► Der <i>COM+ Catalog</i> (der zentrale Datenspeicher von COM+, der dem MTS Catalog entspricht) wird nicht in der Registry, sondern in der sogenannten COM+ Registration Database (RegDB) gespeichert. Die RegDB befindet sich im Verzeichnis /WINNT/REGISTRATION.
Administration	► Die Administration erfolgt über ein geändertes Snap-In, das im Rahmen der MMC-Konfiguration KOMPONENTENDIENSTE im Verwaltungsordner im Startmenü zu finden ist.
Rollen	► Bei der rollenbasierten Sicherheit unterstützt COM+ nun sogar die Zuordnung von Rollen auf Methodenebene.
Aufrufketten	► Die Möglichkeiten des Zugriffs auf die Identität der Aufrufer in einer Kette von Objektaufrufen wurde verbessert: Während bei MTS nur die Identität des direkten und die des ursprünglichen Aufrufers in einer Kette von Objektaufrufen ermittelbar ist, besteht unter COM+ Zugriff auf die komplette Liste der Benutzerkontexte, über die ein Aufruf gelaufen ist. COM+ bietet ein eigenes Objektmodell für den Zugriff auf die Sicherheitsinformationen. Den Einstieg bildet ein <i>SecurityCallContext</i> -Objekt, das wie das <i>ObjectContext</i> -Objekt über eine globale Methode (<i>GetSecurityCallContext()</i>) im Zugriff ist.
MSI	► COM+-Anwendungen können als ein Microsoft Windows Installer-Paket (.MSI) exportiert werden.
Objektmodelle	► Das Objektmodell für den Zugriff auf den Objektkontext hat sich geändert und heißt jetzt COM+ Service Type Library (COMSVCS.DLL). Unter COM+ verfügt das <i>ObjectContext</i>-Objekt über weitere Schnittstellen (<i>IObjectContextInfo</i>, <i>IContextState</i> und <i>ISecurityCallContext</i>), die den Zugriff auf Detailinformationen und ein Feintuning von Transaktionen ermöglichen. ► Ebenso hat sich das Objektmodell für die automatisierte Administration geändert. Es gibt nun eine Komponente mit Namen <i>COM+ Administration Objects</i> (vgl. Kapitel 15).

2.26.2 Neue Dienste in COM+

COM+ unterstützt die folgenden neuen Dienste:

- **Object Pooling:** Das Object Pooling geisterte schon in Zusammenhang mit dem MTS 2.0 durch die Fachwelt, war jedoch nicht implementiert. Object Pooling ist die Möglichkeit, einen bestimmten Vorrat an Instanzen dieser Ressourcen im Speicher vorzuhalten, die dann bei Bedarf aktiviert werden. In COM+ gibt es für das Object Pooling nun ein Eigenschaftsfenster, in dem eine Unter- und eine Obermenge von Instanzen einer jeden Klasse definiert werden können, die im Speicher gehalten werden. Allerdings funktioniert Object Pooling nur mit Objekten im Multithreaded oder Neutral Apartment – also nicht mit Visual Basic 6.0-Klassen.

Jede Instanz hat selbst die Möglichkeit zu kontrollieren, ob sie in den Pool aufgenommen werden möchte. Dazu dient die Schnittstelle `IObjectControl`.

Object Pooling
- **Object Construction String:** Eine einfache Möglichkeit zur Übergabe von Informationen an Klassen besteht in COM+ mit dem sogenannten *Object Construction String*. Im Komponentendienste-Snap-In kann zu jeder Klasse ein beliebiger String eingegeben werden, der der Klasse bei der Aktivierung übergeben wird, sofern sie die Schnittstelle `IObjectConstructionString` implementiert.

Object Construction String
- **Compensating Resource Manager:** Der Compensating Resource Manager ermöglicht es, Ressourcen, für die kein eigener Resource Manager zur Verfügung steht, in Transaktionen zu integrieren. So kann auch das Ergebnis eines Verarbeitungsprozesses, der gar nicht auf Datenbanken beruht, Teil einer Gesamttransaktion sein.

Compensating Resource Manager
- **COM+ Queued Components:** Queued Components sind eine Integration des Microsoft Message Queue Servers (MSMQ) und ermöglichen den asynchronen Aufruf von Methoden.

Queued Components
- **COM+ Events:** In Abänderung zum klassischen COM ermöglicht ein spezieller NT-Dienst, der COM+ Event Service, die Realisierung von lose gekoppelten Ereignissen. Dabei ist im Gegensatz zum Connection Point-Mechanismus (vgl. Kapitel 2.12) eine Filterung der Ereignisse möglich.

Lose gekoppelte Ereignisse

Nicht in Windows2000 enthaltene Dienste

Einige Dienste, die ursprünglich für COM+ Version 1.0 angekündigt waren, haben nicht Einzug in Windows2000 gehalten

- Component Load Balancing (CLB)

Die Lastverteilung zwischen verschiedenen Servern mit denselben installierten Komponenten war ursprünglich in Windows2000 implementiert, wurde aber im Zuge der Beta-Phase herausgenommen. Dieser Dienst ist nun für ein zukünftiges Add-on (Application Center Server) angekündigt.

CLB

IMDB	► In-Memory-Database (IMDB) Die Möglichkeit, eine relationale Datenbank im Hauptspeicher zu führen, ist ebenso wie das CLB nicht im ersten Release von Windows2000 enthalten.
Laufzeitumgebung	► Laufzeitumgebung Aus der ursprünglich in [KIR98] angekündigten einheitlichen Laufzeitumgebung für Komponenten und aus dem Class Store (Speicherung der Komponenteninformationen im Active Directory) ist in den COM+ Versionen 1.0 (Windows2000) und 1.5 (Windows XP) nichts geworden.
COM+ und .NET	Der Nachfolger von COM+ sollte ursprünglich COM+ Version 2.0 heißen. Daraus wurde das .NET-Framework. Einige Dienste von COM+ leben aber auch im .NET-Framework weiter.

2.26.3 COM+-Administration

Zusammensetzung einer COM+-Anwendung	Die Konfiguration von COM+-Anwendungen erfolgt über das Komponentendienst-Snap-In in der Management Console. Über das Snap-In kann auf entfernte COM+-Kataloge zugegriffen werden. Das Snap-In wird in Kapitel 2.28.11 beschrieben.

2.27 Objektmodelle in COM-Komponenten

COM-Objektmodelle	Unter den Entwicklern von COM-Komponenten ist die Verwendung von Objektmodellen sehr verbreitet. Die folgenden Ausführungen sind eine wichtige Grundlage für das Verständnis der in diesem Buch beschriebenen Objektmodelle administrativer Komponenten. Bitte beachten Sie auch die grafische Notation, die in Anhang B erläutert wird.

2.27.1 Objektorientierte Konzepte in COM

Hier soll kurz dargestellt werden, wie Polymorphismus, Mehrfachschnittstellen und Vererbung durch COM realisiert werden.

2.27.1.1 Polymorphismus in COM

Polymorphismus	COM unterstützt Polymorphismus in drei Formen: ► Zwei Schnittstellen, die von der gleichen Schnittstelle abgeleitet sind, verhalten sich polymorph zueinander. So ist es möglich, dass ein Aufruf von <code>QueryInterface()</code> auf jeder von <code>IUnknown</code> abgeleiteten Klasse funktioniert.

- Zwei Klassen (und damit deren Instanzen) sind hinsichtlich jener Schnittstellen polymorph (siehe Anhang A), die beide Klassen implementieren. So kann auch ein früh bindender COM-Client Instanzen unterschiedlicher Klassen verwenden.
- Schließlich sind zwei Klassen auch dann polymorph, wenn sie über IDispatch-Schnittstellen verfügen, die jeweils eine Methode mit dem gleichen Namen und der exakt gleichen Signatur bereitstellen. So kann ein COM-Client via Automation eine Methode `TueEtwas(was,wann)` auch dann erfolgreich in zwei Instanzen unterschiedlicher Klassen ausführen, wenn die Methode `TueEtwas()` in beiden Klassen zu verschiedenen benutzerdefinierten Interfaces gehört. Voraussetzung ist lediglich, dass beide Methoden über die Parameter `was` und `wann` verfügen und die Datentypen gleich oder zumindest implizit konvertierbar sind.

Letzteres kann einen vTable-fähigen Client dazu veranlassen, freiwillig auf die langsamere COM-Automation zurückzugreifen.

2.27.1.2 Mehrfachschnittstellen und Versionierung

Die Mehrfachschnittstellen in COM sind Grundlage der Versionierung (Versionierung bedeutet die parallele Bereitstellung verschiedener Versionen). Bei einer Änderung der Schnittstelle kann ein COM-Server neben der neuen auch die alte Schnittstelle unterstützen. Dadurch kann ein alter Client die Komponente weiterhin nutzen. Neuere Versionen einer Schnittstelle werden in COM oft durch Versionsnummern (z.B. `IMachWas2`) oder den Zusatz `Ex` für Extended (z.B. `IMachWasEx`) bezeichnet. Es ist dabei natürlich möglich, dass für Methoden, deren Implementierung unverändert geblieben ist, sowohl die Methode `IAltesInterface::MethodeX` als auch die Methode `INeuesInterface::MethodeX` auf die gleiche Implementierung verweist.

Mehrfach-
schnittstellen

Dieses Verfahren wird üblicherweise bei der Standardmethode `IUnknown::QueryInterface()` angewendet, die von jeder Schnittstelle implementiert werden und per definitionem die gleiche Aufgabe erfüllen muss. Es ist auch möglich, `IUnknown::QueryInterface()` so zu implementieren, dass die Anfrage nach einer veralteten Schnittstelle automatisch einen Zeiger auf die vTable einer neueren Schnittstelle liefert.

2.27.1.3 Vererbung

COM unterstützt keine Implementierungsvererbung, sondern nur die Schnittstellenvererbung. Die Wiederverwendung von Komponenten ist dennoch durch Aggregation und Delegation möglich. Da dies jedoch sehr weit in den Bereich des Komponentenentwurfs hineingeht, soll es an dieser Stelle bei dieser Erwähnung belassen werden. Mit der Vererbung in Visual Basic.NET (7.0) wird auch das Erben von COM-Klassen möglich. Diese Funktion wird allerdings nicht durch COM selbst, sondern durch den Visual Basic-Compiler bzw. die Visual Basic-Laufzeitumgebung bereitgestellt.

Vererbung

2.27.2 Bausteine von COM-Objektmodellen

Eine COM-Komponente kann beliebig viele Objektmodelle enthalten. Viele Komponenten bestehen jedoch aus nur einem einzigen großen Objektmodell. In COM bestehen Klassendefinitionen aus Attributen, Methoden und Ereignissen. Objektmodelle entstehen, indem Attribute auf andere Objekte verweisen oder Methoden Zeiger auf andere Objekte zurückliefern.

Klassentypen In COM-Objektmodellen gibt es üblicherweise vier Typen von Klassen, die aufgrund von zwei Kriterien gebildet werden:

- ▶ Einerseits unterscheidet man Klassen, die einen Behälter für eine Menge von Objekten darstellen (sogenannte Collections), und Klassen, die Einzelobjekte repräsentieren.
- ▶ Andererseits unterscheidet man, ob eine Klasse von außen durch den Komponentenbenutzer instanziiert ist oder nur intern von Instanzen anderer Klassen instanziiert werden kann.

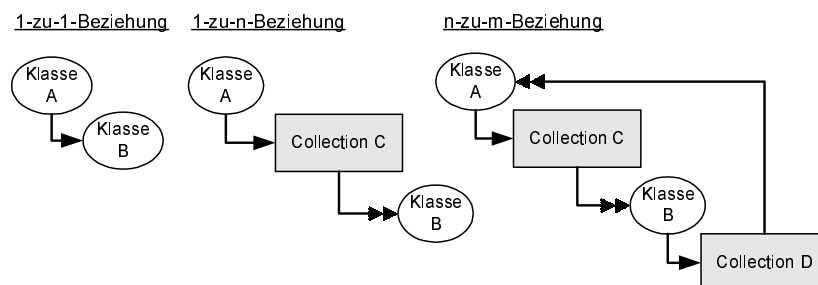
Collectionklassen versus Einzelklassen

Collections In COM wird die in Anhang A diskutierte Unterscheidung zwischen Set, Bag und Collection nicht vorgenommen. In der COM-Welt werden Objektmengen unabhängig davon, ob die enthaltenen Instanzen homogen oder heterogen sind, Collections genannt. Man sagt auch, eine Objekthierarchie bestehe aus Objekten und Collections (obwohl letztere natürlich auch Objekte sind). In der Regel ist die Anzahl der Unterobjekte einer Collection variabel. Häufig gibt es in COM-Komponenten auch Collections, die die Aufnahme von Unterobjekten auf Instanzen einer bestimmten Klasse beschränken (was in der Theorie ein *Set* genannt wird).

Collectionklassen versus Einzelklassen Collections in COM besitzen in der Regel eine Verwaltungsklasse. Diese Verwaltungsklassen werden in Abgrenzung zu Einzelklassen *Collectionklassen* genannt. Einzelklassen sind nicht in der Lage, eine Menge von Objekten zu verwalten. Der Begriff Collection wird insofern doppeldeutig verwendet, als damit einerseits nur die Verwalterklasse bezeichnet wird, andererseits auch alle in der Objektmenge enthaltenen Objekte.

Zirkuläre Referenzen In COM kommen n-zu-m-Beziehungen eher selten vor, da dies zirkuläre Referenzen bedingt. Zirkuläre Referenzen bringen Probleme bei der Freigabe von Speicherplatz mit sich.

Abbildung 2.28:
Modellierung der
Nutzungsbeziehungen
mit verschiedenen
Kardinalitäten
in COM



Es ist möglich, dass ein Objektmodell sich über Klassen aus mehreren Komponenten erstreckt. Dies kommt jedoch selten vor. Die *Microsoft Shell Objects* sind ein Beispiel dafür (siehe Kapitel 6).



Instanziiierbare versus nicht-instanziiierbare Klassen

COM-Komponenten verfügen in der Regel nur über wenige instanziiierbare Klassen. In einigen Fällen ist nur das Stammobjekt selbst instanziiierbar. Eine Komponente mit einer einzigen Stammklasse schließt die eigenständige Instanziiierbarkeit von untergeordneten Klassen nicht aus. In diesem Fall wird in dem Objektbaum nicht an der Wurzel begonnen, sondern an einem Ast, Zweig oder Blatt. Dann sollte jedoch eine Navigationsmöglichkeit zurück zu den in dem Objektmodell übergeordneten Objekten angeboten werden.

Instanziiierbarkeit

2.27.3 Best Practice für Objektmodelle

Microsoft hat mit den COM-Objektmodellen aus Collections und Objekten ein Rahmenwerk geschaffen, das eine große Bandbreite möglicher Realweltsituationen abbilden kann. Es gibt einige Empfehlungen, wie diese Objektmodelle verwendet werden sollten:

Gutes Objektmodelldesign

- ▶ Ein Objektmodell sollte genau eine Stammklasse besitzen, über deren Instanz der Zugriff auf die Instanzen der anderen Klassen möglich ist. Sofern es sich bei der COM-Komponente um eine eigenständige Anwendung handelt, sollte die Stammklasse *Application* heißen (vgl. [MIC98], S. 718).
- ▶ Der Weg von einem Objekt zu einem untergeordneten Objekt oder einer Collection sollte über ein Attribut erfolgen, das den gleichen Namen trägt wie der Klassenname des Objekts bzw. der Collection.
- ▶ Als Weg zurück von einem untergeordneten Objekt zu Objekten, die in der Objekthierarchie darüber liegen, werden folgende Möglichkeiten verwendet:
 - ▶ Über ein Attribut hält jede Klasse einen Verweis auf das Stammobjekt bereit. Dieses Attribut hat den Namen der Stammklasse (also meistens *Application*). Der Anwender kann dann vom Stamm aus zu allen Objekten navigieren.
 - ▶ Über ein Attribut *Parent* hält jede Klasse einen Verweis auf das im Baum nächsthöhere Objekt bereit. Dadurch ist eine schrittweise Navigation von den Blättern des Baums bis zurück zum Stamm möglich.
- ▶ Eine homogene Collection sollte den gleichen Namen haben wie die Klasse der Objekte, die sie enthält, jedoch im Plural (Beispiel: Eine Collection von *File*-Objekten heißt *Files*). Nur wenn Singular und Plural des Begriffs gleich sind (z.B. *Fenster*), sollte die Collection durch ein Suffix (z.B. *FensterListe*) kenntlich gemacht werden (vgl. [MIC98], S. 706f.).

Stammklasse

Attributnamen

Benennung von Collections

Collection-Methoden

- Eine Collection sollte die Methoden `Add()`, `Insert()` und `Remove()` sowie das Attribut `Item()` anbieten. Außerdem sollte sie die zur Unterstützung von `For...Each` benötigte Methode `_NewEnum` besitzen. `_NewEnum` soll eine versteckte Methode sein ([MIC98], S. 716ff.).
- In einer Collection sollte eine Methode, die ein extern erstelltes Objekt in eine Collection aufnimmt, `Insert()` heißen. Die Methode `Add()` sollte selbst eine Instanz der gewünschten Klasse erzeugen und zurückgeben.
- `Item()` sollte den Zugriff sowohl über einen fortlaufenden numerischen Index als auch über einen alphanumerischen Schlüssel unterstützen. Da `Item()` immer einen Parameter erwartet, müsste es sich eigentlich um eine Methode und nicht um ein Attribut handeln.

Attribut oder Methode?**Item()-Attribut oder Item()-Methode?**

Die Frage, ob `Item()` ein Attribut oder eine Methode ist, ist nicht trivial. Die Tatsache, dass `Item()` einen Parameter hat, spricht für eine Methode. Die Tatsache, dass in vielen Komponenten Zuweisungen an `Item()` erfolgen können (z.B. `objDic.Item(key) = Wert` in der Klasse `Scripting.Dictionary`), spricht dagegen für ein Attribut. Microsoft selbst ist nicht ganz eindeutig: Einerseits spricht man von `Item()` als Methode (z.B. [MIC98, S. 717], [MSD01i], [MSD01k]), andererseits von einem Attribut (z.B. [MSD01h], [MSD01g]). Dass Microsoft sich nicht einig ist, wird ganz deutlich in [MSD01j]: Das Dokument heißt `vbmthItemMethodActiveXControls.asp`, die Überschrift lautet aber **Item Property**.

Der Microsoft Objektkatalog zeigt, dass in einigen Collection-Klassen `Item()` als Attribut definiert ist (z.B. `Scripting.Drives`, `ADODB.Parameters` oder `MAPI.Folders`), während in anderen Collection-Klassen `Item()` eine Methode ist (z.B. `SQLDMO.Jobs`, `WbemScripting.SWbemObjectSet` oder `SHDocVw.ShellWindows`). Für COM selbst ist der Unterschied egal, da es in COM auf der Binärebene nur Methoden gibt.

2.27.4 Meta-Objektmodelle**Probleme mit Objektmodellen**

Die Definition eines Objektmodells kann durch zwei Umstände erschwert werden:

- **Komplexität:** Oft ist der durch Objektmodelle abgebildete Realweltausschnitt sehr komplex. Wenn es sehr viele Elementtypen in der Realwelt gibt, dann sind in der Regel auch viele Klassen nötig, um diese Welt zu modellieren.
- **Erweiterbarkeit:** Wenn es in der Realität vorkommt, dass Elementtypen sich in einem Zeitablauf in ihrem Aufbau ändern oder neue Elementtypen hinzukommen können, dann muss das Schema der Komponente (insbesondere das Objektmodell) angepasst werden. COM ist jedoch

nicht darauf ausgelegt, dass sich das Schema einer Komponente zur Laufzeit ändert. Jede Änderung bedeutet also eine Neukompilierung der Komponente.

Diesen Anforderungen wird inzwischen in einigen Fällen durch ein Konzept begegnet, das hier *Meta-Objektmodell* genannt werden soll. Ein Meta-Objektmodell ist eine Abstraktion von einem konkreten Objektmodell. Ein solches Meta-Objektmodell definiert einen allgemeinen Satz von (*Universal-*) *Klassen* (auch: *Meta-Klassen*), mit denen ein Zugriff auf ein anderes Objektmodell möglich ist. Dabei findet eine Abbildung statt: Eine Klasse des Meta-Objektmodells wird auf n Klassen des Basis-Objektmodells abgebildet. Diese Meta-Klasse bietet Funktionen, um auf die Schnittstellen der Basis-Klasse zuzugreifen.

Meta-Objektmodelle

Eine Sonderform bilden Meta-Objektmodelle, die eine 1-zu- n -Abbildung nur auf Ebene von Attributen und Methoden realisieren. Es gibt in diesen Objektmodellen zu jeder Basis-Klasse auch genau eine Klasse im Meta-Objektmodell. Allerdings kann es in der Basis-Klasse mehr Mitglieder geben als in der Meta-Klasse. Für die zusätzlichen Mitglieder bietet die Meta-Klasse ein Konstrukt an.

Wie auch schon bei dem Begriff Objektmodell ist an dieser Stelle der Hinweis notwendig, dass der Begriff Meta-Objektmodell auf Modellierungsebene anders verwendet wird. Dort ist ein Meta-Objektmodell eine weitere Abstraktionsebene über einer Beschreibung der Elemente einer objektorientierten Modellierung.

Andere Definition

Auch ein Meta-Objektmodell wird in einer COM-Komponente implementiert. Dabei unterscheidet sich eine solche Meta-Komponente aus der Sicht von COM in keinsten Weise von einer anderen Komponente. Der Unterschied liegt in der Semantik des implementierten Objektmodells. Ein Basis-Objektmodell kann auch wieder in Form einer COM-Komponente implementiert sein, muss es aber nicht. In der Praxis sind die Basis-Objektmodelle meist in anderer Form realisierte Objektmodelle.

Bindung in Meta-Objektmodellen

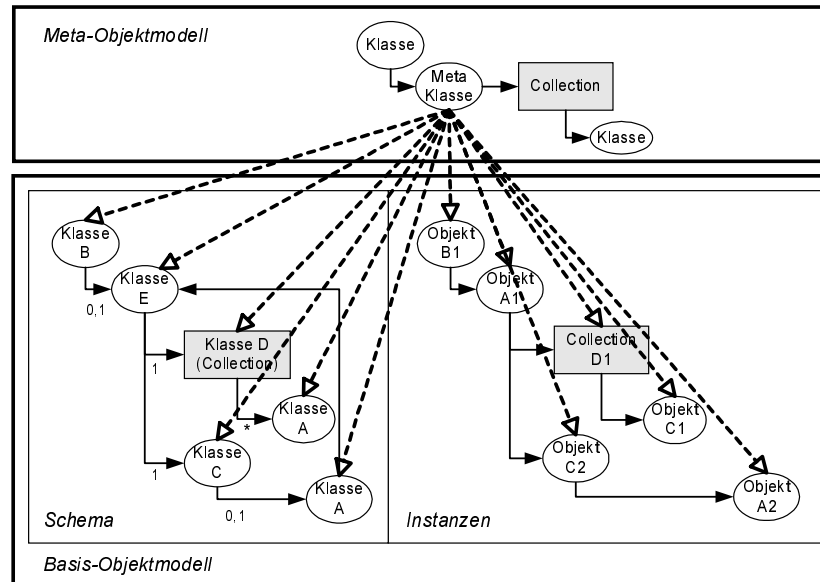
In der Regel ist vorgesehen, dass der Client nur mit dem Meta-Objektmodell, nicht auch mit dem konkreten Objektmodell arbeitet. Er instanziiert eine Klasse aus dem Meta-Objektmodell und *bindet* diese Instanz dann an ein Objekt aus dem Basis-Objektmodell, wobei die Bindung in der Regel an konkrete Instanzen dieses Modells erfolgt. Danach ermöglicht das Meta-Objekt den Zugriff auf die Attribute und Methoden des konkreten Objekts.

Bindung

Sofern das Basis-Objektmodell jedoch über ein transparentes Schema verfügt, ist es üblich, dass das Meta-Objektmodell nicht nur eine Abbildung der Instanzen des Basis-Objektmodells, sondern auch eine Abbildung der Schema-Informationen zulässt. Beide Möglichkeiten zeigt Abbildung 2.29.

In einem Meta-Objektmodell-Konzept ist eine Klasse des Meta-Objektmodells auch dafür zuständig, Instanzen im Basis-Objektmodell zu erzeugen bzw. zu vernichten.

Abbildung 2.29:
Eine Meta-Klasse in
einem Meta-Objekt-
modell bildet eine
Vielzahl von Klassen
und Instanzen in
einem konkreten
Objektmodell ab.



Meta-Objektmodelle in COM

Verfügbare Meta-Objektmodelle

Es gibt inzwischen im Bereich der COM-Komponenten für Windows einige Meta-Objektmodelle, z.B. das *Active Directory Service Interface (ADSI)* und das *Windows Management Instrumentarium Scripting API*. Die *ActiveX Data Objects (ADO)* sind ein Meta-Objektmodell auf Attributebene.

Unzureichende Dokumentation

Komponenten für Architekturen, die ein Provider-Konzept besitzen (z.B. ADSI, WMI, ADO, CDO/MAPI), sind geeignete Kandidaten für Meta-Objektmodelle, da die Programmierschnittstelle mit unterschiedlichen Providern zusammenarbeiten kann. Allerdings zeigen die Dokumentationen dieser Komponenten, dass deren Autoren sich über die Unterschiede zwischen den Meta-Objektmodellen und den Basis-Objektmodellen wenig Gedanken gemacht haben; in der Regel wird dieser Unterschied stillschweigend übergangen. Dabei ist eine saubere Trennung der beiden Konzepte für das Verständnis einer COM-Komponente sehr wichtig.

Hinsichtlich der Frage, wie die Abbildung eines konkreten Basis-Objekts auf Meta-Objekte erfolgt, kann man verschiedene Typen unterscheiden:

- Typ 1** ▶ **Typ 1:** Attribute und Methoden werden selbst wieder durch Meta-Objekte repräsentiert. Für die Attribute gibt es eine Collection von Attribut-Meta-Objekten, für die Methoden eine Collection von Methoden-Meta-Objekten. Die Attribut-Meta-Klasse stellt Methoden wie `Get()` und `Set()` bereit,

um die Eigenschaften des konkreten Objekts zu modifizieren. Die Methoden-Meta-Klasse stellt eine Methode wie `Execute()` bereit, um eine Methode auf dem konkreten Objekt aufzurufen. Diese Vorgehensweise führt zu einer sehr aufwendigen und widersinnig wirkenden Kodierung. Der Vorteil besteht darin, dass die Attribut- und Methoden-Meta-Objekte gleichzeitig Schemainformationen bereitstellen können. Ein Client kann so zur Laufzeit erforschen, welche Attribute und Methoden das konkrete Objekt besitzt.

- **Typ 2:** Eine Modifikation von Typ 1 besteht darin, Attribute und Klassen nicht durch eine eigene Meta-Klasse abzubilden, sondern die o. g. Zugriffsmethoden direkt in der Meta-Klasse abzubilden, die das konkrete Objekt als Ganzes repräsentiert. **Typ 2**

- **Typ 3: (Direct Access):** Die dritte Möglichkeit ist, dass sich das Meta-Objekt in Abhängigkeit von der aktuellen Bindung verwandelt, so dass der Nutzer keinen Unterschied zwischen dem Meta-Objekt und dem konkreten Objekt bemerkt. Das bedeutet, dass eine Meta-Klasse wie `CIMObject` nach der Bindung an ein Grundobjekt die Mitglieder des Grundobjekts so in seine `IDispatch`-Schnittstelle einbindet, dass Attribute und Methoden doch wieder über die übliche Mitglieder-Zugriffsnotation (z.B. `obj.Methode`) aufgerufen werden können. Eine solche Meta-Klasse ist also in der Lage, zur Laufzeit ihre `IDispatch`-Schnittstelle zu erweitern und die Grundklasse zu emulieren. Der Programmierer sieht keinen Unterschied mehr zwischen Meta- und Basis-Objekt und kann auf natürliche Weise programmieren. Dieses Verfahren hat im Gegensatz zu den anderen Typen einen Namen: *Direct Access*. **Typ 3**

Der Nachteil des Direct Access ist, dass der Nutzer nicht mehr unterscheiden kann, welche Mitglieder der Meta-Klasse und welche der konkreten (emulierten) Klasse angehören. Kritisch könnte es bei Namensüberschneidungen werden. Es empfiehlt sich daher, per Namenskonvention eine Überschneidung zu verhindern.

- **Typ 4:** Während alle oben beschriebenen Typen in der COM-Welt bereits vorkommen, ist dem Autor kein Meta-Objektmodell bekannt, das eine Emulation über eine Mehrfachschnittstelle realisiert. Ein Meta-Objekt würde neben der eigenen Standardschnittstelle, die unverändert bleibt, eine weitere Schnittstelle mit den Mitgliedern der Klasse des Basis-Objektmodells anbieten. Damit COM-Automation-Clients davon Gebrauch machen können, müsste die Standardschnittstelle ein Attribut anbieten, um zu der zweiten Schnittstelle zu navigieren. Wenn das konkrete Objekt selbst Mehrfachschnittstellen besitzt, könnte das Meta-Objekt auch diese abbilden: Für ein konkretes Objekt mit n Schnittstellen würde das Meta-Objekt dann $n+1$ Schnittstellen anbieten. Das Meta-Objekt ändert auch dabei seine Standardschnittstelle nicht, sondern fügt die Schnittstellen des konkreten Objekts hinzu. Damit COM-Automation-Clients davon Gebrauch machen können, müsste die Standardschnittstelle eine Methode `GetInterface()` anbieten, die die Navigation zu n Schnittstellen ermöglicht. **Typ 4**



Probleme bei den Eingabehilfen

Meta-Objektmodelle besitzen hinsichtlich der Unterstützung bei der Programmcodeeingabe den Nachteil, dass das Konzept der Typbibliotheken nur für die direkt angesprochene Komponente, also das Meta-Objektmodell, nicht aber für das Basis-Objektmodell funktioniert. Meta-Objektmodelle sind eine Form des späten Bindens; der bisherige Mechanismus zur Typermittlung auf Basis der Typdeklaration kann also nicht wirken, da die Typdeklaration stets für die Meta-Klasse erfolgt. Eine Unterstützung für die Funktionen der Grundklassen wäre zwar denkbar (z.B. durch den Zugriff auf das Schema des Basis-Objektmodells), ist aber heute noch an keiner Stelle implementiert.

Beispiele für Meta-Objektmodelle

Im Folgenden sollen drei Beispiele für die Realisierung von Meta-Objektmodellen vorgestellt werden. Die Realisierung ist dabei jeweils sehr unterschiedlich.

► Beispiel 1: WMI

WMI

Die Windows Management Instrumentation (WMI) abstrahiert am weitesten von den eigentlichen Klassen. Die WMI-COM-Komponente definiert nur Meta-Klassen wie `SWbemObject`, `SWbemMethod` und `SWbemProperty`, deren Instanzen an Basis-Objekte gebunden werden können. Attributzugriffe und Methodenaufrufe sehen in WMI dann folgendermaßen aus:

```
wert = Obj.Properties_("Attributname") und
Obj.Properties_("Attributname") = wert.

Set method = process.Methods_("Create")
Set inParam = method.inParameters.SpawnInstance_()
inParam.CommandLine = "calc.exe"
Set outParam = process.ExecMethod_("Create", inParam)
```

Unterscheidung
per Unterstrich

Da aber zusätzlich auch der direkte Zugriff angeboten wird, ist WMI ein Meta-Objektmodell sowohl von Typ 1 als auch von Typ 3. WMI verwendet zur Trennung der Mitglieder der Meta-Klasse von den emulierten Mitgliedern einen Unterstrich nach den Namen der Meta-Klassen-Mitglieder.

► Beispiel 2: ADSI

ADSI

Das Active Directory Service Interface (ADSI) definiert einen Satz von Standardschnittstellen. ADSI-Provider können Klassen aus diesen Standardschnittstellen zusammensetzen, aber auch neue Schnittstellen implementieren. Jede Klasse muss jedoch über eine bestimmte Standardschnittstelle verfügen, die Methoden zur Erweiterbarkeit bereitstellt. Im Fall von ADSI sind dies u. a. die Methoden `Put()` und `Get()`.

COM-Werkzeuge

```
obj.Put("Attributname") = Wert  
Wert = obj.Get("Attributname")
```

ADSI ist also ein Meta-Objektmodell vom Typ 2. Ein dynamischer Methodenaufruf ist nicht vorgesehen.

► Beispiel 3: DMO und ADO

Die *Distributed Management Objects* (DMO) und die *ActiveX Data Objects* (ADO) bieten einen direkten Zugriff auf die Grundobjekte und zusätzlich eine Erweiterbarkeit hinsichtlich der Attribute. Jede Klasse verfügt über eine *Properties-Collection* mit *Property*-Objekten, die jeweils ein Attribut repräsentieren. Ein *Property*-Objekt liefert Informationen über Name, Typ, Wert sowie Eigenschaften des Attributs und kann auch zur Veränderung des Werts verwendet werden. Somit lassen sich dynamische Attribute realisieren. Die Art des Zugriffs entspricht Typ 1.

DMO und ADO

Leider gibt es auch bei diesem Typ von dynamischen Objektmodellen keinen Standard. So verfügt ein *Property*-Objekt in ADO über die Attribute Name, Type, Value und Attributes. DMO verwendet dagegen Name, Type, Value, Get und Set.

2.28 COM-Werkzeuge

Die nachfolgend vorgestellten Werkzeuge sind eine nützliche Hilfe für Systemadministratoren und Programmierer. Sie vereinfachen die Suche und die Veränderungen von COM-Informationen in der Windows-Registry. Leider gibt es kein allumfassendes COM-Werkzeug. Bei vielen Aufgaben ist ein Zusammenspiel verschiedener Werkzeuge notwendig. Dabei sind die unterschiedlichen Sichten und Möglichkeiten, die die verschiedenen Werkzeuge bieten, für den Nicht-COM-Profi sehr verwirrend. Dies werden Sie insbesondere am Beispiel der AppID im *COM-Viewer* und beim *DCOM Configuration Utility* sehen.

Werkzeuge für COM

2.28.1 Microsoft Registry-Editoren

Das einfachste Werkzeug für die Konfiguration von COM sind Registry-Editoren. Microsoft stellt mit jedem 32-Bit-Betriebssystem zwei verschiedene Registry-Editoren zur Verfügung:

RegEdit und RegEdt32

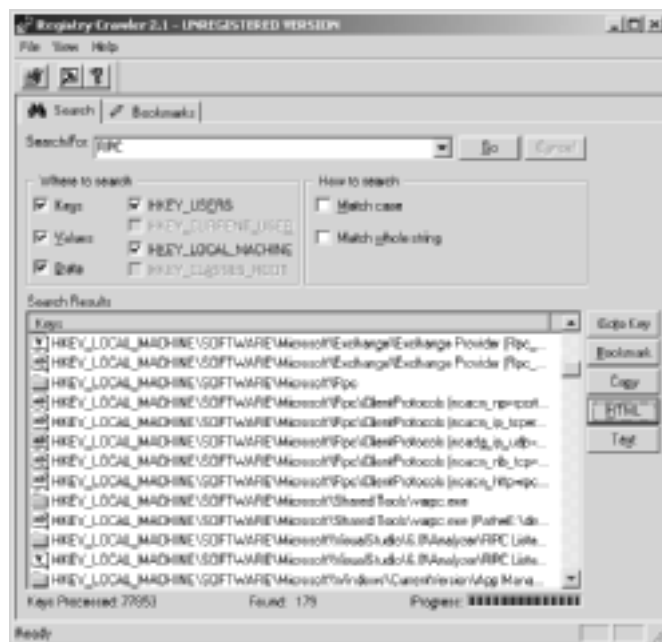
- RegEdit (REGEDIT.EXE)
- RegEdt32 (REGEDT32.EXE)



COM-Rechte und RegEdt32

Bewertung

Abbildung 2.30:
Suche nach dem
Begriff »RPC« im
Registry Crawler



2.28.2 Registry Crawler

Registry Crawler

132

nis zu Suchergebnis springen, sondern erhält eine übersichtliche Liste aller gefundenen Schlüssel und Einträge. Ein Doppelklick darauf öffnet RegEdit an der entsprechenden Stelle. Interessant ist auch die Möglichkeit, Bookmarks auf Registrierungsschlüssel zu setzen und so schnell zu interessanten Fundstellen zurückzufinden.

2.28.3 Regsvr32, SWBregsvr und CliReg

Das Kommandozeilentool *regsvr32* (REGSVR32.EXE) ermöglicht die Registrierung von COM-DLLs und Scriptlets. COM-EXE-Dateien registrieren sich beim Aufruf selbst bzw. über die Kommandozeilenparameter /REGSERVER und /UNREGSERVER. Da prozessinterne Komponenten nicht eigenständig ausgeführt werden können, ist ein Hilfsprogramm zur Registrierung notwendig. Dieses heißt REGSVR32.EXE und wird bei allen 32-Bit-Windows-Betriebssystemen im Systemverzeichnis installiert.

regsvr32

```
REGSVR32 [/u] [/s] [/i] [/n] dllname
```

Eine COM-DLL, die im Systemverzeichnis (SYSTEM bzw. SYSTEM32) liegt, wird einfach mit *regsvr32 dllname.dll* registriert. Sofern die DLL in einem anderen Verzeichnis liegt, muss der Pfad angegeben werden:

Registrierung und
Deregistrierung

```
REGSVR32 Laufwerk:\pfad\dllname.dll.
```

Dabei sollten keine UNC-Pfade, sondern lokale Pfade verwendet werden. Bei der Aufhebung einer Registrierung mit der Option /u werden alle zugehörigen Registry-Einträge wieder gelöscht.

```
REGSVR32.abc /u abc.dll
```

Normalerweise gibt *regsvr32* modale Dialogboxen zur Bestätigung aus. Dies kann mit der Option /s unterdrückt werden.

regsvr32 registriert Typbibliotheken automatisch, sofern sie in die Komponentendateien (.DLL, .EXE, .OCX) eingebettet sind.

Registrierung von Windows Script Components

Auch Windows Script Components können mit *regsvr32* registriert werden. Die Registrierung erfolgt jedoch nicht direkt, sondern über die SCROBJ.DLL.

WSC-
Registrierung

```
REGSVR32.EXE /n /i:scriptletname.wsc SCROBJ.DLL
```

Die Deregistrierung erfolgt analog mit der zusätzlichen Option /u.

```
REGSVR32.EXE /u /n /i:scriptletname.wsc SCROBJ.DLL
```

Mit der in Windows2000 enthaltenen Version 5.0.2134.1 von REGSVR32.EXE ist eine direkte Registrierung von .WSC-Dateien möglich:

```
REGSVR32.EXE scriptletname.wsc
```

Registrierung per Kontextmenü

Es wäre hilfreich, die Registrierung und Deregistrierung direkt über das Kontextmenü der Komponentendatei ausführen zu können. Leider bietet Windows im Standard derartige Kontextmenüeinträge nur für WSC-Dateien an, sofern die WSC-Runtime installiert ist. Für EXE- und DLL-Komponenten können Sie dies jedoch durch eine kleine Veränderung der Registry selber konfigurieren. Fügen Sie dazu die nachfolgende Registrierungsdatei in die Registry ein.

```
REGEDIT4
[HKEY_CLASSES_ROOT\.exe]
@="exefile"
[HKEY_CLASSES_ROOT\exefile\shell\Registrieren\command]
@="\"%1\" /regserver"
[HKEY_CLASSES_ROOT\exefile\shell\Registrierung aufheben\command]
@="\"%1\" /unregserver"
[HKEY_CLASSES_ROOT\dll]
@="dllfile"
[HKEY_CLASSES_ROOT\dllfile\shell\Registrieren\command]
@="regsvr32 \"%1\" "
[HKEY_CLASSES_ROOT\dllfile\shell\Registrierung aufheben\command]
@="regsvr32 /u \"%1\" "
```

Listing 2.10: Registrierungsdatei zur Erweiterung der Kontextmenüs bei .DLL- und .EXE-Dateien [CD: /install/tools/regsvr32/kontextmenüregistrierung.reg]

SWBregsvr**Registrierung per Drag&Drop**

Auf der Buch-CD [CD:/install/Werkzeuge/SWBregsvr/] finden Sie das Tool *SWBregsvr*, das ein kleines Anwendungsfenster bereitstellt, mit dem Sie per Drag&Drop Komponenten registrieren und deregistrieren können.

*Abbildung 2.31:
SWBregsvr*

**Registrierung von entfernten Visual Basic-Komponenten**

Clireg Komponenten, die nicht lokal, sondern ausschließlich remote ausgeführt werden, benötigen nur eine Minimalinstallation. Mit Visual Basic 6.0 Enterprise Edition erstellte Komponenten können mit dem Werkzeug *CLIREG.EXE* für den Remote-Zugriff konfiguriert werden. *CLIREG.EXE* erwartet als Steuerdatei ein Visual Basic Registration File (.VBR).

2.28.4 COM-Viewer (OLE Viewer)

Der *COM-Viewer* ist ein Werkzeug, das die Betrachtung und Bearbeitung der COM-Informationen in der Registry wesentlich vereinfacht. Der COM-Viewer nennt sich vollständig ausgeschrieben *OLE/COM Object Viewer*. An einigen Stellen heißt er auch nur *OLE Viewer*. Er ist Bestandteil der Resource Kits für NT 4.0 und Windows-2000 und Teil der Werkzeuge im Visual Studio-Produktpaket. Er ist auch auf der Buch-CD enthalten [CD:/install/Werkzeuge/COM Viewer] oder kann als Einzelanwendung von der Microsoft-Homepage geladen werden. Die aktuelle Versionsnummer ist 2.1.0.59 (Windows2000 Resource Kit). Mit Visual Studio 98 wurde Version 2.1.0.054 ausgeliefert. Es gibt keine lokalisierte Version mit deutschen Menüs. Beim Setup der deutschen Version von Visual Studio wird lediglich der Startmenü-Link auf den deutschen Namen OLE-ANSICHT umbenannt.

OLE/COM Object Viewer

In der Baumdarstellung stellt der COM-Viewer links COM-Bausteine dar, während auf der rechten Seite die zugehörigen Konfigurationsoptionen gezeigt werden. Der COM-Viewer kennt zwei Ansichtsarten: die Standardansicht und die Expertenansicht. In der Expertenansicht zeigt die Baumdarstellung in der linken Fensterhälfte die Oberpunkte OBJECT CLASSES, APPLICATION IDS, TYPE LIBRARIES und INTERFACES an. Der erste Punkt enthält die folgenden Unterpunkte:

Anzeigenmodi

- ▶ GROUPED BY COMPONENT CATEGORY: Hier sind die COM-Klassen nach COM-Komponentenkategorien gruppiert aufgelistet.
- ▶ OLE 1.0 OBJECTS sind alte Klassen aus OLE Version 1.0.
- ▶ COM LIBRARY OBJECTS sind Klassen, die in der COM-Bibliothek selbst definiert sind.
- ▶ ALL OBJECTS ist eine Liste aller registrierten Klassen.

In der Standardansicht werden in der Baumdarstellung nur COM-Komponentenkategorien und die für die jeweilige Kategorie registrierten Klassen angezeigt (also der Ast OBJECT CLASSES/GROUPED BY COMPONENT CATEGORY aus der Expertenansicht). Die Expertenansicht wird im Menü VIEW über den Menüpunkt EXPERT MODE aktiviert.

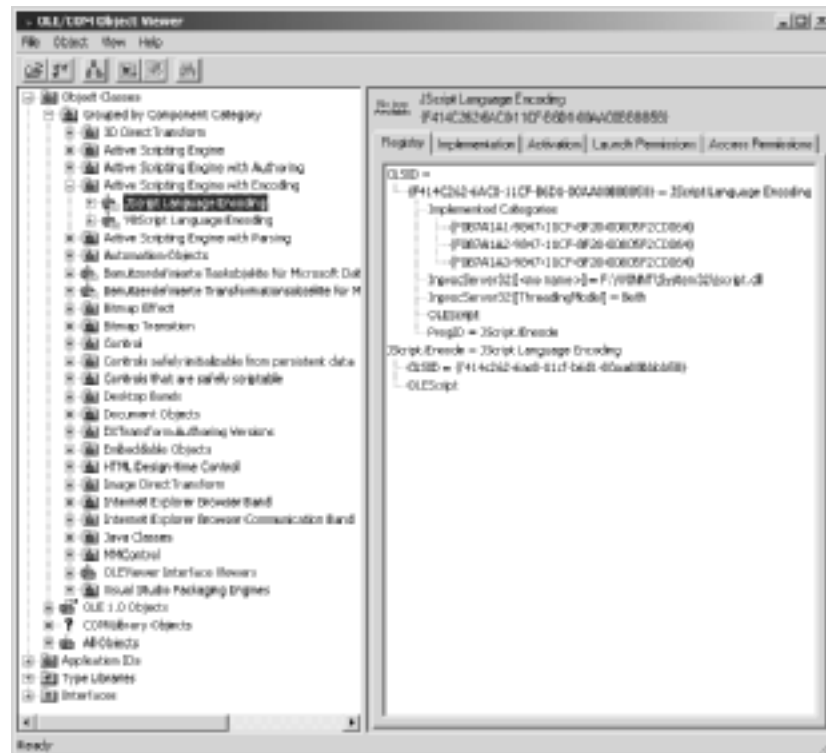
Standardansicht

Klassenansicht

Der COM-Viewer benutzt in der Auflistung der Klassen unterhalb von OBJECT CLASSES den in der Registry unterhalb des CLSID-Schlüssels als Standardattribut hinterlegten Klassennamen (*Friendly Class Name*), nicht die ProgID. Dies macht es nicht immer einfach, die gesuchte Klasse zu finden. Es ist besonders schwierig, Klassen einer bestimmten Komponente zu finden, da der Friendly Class Name im Gegensatz zur ProgID in der Regel nicht mit dem Komponentennamen beginnt. Leider bietet der COM-Viewer auch keine Suchfunktion. Im Zweifel müssen Sie also doch den Registry-Editor verwenden, um von der ProgID über die CLSID den Klassennamen zu ermitteln. Der COM-Explorer ist in diesem Punkt in der Darstellung besser konfigurierbar.

Klassennamen

Abbildung 2.32:
Expertenansicht im
COM-Viewer



Schnittstellen Bei einem Klick auf einen Klassennamen erzeugt der COM-Viewer eine Instanz der Klasse. Links in der Baumdarstellung werden die Schnittstellen der Klasse angezeigt und rechts die Konfigurationsoptionen der Klasse. Wie in Kapitel 2 dargestellt gibt die `IUnknown`-Schnittstelle nur Auskunft darüber, ob eine bestimmte Schnittstelle unterstützt wird, liefert aber keine Liste aller unterstützten Schnittstellen. Der COM-Viewer kann die Schnittstellen daher nur für die Klassen vollständig auflisten, für die es eine Typbibliothek gibt. Bei Klassen ohne Typbibliothek zeigt der COM-Viewer nur die vorhandenen COM-Standardschnittstellen an, die er offensichtlich durch Aufruf von `QueryInterface()` austestet. Wünschenswert wäre eine Funktion, die eine Klasse gegen eine registrierte Schnittstelle testet, um die Schnittstellen auch bei Klassen ohne Typbibliothek zu ermitteln.

Beziehungen Der große Vorteil gegenüber der Ansicht in der Registry ist, dass auch die Schlüssel, auf die die CLSID verweist (z.B. TypeLibID und ProgID), mit ihren Werten dargestellt werden. Im Registry-Editor müssten Sie diese Informationen mühsam herausuchen. Die weiteren Registerkarten bieten komfortable Eingabemasken für ausgewählte Einträge.

Bitte beachten Sie, dass jede Eingabe in eine der Eingabemasken sofort in die Registry geschrieben wird. Ein explizites Speichern oder ein Rückgängigmachen gibt es nicht.



Verwirrung mit den AppIDs

Bei den vom COM-Viewer angebotenen Eingabefeldern finden sich zu jeder Klasse unter den Registerkarten ACTIVATION, LAUNCH PERMISSIONS und START PERMISSION auch Einstellungen, die zu einer COM-Anwendung gehören und unterhalb einer AppID gespeichert werden. Der COM-Viewer ermöglicht also eine klassenweise Sicht auf eine COM-Anwendung, während das DCOM-Configuration Utility (DCOMCNFG, siehe Kapitel 2.28.9) eine anwendungsbezogene Sicht bietet.

**COM-Viewer
versus DCOM
CNFG**

Dies ist für COM-Einsteiger sehr verwirrend, denn der COM-Viewer suggeriert, dass Einstellungen wie die LAUNCH PERMISSIONS auf Klassenebene gesetzt werden können. In Wirklichkeit wirkt sich jede Änderung auf alle zu der gleichen COM-Anwendung gehörenden Klassen aus. Kritik ist auch in einem weiteren Punkt angebracht. Der COM-Viewer bietet nur die klassenbezogene Sicht, die anwendungsbezogene Sicht ist verkümmert: Unterhalb des APPLICATION IDS-Zweigs werden zwar die Registry-Schlüssel in Rohform angezeigt, es gibt jedoch enttäuschenderweise keinerlei Eingabemasken. Diese komfortablen Eingabemasken pro COM-Anwendung bietet nur das DCOM-Configuration Utility. Allerdings hat dieses Werkzeug den Nachteil, dass es nicht die Klassen anzeigen kann, die zu einer COM-Anwendung gehören.

Verwirrung

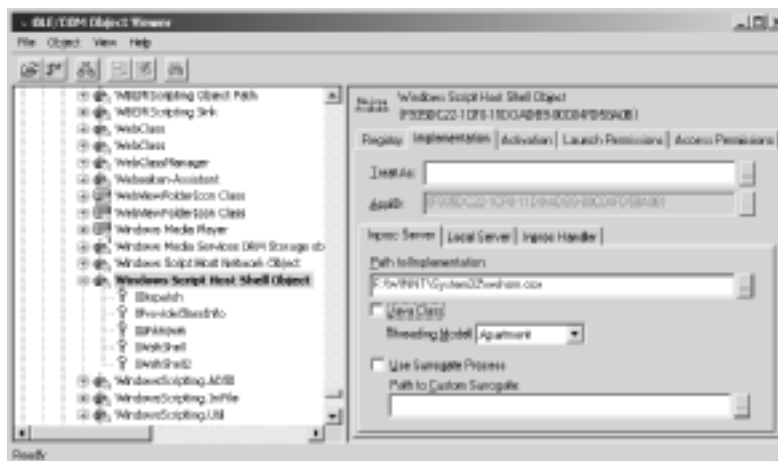


Abbildung 2.33:
Anzeige der Klasse
Windows Script Host
Network Object
(WScript.
Network)

Übrigens kann auch der COM-Viewer die Zugehörigkeit einer Klasse zu einer COM-Anwendung nur darstellen, nicht aber verändern. Wenn jedoch eine Klasse noch nicht zu einer COM-Anwendung gehört, dann wird bei der Eingabe eines Werts in eines der Eingabefelder, das sich auf eine COM-Anwendung bezieht, eine neue COM-Anwendung erzeugt. Leider sind

**Zuordnung von
Klassen zu Anwen-
dungen**

diese Felder weder grau unterlegt, wenn keine Anwendungszuordnung existiert, noch gibt es beim Anlegen einer neuen Anwendung eine Nachfrage beim Benutzer.

TypeInfo-Viewer

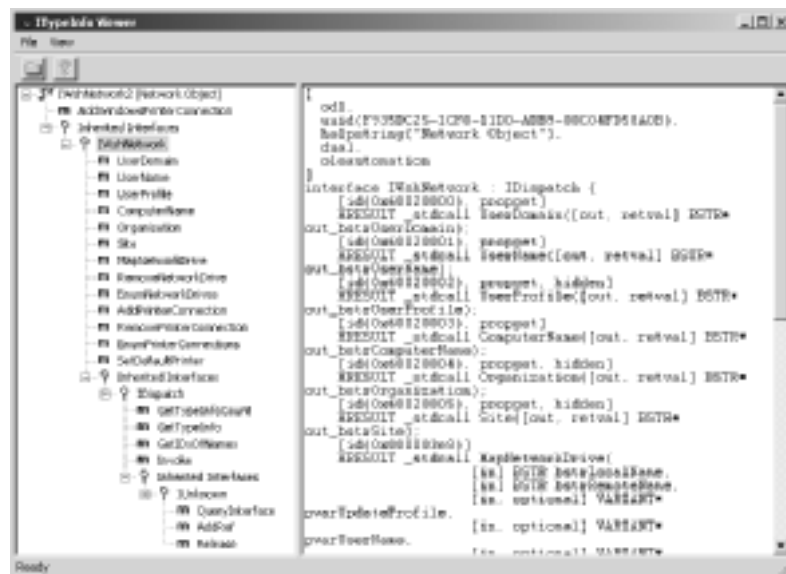
TypeInfo-Viewer

Durch einen Doppelklick auf ein Interface öffnet sich der *TypeInfo-Viewer*, der ein Teil des COM-Viewers ist. Im linken Teilfenster des TypeInfo-Viewers werden die enthaltenen Schnittstellen mit ihren Methoden angezeigt. Sofern die Schnittstellen von anderen Schnittstellen geerbt haben, werden die ererbenden Schnittstellen als untergeordnete Äste angezeigt. Auf der rechten Seite wird die zugehörige IDL-Definition angezeigt. Die Schnittstelleninformationen können als Textdatei mit der Extension .IDL abgespeichert oder per Cut&Paste übernommen werden. Sie können also mit dem TypeInfo-Viewer Typbibliotheken dekompileieren, um sie anschließend mit MIDL.EXE neu aufzubauen. Dies können Sie dazu nutzen, um eine Typbibliothek aus der DLL- oder EXE-Komponente zu extrahieren, wenn Sie die Schnittstellendefinitionen ohne die Implementierung weitergeben wollen.



Über das FILE-Menü des COM-Viewers können Typbibliotheksdateien auch direkt geöffnet werden. Typbibliotheken können in folgenden Dateitypen enthalten sein: .TLB, .DLL, .EXE, .OLB, .OCX, .RLL.

Abbildung 2.34:
Der TypenInfo-
Viewer zeigt die
Schnittstelle
IWSHNetwork2,
die von IWSHNetwork
abgeleitet ist.
IWSHNetwork
implementiert
IDispatch, die
pflichtgemäß von
IUnknown
abgeleitet ist.



Schwäche des COM-Viewers

Klassen werden übersehen

Die wichtigste Komponentenkategorie für den Skriptprogrammierer stellen die Automation Objects dar. Leider hat der COM-Viewer ausgerechnet hier ein Problem: Er zeigt nicht alle automationsfähigen Klassen an. Natürlich kann der COM-Viewer höchstens die automationsfähigen Klassen anzeigen,

für die es in der Registry eine CLSID gibt. Aber auch aus dieser Menge über-
sieht der COM-Viewer einige, z.B. die Klasse `ADODB.Recordset`. Der im
nächsten Kapitel vorgestellte COM-Explorer findet mehr, wenngleich eben-
falls nicht alle relevanten Klassen.

2.28.5 COM-Explorer

Der *COM-Explorer* ist ein Shareware-Werkzeug der Firma 4Developers
[DEV00]. Er zeigt die auf einem System registrierten automationsfähigen
Klassen, getrennt nach Steuerelementen, COM-DLLs und COM-EXEs. Zu
jeder Klasse werden die Informationen über die entsprechende Komponente
angezeigt. Dabei werden nicht nur die Informationen aus der Registry, son-
dern auch die aus der Komponentendatei selbst ausgelesen. In der Kompo-
nentendatei sind Beschreibungen sowie Informationen über die Version und
den Hersteller abgelegt. Die Anzeige kann auch so umkonfiguriert werden,
dass in der Listendarstellung die CLSID, der Typbibliotheksname oder die
Dateiposition angezeigt wird. Alle Informationen können in Form einer
CSV- oder HTML-Datei exportiert werden (Menü `TOOLS/GENERATE
REPORT`). Die Registerkarte `REGISTRY ENTRIES` zeigt die für die Komponente
relevanten Ausschnitte aus der Registry, allerdings ohne – anders als der
COM-Viewer – Eingabehilfen anzubieten.

**Konfigurierbare
Darstellung**

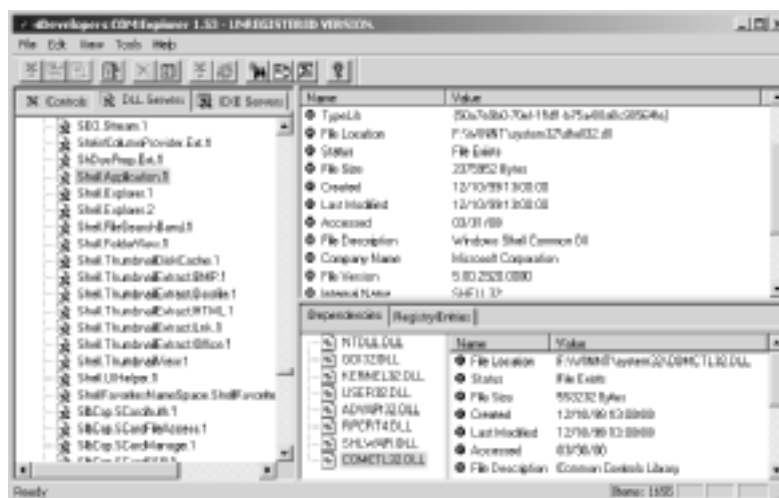


Abbildung 2.35:
Der COM-Explorer
zeigt die Klasse
`Shell.Application`.

Der Schwerpunkt des Werkzeugs liegt darauf, die Funktionsfähigkeit der
Komponente auf der Grundlage ihrer Abhängigkeit zu überprüfen. Zu jeder
Komponente werden die DLLs angezeigt, von denen die Komponente
abhängig ist. Über eine Suchfunktion (Menü `TOOLS/FIND MISSING DEPE
NCENCIES`) können Komponenten mit Abhängigkeitsproblemen gezielt
gesucht werden. Der COM-Explorer ergänzt das Kontextmenü von DLLs
um die Punkte `REGISTER`, `UNREGISTER`, `COMPARE TO` und `VIEW DLL DETAILS`.

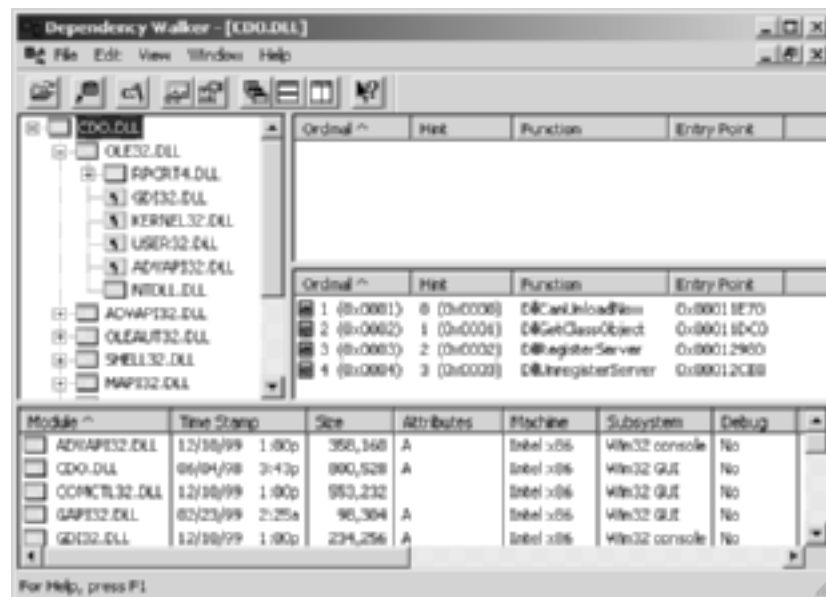
Abhängigkeiten

2.28.6 Dependency Walker

Dependency Walker

Der *Dependency Walker* zeigt die Abhängigkeiten von DLLs und EXEs von anderen DLLs sowie die von DLLs exportierten bzw. importierten Funktionen an. Sie benötigen den Dependency Walker, um zu analysieren, welche DLLs eine Komponente braucht. Dies ist hilfreich, wenn die Registrierung fehlschlägt. Der Dependency Walker wird mit Visual Studio und den Resource Kits für NT 4.0 und Windows2000 ausgeliefert. Der COM-Explorer zeigt zwar auch die Abhängigkeiten, aber nur für solche Komponenten, die bereits erfolgreich registriert wurden. Für Komponenten, die Sie aufgrund fehlerhafter Abhängigkeitsbeziehungen nicht registrieren können, benötigen Sie den Dependency Walker.

Abbildung 2.36:
Die CDO-Komponente im Dependency Walker



2.28.7 Objektkatalog

Objektkatalog

Der *Microsoft Objektkatalog* (engl. *Object Browser*) ist ein Werkzeug zur Darstellung von bzw. zur Suche in Typbibliotheken und vereinfacht die Darstellung sehr stark. Er unterscheidet nicht zwischen Klassen und Schnittstellen und stellt Schnittstellen als Klassen dar. Zudem werden Standardschnittstellen von Klassen verborgen. Insgesamt entsteht so eine Sicht auf die Komponenten, aus der der Entwickler nicht mehr erkennen kann, welche Schnittstellen in welchen Klassen bzw. in deren Instanzen vorhanden sind, obwohl genau dies eine wertvolle Information der Typbibliotheken ist. Eigentlich sollte der Objektkatalog besser Klassenbrowser heißen, da er die Objekte nicht in ihren Beziehungen zur Laufzeit zeigt. Dies leistet das Lokalfenster der Visual Basic-Entwicklungsumgebung.

COM-Werkzeuge

Der Objektkatalog wird mit den Entwicklungsumgebungen von Visual Basic, Visual Basic for Applications und Visual InterDev ausgeliefert. Er bildet ein Fenster innerhalb dieser IDEs, das leider nicht separat gestartet werden kann.

Keine Einzelanwendung

Im Objektkatalog werden nicht automatisch alle auf dem System verfügbaren Typbibliotheken angezeigt. Damit eine Komponente hier betrachtet werden kann, müssen Sie erst im VERWEISE-Dialog (PROJEKT/VERWEISE in VB6 bzw. EXTRAS/VERWEISE in VBA/Office 2000) die Typbibliothek aktivieren. Die VERWEISE-Dialoge berücksichtigen jedoch keine ActiveX-Steuerelemente. Diese fügen Sie durch Aufnahme in die Werkzeugsammlung einem Projekt hinzu (Eintrag KOMPONENTEN im Kontextmenü der Werkzeugsammlung in VB6; in VBA ist es an gleicher Stelle der Eintrag ZUSÄTZLICHE STEUERELEMENTE).

Verweise

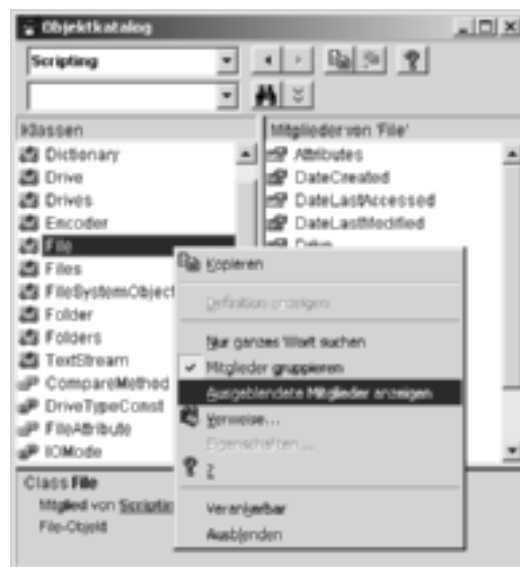


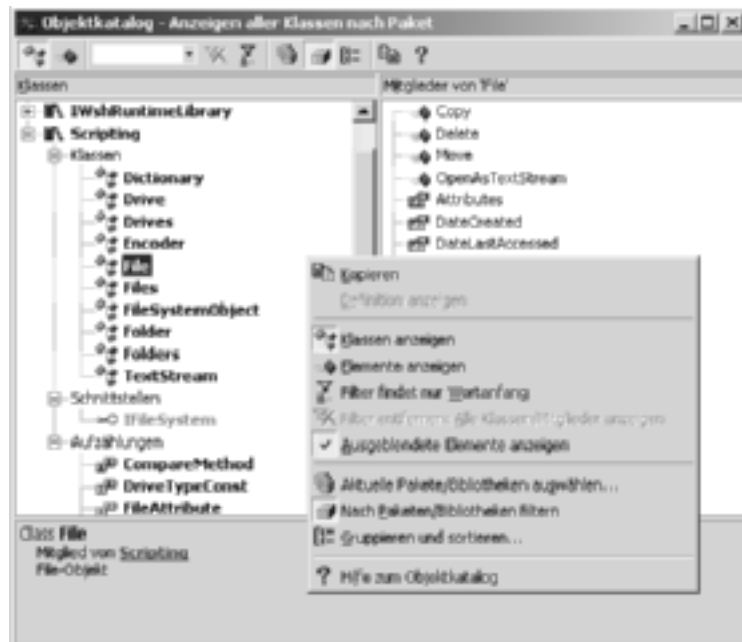
Abbildung 2.37:
Der Objektkatalog
in Visual Basic 6.0
zeigt die Klasse File
aus der Scripting
Runtime-Kompo-
nente.

Objektkatalog in Visual InterDev 6.0

Der in Visual InterDev 6.0 enthaltene Objektkatalog unterscheidet sich etwas von seinen Kollegen: Anhand der konfigurierbaren Darstellung ist erkennbar, ob ein Eintrag eine Klasse oder eine Schnittstelle ist. Einiges hat sich allerdings gegenüber den anderen Versionen des Objektkatalogs nicht verbessert: Der Zusammenhang von Klassen und Schnittstellen wird auch hier nicht dargestellt, und Standardschnittstellen werden unterdrückt.

**InterDev-Objekt-
katalog**

Abbildung 2.38:
Objektkatalog in
Visual InterDev 6.0
(wieder die Klasse
Scripting.File)

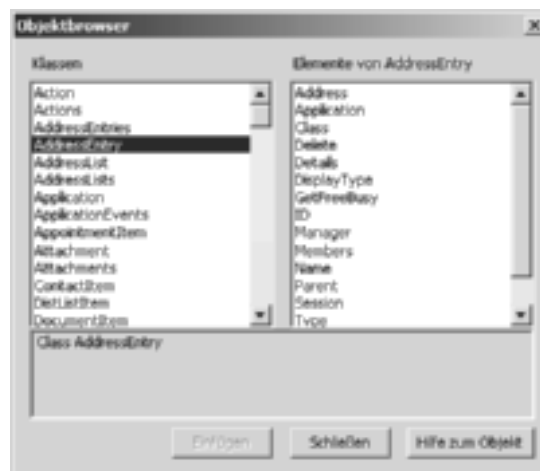


Objektkatalog in Outlook 2000

Outlook-Objektkatalog

Eine dritte Variante des Objektkatalogs findet man innerhalb des Skript-Editors von Outlook 2000. Dieser Objektkatalog (hier *Objektbrowser* genannt) kann jedoch nur die Objekte aus der Outlook-Komponente darstellen.

Abbildung 2.39:
Outlook 2000-
Objektkatalog



Objektkatalog in Visual Studio.NET

Der Objektkatalog in Visual Studio.NET basiert optisch auf dem Objektkatalog von Visual InterDev 6.0. Der Objektkatalog kann nicht nur .NET-Komponenten, sondern auch COM-Komponenten anzeigen.

VS.NET

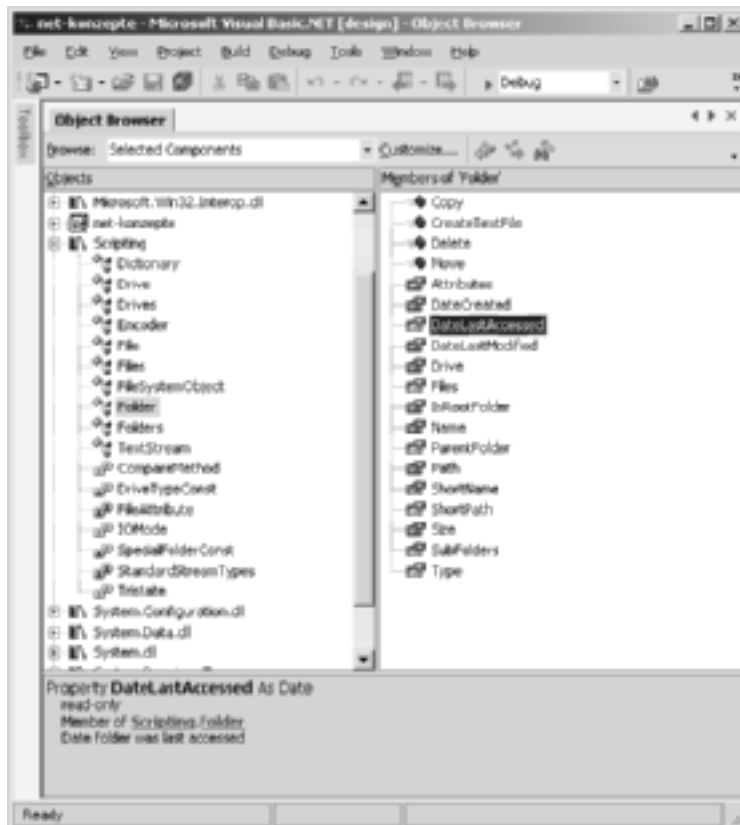


Abbildung 2.40:
Der Objektkatalog zeigt eine COM-Komponente. Die anderen Komponenten in der Baumstruktur sind .NET-Komponenten.

2.28.8 comTLBrowser

Der *comTLBrowser* ist als Nebenprodukt während der Arbeit an diesem Buch entstanden. Er stellt ebenso wie der Objektkatalog eine Typbibliothek dar, jedoch unterscheidet er sich in einigen Funktionen stark von den Microsoft Objektkatalogen:

comTLBrowser

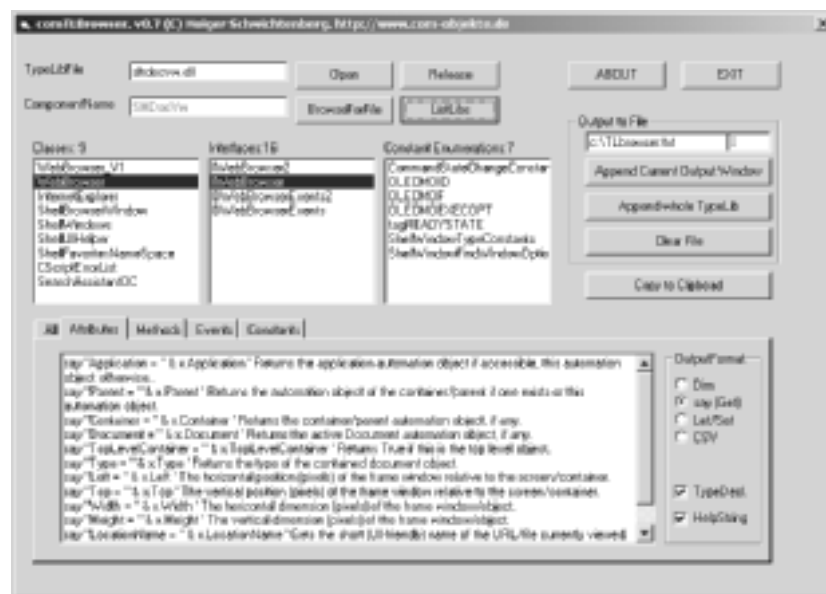
- ▶ Der *comTLBrowser* ist eine eigenständige Anwendung (COMTLBROWSER.EXE).
- ▶ Es wird klar zwischen Klassen und Schnittstellen getrennt, und es ist ersichtlich, welche Schnittstellen eine bestimmte Klasse unterstützt.
- ▶ Alle Informationen werden in Form von Listen dargestellt, die per Cut&Paste in andere Umgebungen übernommen werden können.

- Der Umfang der Darstellung ist wählbar.
- Aus Attributdefinitionen kann in Visual Basic-Code für Wertzuweisungen bzw. Lesezugriffe auf die Werte generiert werden. Diesen Code können Sie per Cut&Paste übernehmen. Das erspart viel Tipparbeit bei der Erforschung neuer Klassen.
- Konstantendefinitionen können als Const-Deklarationen oder als Array-Deklarationen ausgegeben werden. Die Const-Deklarationen sind in Umgebungen nützlich, die nicht auf die Typbibliothek zugreifen können. Im *Microsoft Objektkatalog* können Sie jede Definition einer symbolischen Konstante nur einzeln per Cut&Paste übernehmen. Im comTLBrowser können Sie ganze Listen auf einmal übernehmen.
- Die Deklarationen können alternativ auch in einer Textdatei abgespeichert werden.



Den comTLBrowser finden Sie auf der Buch-CD im Verzeichnis /INSTALL/ WERKZEUGE/COMTLBROWSER.

Abbildung 2.41:
Anwendungsfenster
des comTLBrow-
sers. Dargestellt
werden hier die
Attribute der
Schnittstelle
IWebBrowser, die
von der Klasse
WebBrowser in
SHDOCVW.DLL
implementiert wird.



2.28.9 DCOM-Konfigurationswerkzeug

DCOMCNFG

Das *DCOM-Configuration Utility* (kurz: *DCOMCNFG*) dient der Konfiguration von globalen DCOM-Einstellungen sowie von COM-Anwendungen. Der COM-Viewer listet zwar die installierten COM-Anwendungen auf, bietet jedoch Konfigurationsmöglichkeiten für AppID-Werte nur aus der Sicht einer Klasse an. DCOMCNFG zeigt leider nicht an, welche Klassen zu einer COM-Anwendung gehören und ermöglicht auch nicht die Anlage neuer COM-Anwendungen.

COM-Werkzeuge

Das DCOM-Configuration Utility gehört zum Installationsumfang von DCOM. Es wird in das Systemverzeichnis kopiert, ohne jedoch eine Verknüpfung im Startmenü herzustellen. Das Werkzeug hat den Dateinamen DCOMCNFG.EXE. Mit Windows2000 wurde die Version 5.0.1447.1 ausgeliefert. Unter NT4-Systemen mit älteren Service-Packs bzw. Windows 9x/ME-Versionen mit älteren Internet Explorer-Versionen ist noch Version 4.x installiert, die Sie unbedingt auf die aktuelle Version bringen sollten, da Ihnen sonst einige wichtige Funktionen nicht zur Verfügung stehen.

In Whistler (Windows XP/Windows.NET) ist das DCOM-Konfigurations-tool Teil des MMC-Snap-In »Komponentendienste«. Der Aufruf von DCOMCNFG.EXE startet dieses Snap-In.

Installation und Versionen

Whistler

DCOMCNFG kann unter Windows 95/98/ME nur ausgeführt werden, wenn sich das Betriebssystem im Sicherheitsmodus ZUGRIFFSKONTROLLE AUF BENUTZEREbene befindet, da die COM-Sicherheit nur in diesem Modus verfügbar ist.

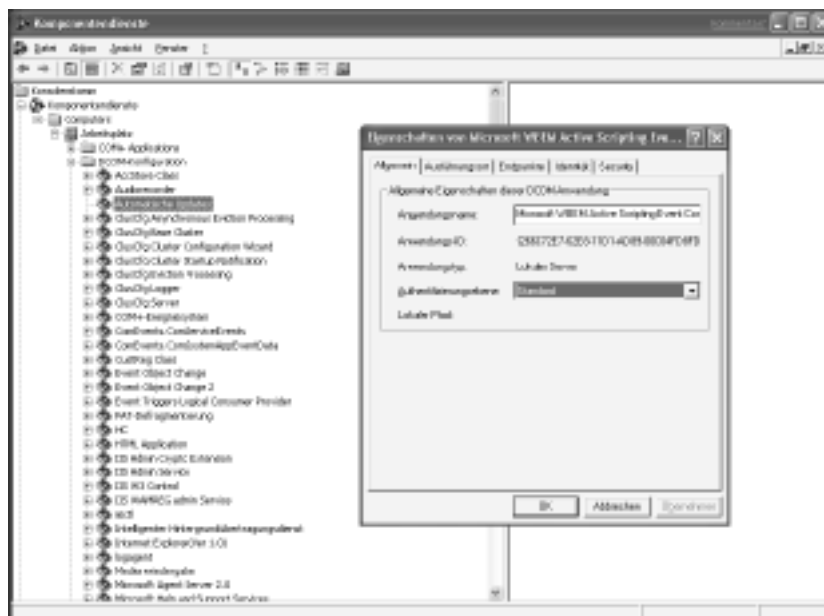


Abbildung 2.42:
DCOMcnfg als Teil
des MMC-Snap-In
»Komponenten-
dienste« in Win-
dows Whistler

Hauptansicht

Die Hauptansicht von DCOMCNFG bietet einerseits eine Liste der installierten COM-Anwendungen und andererseits den Zugriff auf die globalen DCOM-Einstellungen (HKEY_LOCAL_MACHINE\SOFTWARE\MICROSOFT\OLE) über weitere Registerkarten (vgl. nächste Abbildung).

- Standardwerte**
- ▶ Unter den STANDARDEIGENSCHAFTEN können Sie DCOM und CIS grundsätzlich aktivieren bzw. deaktivieren sowie die Standardeinstellung für die Authentifizierung und die Impersonifizierung festlegen.
 - ▶ STANDARDSICHERHEIT ermöglicht die Konfiguration der Standardzugriffsrechte, der Standardstartberechtigungen und der Standardkonfigurationsberechtigungen, die für alle Komponenten gelten, für die es keine speziellen Sicherheitseinstellungen über eine COM-Anwendung gibt.
 - ▶ STANDARDPROTOKOLLE legt die DCOM zur Verfügung stehenden Transportprotokolle und deren Priorität fest.



In Windows2000 sind diese Standardeinstellungen auch über das MMC-Snap-in »Komponentendienste« möglich.

Abbildung 2.43:
Hauptansicht des
DCOM-Configura-
tion Utility



Anwendungskonfiguration

Der Button EIGENSCHAFTEN führt zu den AppID-Einstellungen der jeweiligen COM-Anwendung.

- Allgemein**
- ▶ Die Registerkarte ALLGEMEIN zeigt den Pfad zu der zugehörigen Implementierung. Einzige Einstellmöglichkeit ist die Authentifizierungsstufe.
- Standort**
- ▶ Auf der Registerkarte STANDORT kann der Computer spezifiziert werden, auf dem die zugehörigen COM-Klassen instanziiert werden sollen. Diese Einstellungen beeinflussen die Schlüssel APPID\REMOTESEVERNAME und APPID\ACTIVATEATSTORAGE.

- ▶ Auf der Registerkarte SICHERHEIT können Start-, Zugriffs- und Konfigurationsberechtigungen gesetzt werden. Dabei besteht jeweils die Möglichkeit, die im Hauptmenü spezifizierten Standardeinstellungen zu übernehmen oder aber für die ausgewählte COM-Anwendung eigene Rechte zu vergeben. Vergeben werden können Start-, Zugriffs- und Konfigurationsberechtigungen.
- ▶ Auf der Registerkarte IDENTITÄT kann die Impersonifizierungsart der einzelnen Komponente eingestellt werden, wodurch der Schlüssel APPID\RUNAS beeinflusst wird. Die Auswahl »Benutzer, der die Anwendung startet« führt zu einem leeren Eintrag; »Interaktiver Benutzer« zum Eintrag »RunAs=Interactive User«.
- ▶ In der Registerkarte ENDPUNKTE können individuelle Protokollpräferenzen mit den zugehörigen Endpunkten (Portnummer bzw. Pfad) für jede COM-Anwendung festgelegt werden.

Sicherheit

Identität

Endpunkte



Abbildung 2.44: Anwendungskonfiguration der COM-Anwendung »Microsoft WBEM Server«, die den WMI-Dienst repräsentiert

Bei COM-DLLs stellt DCOMCNFG lediglich die Registerkarten ALLGEMEIN, STANDORT und ENDPUNKTE zur Verfügung.



2.28.10 MTS Explorer

Die Administration des Microsoft Transaction Server (MTS) erfolgt unter NT 4.0 über ein Snap-In für die Microsoft Management Console (MMC) mit dem Namen Transaction Server Explorer.

MTS-Administration

Den MTS Explorer gibt es auch als Stand-alone-Anwendung (MTXEXP.EXE) für Windows 95/98/ME – allerdings mit anderer Oberfläche und eingeschränkten Möglichkeiten. Unter Windows2000 und den Folgeversionen übernimmt die Aufgabe des MTS Explorers das MMC-Snap-In »Komponentendienste«.

Paketverwaltung

Computer und Packages	Der MTS Explorer stellt auf der obersten Ebene Computer dar. Ein Computer enthält Packages; ein Package ist unterhalb von Computern angeordnet und besteht aus Klassen. Jede Klasse implementiert Schnittstellen, die wiederum aus Methoden bestehen. Der MTS Explorer bietet auch statistische Informationen über die Anzahl der aktivierten Objekte sowie über den Zustand der Transaktionen an. Eigenschaftsfenster und Aktionen stehen jeweils im Kontextmenü des Eintrags zur Verfügung. Die wichtigste Eigenschaft ist dabei der Transaktionsmodus, der auf Klassenebene konfiguriert wird.
Pakete schnüren	Auf der Ebene des Packages können neue Klassen hinzugefügt werden. Dabei hat der Administrator die Möglichkeit, aus den bereits auf dem Rechner installierten Klassen auszuwählen oder aber eine neue In-process-COM-DLL zu installieren. Im letzteren Fall werden alle Klassen der Komponente in das Package eingefügt. Die nachträgliche Entfernung einzelner Klassen ist aber möglich.
Exportieren von Packages	Der Kontextmenüeintrag Export bei einem Package verbindet zwei interessante Funktionen miteinander: Zum einen speichert Export eine Package-definition in Form einer .PAK-Datei zusammen mit allen zugehörigen Komponenten-DLLs in ein Verzeichnis ab und ermöglicht so die einfache Weitergabe von Packages an andere Server. Zum anderen legt Export auch eine Setup-Routine für den Client an, die alle benötigten Dateien und Konfigurationen enthält, damit ein Client auf das MTS-Package bzw. die COM+-Anwendungen zugreifen kann. Die auf einem Client notwendigen Installationen für den Zugriff auf die entfernten Komponenten heißen application proxy.



Das Installationspaket enthält den Namen des Servers, auf dem die Packages erzeugt wurden. Diese Standardeinstellung lässt sich jedoch im Snap-In verändern (Eigenschaften des Computereintrags).

Rollenbasierte Sicherheit

Rollen	MTS erweitert die Konfigurationsmöglichkeiten von COM um ein Rollenkonzept und ermöglicht eine feiner granulierte Sicherheitseinstellung, als dies mit dem DCOM Configuration Utility (DCOMCNFG) möglich ist. Innerhalb eines Packages sind Rollen definierbar. Jeder Rolle kann der Administrator NT-Benutzer und NT-Benutzergruppen zuordnen. Sofern dann für das Package die Sicherheit in den Package-Eigenschaften aktiviert wird, können nur noch NT-Benutzer die Klassen eines Packages nutzen, die entweder direkt oder über ihre NT-Gruppenzugehörigkeit einer der definierten Rollen angehören. Für diese Funktionalität allein wäre es nicht not-

wendig gewesen, mehrere Rollen zu definieren. Rollen können dafür verwendet werden, die Zugriffsmöglichkeiten auch auf einzelne Schnittstellen einer Klasse festzulegen.

Rollenbasierte Sicherheit wird unter Windows 95/98/ME nicht unterstützt. Alle Zugriffe sind immer erlaubt.



2.28.11 Snap-In »Komponentendienste«

In Windows 2000 und den Folgeversionen Windows Whistler (Windows XP/Windows.NET) wird ein Snap-In für die Microsoft Management Console (MMC) mitgeliefert, mit dem man COM-DLLs zu einer COM+-Anwendung zusammenfassen kann. Das Snap-In erfüllt die Funktion des MTS Explorers, den es für Windows NT 4.0 und Windows 95/98/ME gibt. Darüber hinaus gibt es noch COM+-spezifische Funktionen.

Die Aufnahme einer COM-DLL in eine COM+-Anwendung ermöglicht den Fernaufruf einer COM-DLL, da dann für diese DLL ein Surrogat-Prozess bereitgestellt wird.

COM+-Anwendungen konfigurieren

In der folgenden Abbildung sehen Sie die COM+-Anwendung »Scripting-Buch«, die mehrere Klassen enthält. Die Funktionen zum Hinzufügen von Bausteinen werden wie üblich über die Kontextmenüs bereitgestellt. Die Anzeige der Schnittstellen und deren Mitglieder unterhalb jeder Klasse wird über die Typbibliothek realisiert.

COM+-Anwendungen konfigurieren

DCOM-Konfiguration

Unter Windows Whistler enthält das Snap-In zusätzlich den Ast »DCOM-Konfiguration«. Dahinter verbergen sich die Funktionen, die in früheren Windows-Versionen (einschließlich Windows 2000) durch das DCOM-Konfigurationswerkzeug DCOMCNFG.EXE bereitgestellt wurden.

Ersatz für dcomcnfg.exe

2.28.12 ROT-Viewer

Der ROT-Viewer (IROTVIEW.EXE) ist ein kleines Tool, um die aktuellen Inhalte der Running Objects Table (ROT) anzuzeigen. Der ROT-Viewer wird mit Visual Studio ausgeliefert.

Inhalt der ROT

COM-Theorie

Abbildung 2.45:
Konfiguration
einer Klasse
in einer
COM+-
Anwendung

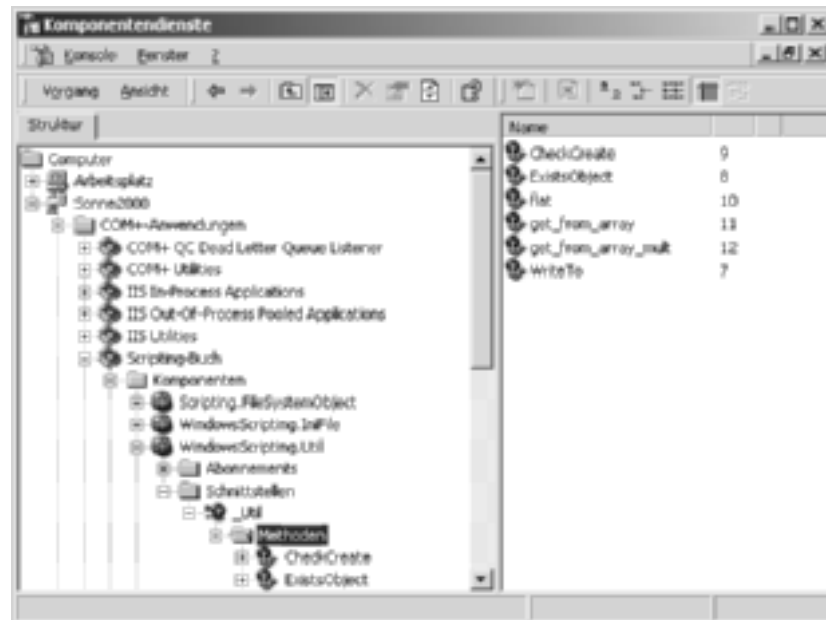
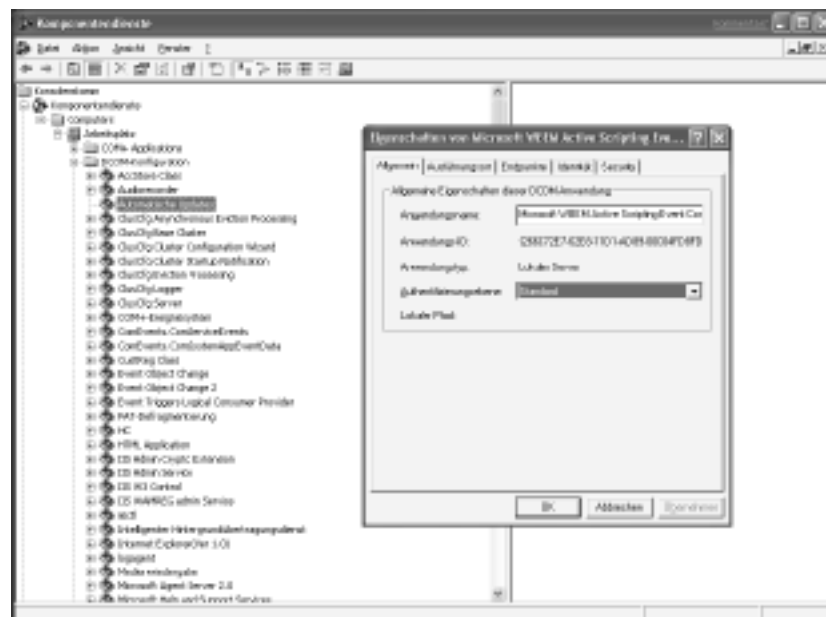


Abbildung 2.46:
DCOM-Konfigu-
ration unter
Windows Whistler



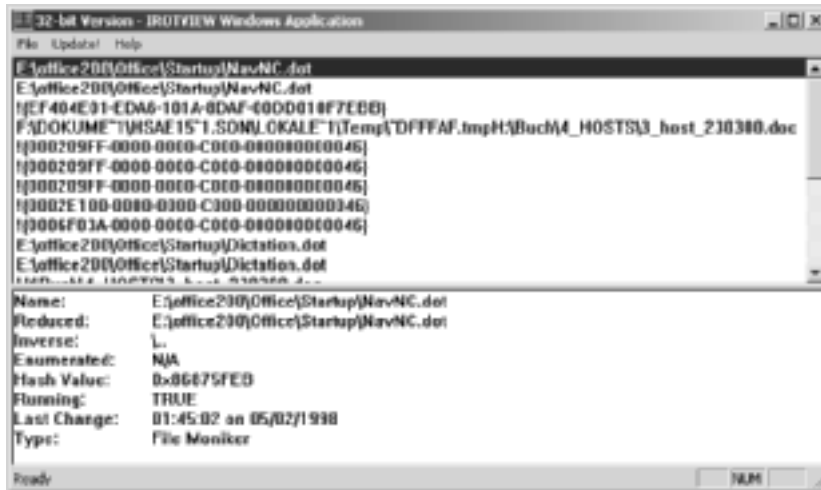


Abbildung 2.47:
ROT-Viewer

2.29 Bewertung von COM

Die COM-Welt könnte so schön sein, wenn es nicht einige Unzulänglichkeiten gäbe. Zunächst werden in diesem Unterkapitel Unzulänglichkeiten der COM-Spezifikation bzw. der COM-Implementierung angesprochen. Anhand einiger real existierender Objektmodelle sollen dann Beispiele für schlechte Objektmodelle und Typbibliotheken aufgezeigt werden. Dies ist jedoch nur eine beispielhafte Auflistung, um Sie grundsätzlich für Problembereiche zu sensibilisieren. Weitere Informationen erhalten Sie im Rahmen der detaillierten Beschreibung der Komponenten in den folgenden Kapiteln.

COM-Kritik

2.29.1 Vorteile von COM

Zunächst zu den unbestreitbaren Vorteilen von COM:

- ▶ COM ist ein Binärstandard für Komponenten und weitestgehend sprachunabhängig.
- ▶ COM ist in der Windows-Welt sehr weit verbreitet.
- ▶ COM unterstützt Mehrfachschnittstellen und bietet damit ein Instrument zur Versionierung (Versionierung bedeutet die parallele Bereitstellung verschiedener Versionen) von Komponenten.
- ▶ COM ist integriert in die Windows-Sicherheit.
- ▶ COM bietet mit dem Dienst Distributed COM (DCOM) eine transportprotokollneutrale Middleware zur entfernten Nutzung von Komponenten.
- ▶ COM ist ein objektorientierter Ansatz, der verschiedene Konzepte der Objektorientierung unterstützt.

2.29.2 COM auf anderen Plattformen

COMSource Auf anderen Plattformen hat COM bislang kaum Bedeutung. Neben *COM-Source*, der COM-Referenzportierung der Open Group für Solaris und True64UNIX [COS00], gibt es jedoch inzwischen auch Ansätze von Softwareherstellern, COM auf anderen Plattformen verfügbar zu machen (z.B. *EntireX* von der Software AG).

Active Group Die COM-Spezifikation hat Microsoft im Jahre 1996 offiziell in die Verantwortung der Active Group, einem Zusammenschluss im Rahmen der Open Group, übergeben. Allerdings wird die Active Group ihrem Namen nicht gerecht und ist äußerst passiv – wie ein Blick auf die seit Jahren nicht mehr gewartete Homepage schnell enthüllt [AGR00]. COM ist faktisch weiterhin eine proprietäre Architektur, deren Weiterentwicklung allein Microsoft bestimmt. Diese Bewertung wird von Gruhn/Thiel [GRU00, Seite 260] und Chappell [CHA97, Seite 56] unterstützt.

Immerhin gibt es auf den Webseiten der Open Group inzwischen eine COM-Referenz unter dem Namen »ActiveX Core Technology Reference« [ATX00]. Auch hier wird also ActiveX nicht klar von COM abgegrenzt.

2.29.3 Unzulänglichkeiten von COM

Dieses Unterkapitel listet zunächst einige grundsätzliche Unzulänglichkeiten von COM auf. Danach folgt eine genauere Kritik an den Objektmodellen und Typbibliotheken vieler Komponenten.

Die allgemeinen Unzulänglichkeiten von COM sind:

- IDispatch** ► Nicht alle COM-Sprachen können alle COM-Komponenten nutzen, da es zwei unterschiedliche Mechanismen zum Methodenaufruf (*IUnknown* und *IDispatch*) gibt. Leider unterstützen nicht alle COM-Komponenten beide Verfahren.
- Namen** ► Nicht alle Klassen verfügen über eine ProgID oder einen Friendly Class Name, die dem Komponentennutzer die Bedeutung der Klasse offen legen.
- DLL-Hölle** ► COM ermöglicht die Versionierung innerhalb einer Komponente durch Mehrfachschnittstellen. COM ermöglicht – zumindest vor Windows Whister – aber nicht die parallele Installation einer Komponente in verschiedenen (Unter-)Versionen und die eindeutige Zuordnung dieser Komponenten zu COM-Clients. So kommt es zur sogenannten »DLL-Hölle« (engl.: DLL Hell), wenn verschiedene COM-Clients jeweils eine andere Version einer Komponente für den korrekten Betrieb benötigen. Dieses Problem ist aber in Windows Whistler (Windows XP/Windows .NET) durch die sogenannten *Side by Side Assemblies* gelöst.
- Registry** ► Die Einstellungsmöglichkeiten in der Registry sind zu unstrukturiert bzw. zu flexibel. Beispielsweise kann ein Entwickler gegen die Konventionen zum Aufbau einer ProgID verstoßen und den Anwender durch

Bewertung von COM

Inkonsistenzen zwischen ProgID, Typbibliotheksname, Typbibliothek-Helpstring und Friendly Name einer Klasse vollkommen verwirren.

► COM fehlt es an einem Komponentenrepository, das Meta-Informationen über Komponenten in strukturierter Form speichert. Die COM-Informationen in der Registry werden den Anforderungen an Meta-Informationen über Komponenten nicht gerecht.	Komponenten-repository
► Es fehlt ein Object Trader, der auf Anfrage nach bestimmten Fähigkeiten eine entsprechende Klasse ermittelt. Die Möglichkeit der Zuordnung zu Komponentenkategorien ist unzureichend.	Object Trader
► Eine einheitliche Laufzeitumgebung für COM-Komponenten ist bislang nicht vorhanden. Es ist definiert, wie Komponenten auf binärer Ebene genutzt werden sollen; aus der Sicht des Programmierers ergeben sich jedoch je nach Sprache, in der der COM-Client implementiert werden soll, große Unterschiede in der Art des Umgangs mit den Komponenten.	Laufzeit-umgebung
► COM bietet keine Mechanismen zur Implementationsvererbung. Diese kann bislang allenfalls durch eine sprachspezifische Laufzeitumgebung (z.B. Visual Basic 7.0) geschaffen werden.	Vererbung
► Nicht alle COM-Komponenten sind dokumentiert. Ohne Dokumentation bleibt nur die Selbsterforschung auf Basis von Typbibliotheken.	Dokumentation
► Nicht alle COM-Komponenten verfügen über eine Typbibliothek. Ohne eine solche ist die Erforschung einer (undokumentierten) Komponente allerdings nicht möglich. Ebenso stehen die Unterstützungsfunktionen zur Codeeingabe ohne Typbibliothek nicht zur Verfügung.	Typbibliothek
► Die Objektmodelle in den COM-Komponenten sind oft so uneinheitlich, dass keine intuitive Nutzung möglich ist.	Objektmodelle

2.29.3.1 Schlechte Objektmodelle

Der letzte der o.g. Punkte wiegt besonders schwer aus der Sicht von Programmierern und Administratoren, die sich ständig in neue Komponenten einarbeiten sollen. Außer den wenigen Empfehlungen von Microsoft gibt es keine Richtlinien für die Umsetzung von Objektmodellen. Leider verstößt Microsoft selbst häufig gegen die eigenen Empfehlungen, wie folgende Beispiele für von Microsoft herausgegebene Komponenten zeigen:

- Einige Collections (z.B. in der *WSH Runtime Library*) enthalten keine Objekte, sondern eine Menge elementarer Datentypen. Die Effizienz mag höher sein, wenn sowieso nur ein Attribut pro Objekt zu verwalten ist. Die Konsistenz leidet allerdings darunter.

Im Fall der Klasse *WSHNetwork* gibt es zwei Collections, bei denen jeweils ein Unterobjekt mit zwei Attributen notwendig gewesen wäre. Statt einer korrekten Implementierung mit Unterobjekten hat sich Microsoft jedoch unverständlicherweise für eine Liste von String-Werten entschieden, in der die beiden notwendigen Attribute abwechselnd enthalten sind.

Kritik an bestehenden Objektmodellen

- ▶ Nicht immer heißt ein Attribut, das auf ein Unterobjekt verweist, wie die Klasse des Unterobjekts. Einen identischen Namen zu wählen ist zwar grundsätzlich nur dann möglich, wenn maximal ein Objekt dieser Klasse existiert; dennoch hält sich Microsoft auch dann nicht daran, wenn es möglich wäre.
- ▶ Bei ADSI spricht Microsoft nicht von Collections, sondern verwendet die aus der Welt der Verzeichnisdienste stammende Bezeichnung Container. Die Methode zur Erzeugung neuer Instanzen in einem Container heißt `Create()` und nicht `Add()`. Das Löschen erfolgt über `Delete()` statt über `Remove()`.
- ▶ Die Klassen `SecurityCallContext` und `SecurityIdentity` in der *COM+ Services Library* wurden als Collections implementiert, obwohl es nur eine feste Anzahl von Attributen bzw. Unterobjekten gibt. Dies führt dazu, dass Attribute nicht mehr über die übliche Punktnotation, sondern völlig untypisch über den Zugriff auf die `Item()`-Methode angesprochen werden können.
- ▶ In den *COM+ Administration Objects* sind alle Collections beim ersten Zugriff zunächst einmal leer. Erst der Aufruf der Methode `Populate()` ermöglicht den Zugriff auf die enthaltenen Objekte.
- ▶ Und noch einmal die *COM+ Administration Objects*: Der Zugriff auf Collections erfolgt nicht über Attribute des übergeordneten Objekts, sondern über die Methode `GetCollection()` auf der Ebene der Collection, die über dem übergeordneten Objekt liegt. `GetCollection()` benötigt dazu einen eindeutigen Bezeichner für das übergeordnete Objekt und den Namen der diesem Objekt untergeordneten Collection, auf die zugegriffen werden soll.
- ▶ Die unterschiedliche Art und Weise, Meta-Objektmodelle zu implementieren, wurde schon im vorherigen Kapitel ausführlich besprochen.
- ▶ Leider kann man bei den Microsoft-Objektmodellen oft vom Namen eines Mitglieds nicht darauf schließen, ob es sich um ein Attribut oder eine Methode handelt: So ist `GetFolder()` in der Klasse `Shell32.FolderItem` ein Attribut; die Verweise auf untergeordnete Collections sind dagegen in der gleichen Komponente als Attribute deklariert.
- ▶ Die vorhandenen Komponenten-Dokumentationen in der MSDN Library sind nicht einheitlich und oft zu wenig strukturiert. Die Art der Auflistung der Klassen, ihrer Schnittstellen, Methoden, Attribute und Ereignisse ist von Komponente zu Komponente grundverschieden. Zudem gibt es selten eine grafische Darstellung der Objektmodelle in der Dokumentation. Das erschwert die Einarbeitung in neue Komponenten und stellt den Microsoft-Kunden vor die Frage, wie es sein kann, dass ein solches Unternehmen keine Standards für die Dokumentation von Software hat.

Zukunft

Leider werden diese schlechten Objektmodelle die Windows-Programmierer auf sehr lange Zeit verfolgen: aus Kompatibilitätsgründen erfolgt ein grundsätzlicher Umbau von Objektmodellen nur selten. Umso wichtiger wäre es, Objektmodelle von Beginn an nach den festgesetzten Prinzipien zu erstellen. Man merkt, dass Microsoft zwar das Prinzip der Arbeitsteilung bei der Entwicklung der Komponenten beherrscht, jedoch nicht in der Lage ist, zwischen den Entwicklungsteams eine gute Kommunikation herzustellen.

Ein wichtiger Tipp: Halten Sie sich bei Ihren selbst erstellten Objektmodellen an die Modellierungsempfehlungen!

Zukunft



2.29.3.2 Schlechte Typbibliotheken

Kritikpunkte finden sich auch bei den COM-Typbibliotheken:

- ▶ Die Typbibliotheken geben manchmal nur einen Teil der vorhandenen Schnittstellen wieder.
- ▶ Anstatt Klassen und ihre Schnittstellen zu definieren, beschränken sich viele Typbibliotheken auf die Schnittstellen. Viele Typbibliotheken enthalten Klassenbeschreibungen nur für die instanziiierbaren Klassen. Der Objektkatalog zeigt in diesem Fall die nichtzugeordneten Schnittstellen mit Namen der Form `IName` als Klassen an (z.B. `MSXML.DLL`, vgl. Kapitel 8.2).
- ▶ Einige Klassennamen in den Typbibliotheken verhalten sich zu den Registry-Einträgen inkonsistent (z.B. bei der *WSH Runtime Library*, *MSHTML* und der *Taskscheduler-Komponente*).
- ▶ In einigen Typbibliotheken werden die Namenskonventionen verletzt. Zum Beispiel heißen Klassen in der *WSH Runtime Library* `IWSHShell_Class` und `IWSHNetwork_Class`. Ein anderes Beispiel sind die Schnittstellennamen in den *Collaboration Data Objects (CDO)*, die nicht mit einem großen `I` beginnen. In der *Taskscheduler-Komponente* heißen einige Schnittstellen `IDisp[Name]`. Das »disp« soll dabei auf die Existenz einer `IDispatch`-Schnittstelle verweisen. Das ist insofern zu kritisieren, als dies entweder durchgängig bei allen Komponenten erfolgen sollte oder gar nicht.

Kritik an
bestehenden
Typbibliotheken

Um auch ein Beispiel zu nennen, in dem es richtig gemacht wurde: `SCRRUN.DLL` (*Scripting Runtime Library*). Hier besitzt eine Klasse wie `File` eine Schnittstelle `IFile`.

Positivbeispiel

2.30 .NET (»DOTNET«)

Das Microsoft .NET-Framework, das im Jahr 2000 angekündigt wurde und Ende 2001/Anfang 2002 in der endgültigen Fassung erscheinen soll, ist die Weiterentwicklung von COM und dennoch ein völlig neues Komponentenmodell. Eine vollständige Darstellung von .NET würde mehrere Bücher füllen. Dieses Kapitel gibt nur einen Ausblick auf die wesentlichen Verbesserungen in .NET.

Die neue .NET-Komponentenarchitektur verbessert zehn wesentliche Unzulänglichkeiten von COM:

1. Komplexität

Komplexität

Das neue Komponentenmodell ist einfacher und universeller. Es gibt nicht mehr verschiedene Zugänge zu einem Objekt 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, statt 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.

2. Selbstbeschreibung

Metadaten

Im .NET-Framework ist es Pflicht, dass alle Komponenten eine Selbstbeschreibung enthalten. Diese Metadaten liefern Informationen über die enthaltenen Klassen mit ihren Schnittstellen und über deren Attribute, Methoden und Ereignissen. Die Sprachcompiler haben dafür zu sorgen, dass diese Metadaten erzeugt werden und konsistent zum wirklichen Inhalt der Komponenten sind.

3. Einheitliche Laufzeitumgebung

Common Language Runtime

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 neunziger Jahre bereits für das mit Windows2000 ausgelieferte COM+ angekündigt war. COM+ enthielt jedoch diese Laufzeitumgebung noch nicht.

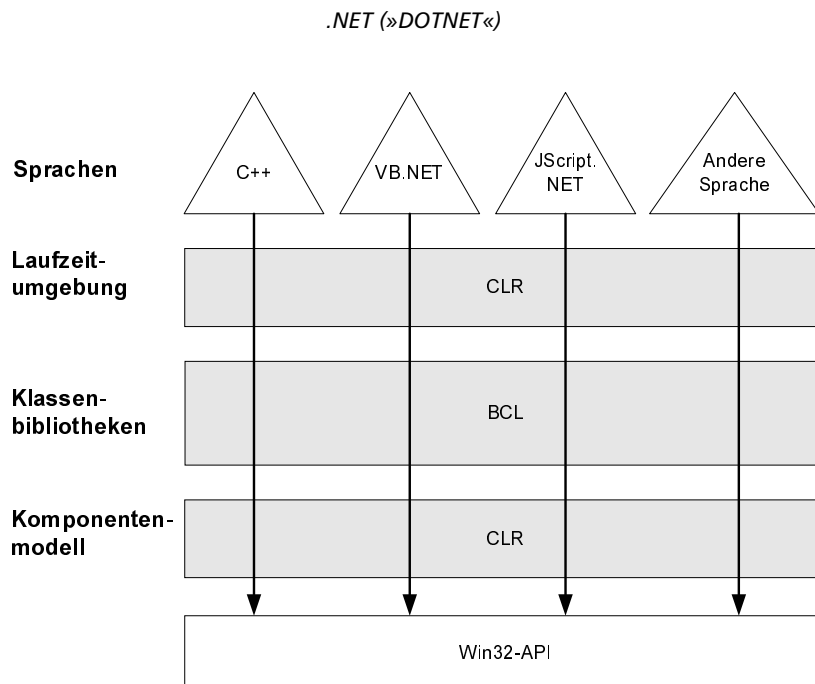


Abbildung 2.48:
Einheitliche Laufzeitumgebung und einheitliche Klassenbibliothek in .NET

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 Festplatte des Systems zu kopieren (Schlagwort »XCOPY-Deployment«). Lediglich globale Komponenten erfordern eine spezielle Behandlung.

Keine Registrierung

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 Protocol (SOAP).

Remoting

7. Vererbung

Das .NET-Komponentenmodell unterstützt im Gegensatz zu COM die Vererbung – auch zwischen verschiedenen Sprachen.

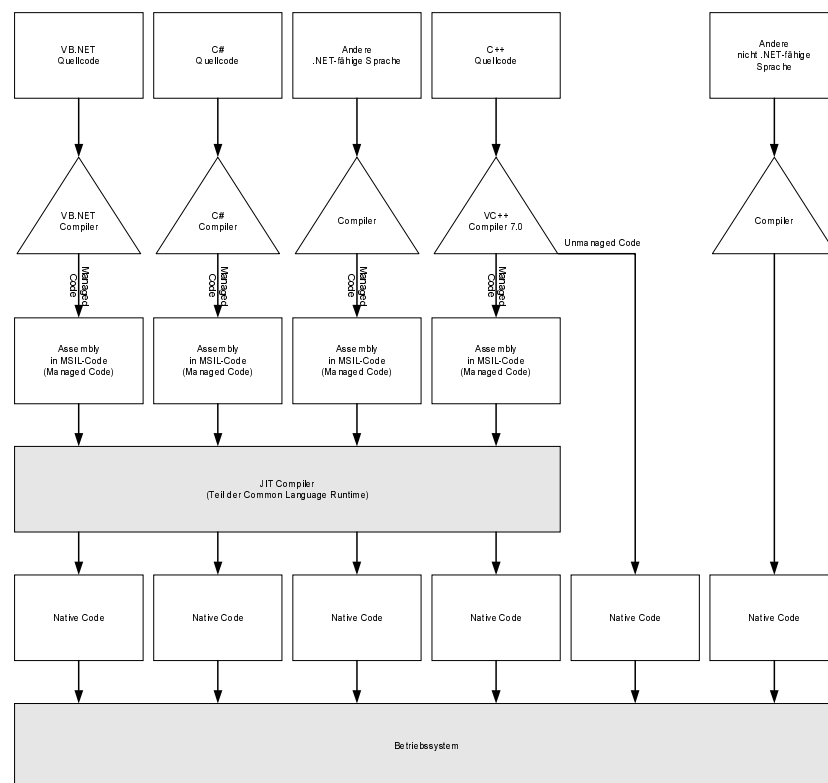
Vererbung

8. Plattformunabhängigkeit

Plattform-unabhängigkeit

.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 sogenannte *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).

Abbildung 2.49:
.NET-Intermediation



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)

Im .NET-Framework kann sehr genau gesteuert werden, auf welche Systemressourcen eine .NET-Anwendung zugreifen kann. Dabei 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.

Sicherheit

3 COM-Praxis mit Visual Basic und VBScript

Der COM-Theorie im letzten Kapitel folgt nun die praktische Nutzung von COM-Komponenten in Visual Basic, Visual Basic for Applications (VBA) und Visual Basic Script. Sie werden dabei aber nicht nur erfahren, wie Sie Komponenten nutzen können, sondern auch wie Sie eigene Komponenten mit Visual Basic selbst erzeugen können.

3.1 COM in den verschiedenen Visual Basic-Dialekten

Die Sprache Visual Basic ist dafür bekannt, den Umgang mit COM besonders einfach zu machen. Die VB-Laufzeitumgebung kapselt viel von der Komplexität von COM, insbesondere kapselt sie die Arbeit mit den Standardschnittstellen `IUnknown` und `IDispatch`. Die Vereinfachung, die die Visual Basic-Laufzeitumgebung bietet, geht – wie so oft – andererseits auf Kosten der Leistungsfähigkeit.

**VB-Laufzeit-
umgebung**

Komponenten nutzen

Grundsätzlich verfügen alle Visual Basic-Dialekte über die Möglichkeit, auf dem System vorhandene COM-Komponenten zu nutzen, also als COM-Client zu agieren. Es gibt jedoch einige Einschränkungen:

- ▶ Visual Basic Script (VBScript) kann – wie andere Skriptsprachen auch – nur auf COM-automationsfähige Komponenten via `IDispatch`-Schnittstelle zugreifen.
- ▶ Visual Basic 6.0 und Visual Basic for Applications (VBA) können zwar nicht nur auf `IDispatch`, sondern auch auf `IUnknown` zugreifen, allerdings benötigen diese Sprachen eine Typbibliothek mit den Schnittstelleninformationen. Eine COM-Komponente, zu der es keine Typbibliothek gibt, kann also in VB6 und VBA nicht genutzt werden.

VBS

VB6 und VBA

Die Fähigkeit, als Event Publisher oder Event Subscriber zu agieren, ist in den verschiedenen Visual Basic-Dialekten unterschiedlich stark ausgeprägt:

Ereignisse

- ▶ Sowohl VB6 und VBA als auch VBS ermöglichen es Klassen, auf die vordefinierten Klassenereignisse `Class_Initialize()` und `Class_Terminate()` zu reagieren.
- ▶ VB6/VBA-Klassen können Event Publisher sein, die eigene Ereignisse definieren und an ihre Umwelt aussenden. VBS-Klassen können dagegen keine Event Publisher sein.
- ▶ VB6/VBA-Klassen können Event Subscriber für VB- und COM-Klassen sein. VB6/VBA-Module können keine Event Subscriber sein. In VBS ist

Klassenevents

Event Publisher

Event Subscriber

der Empfang von Ereignissen eine Funktionalität, die vom Scripting Host bereitgestellt wird. Das Abonnement auf ein Ereignis wird in jedem Scripting Host anders definiert.

Komponenten erzeugen

Alle Visual Basic-Dialekte verfügen inzwischen auch über die Möglichkeit, COM-Server zu implementieren. Die Erzeugung neuer COM-Komponenten wird in den verschiedenen Visual Basic-Dialekten allerdings unterschiedlich unterstützt:

- VB** ▶ Das »große« Visual Basic kann seit Version 4.0 die wichtigsten Typen von COM-Komponenten in Form von COM-DLLs und COM-EXEs erstellen.
- VBA** ▶ VBA kann seit der Version 6.0 (Microsoft Office 2000) einige wenige ausgewählte Arten von COM-Komponenten erstellen.
- VBS** ▶ Mit VBS können COM-Komponenten nur in Form von Scriptlets erstellt werden.

Eingeschränkte Freiheiten

Visual Basic 6.0 automatisiert bei der Komponentenerstellung viele Dinge, die in anderen COM-Komponenten-Entwicklungsumgebungen (z.B. Visual C++) in der freien Entscheidung des Programmierers liegen. In den meisten nachstehend genannten Fällen ist es jedoch als positiv zu bewerten, den Programmierer in seiner Entscheidungsfreiheit einzuschränken:

- TypLibE** ▶ Visual Basic legt immer eine Typbibliothek an, die in die Komponentendatei hineinkompiliert wird.
- IDispatch** ▶ Visual Basic legt immer duale Schnittstellen an. Für jede Klasse gibt es also IDispatch-Unterstützung.
- ProgID** ▶ Die ProgID einer Klasse wird in der Form `Projektname.Klassenname` gebildet und ist stets konsistent zur Typbibliothek.
- AppIDs** ▶ Visual Basic legt für jede COM-Klasse eine eigene AppID an und nicht etwa – wie man vermuten könnte – eine AppID pro Komponente. Leider bekommt die AppID keinen Namen, sie erscheint im DCOM-Configuration Utility (DCOMCNFG) daher nur unter der AppID-GUID.



Sie können aber einen sprechenden Namen vergeben, indem Sie mit einem Registry-Editor in dem Standardwert eines AppID-Schlüssels einen Namen eintragen.

Dieses Kapitel

Dieses Kapitel gibt einen Überblick über die verschiedenen Aspekte der COM-Programmierung mit Visual Basic. Dabei wird sowohl die Nutzung von vorhandenen COM-Komponenten als auch die Erzeugung neuer Komponenten behandelt. Insbesondere werden folgende Konzepte und Schlüsselwörter besprochen:

- Objektvariablen
- Operator `new`
- Funktion `CreateObject()`
- Funktion `GetObject()`
- Verwendung von COM-Objekten einschließlich der Schnittstellenwahl
- Objektoperationen wie Wertzuweisung, Objektvergleich, Typüberprüfung
- Erzeugung von Klassen in Visual Studio 6.0 mit Einfach- und Mehrfach-schnittstellen.

3.2 Objektvariablen

Objektvariablen sind das Instrument in Visual Basic, um Objekte zu verwalten. Objekte können dabei sowohl Instanzen von Visual Basic-Klassen als auch von COM-Klassen sein.

Genau genommen ist eine Objektvariable ein Zeiger auf ein Objekt. Dieser Umstand ist wichtig für das Verständnis von Operationen auf Objektvariablen.

Zeiger auf Objekte

Im untypisierten VBS wird eine Objektvariable wie jede andere Variable auch deklariert. In VB6/VBA kann eine Objektvariable typisiert werden, indem bei der Deklaration der Klassenname angegeben wird. Dabei nimmt der Klassenname die Stelle des elementaren Datentyps ein.

	VBScript	VB6/VBA
Untypisierte Objektvariable (spätes Binden)	<code>Dim obj</code>	<code>Dim obj</code> <code>Dim obj as Variant</code> <code>Dim obj as Object</code>
Typisierte Objektvariable (frühes Binden)	nicht möglich!	<code>Dim obj As Klassenname</code>

Tabelle 3.1:
Deklaration von
Objektvariablen

Frühes und spätes Binden

Ob frühes oder spätes Binden angewendet wird, hängt von der Typisierung der Objektvariable ab. Wenn eine typisierte Variable verwendet wird, bedingt dies frühes Binden. Polymorphismus ist dann nicht möglich. Bei untypisierten Variablen werden die Aufrufe erst zur Laufzeit an die Implementierung gebunden (spätes Binden). Allerdings ist spätes Binden – wie üblich – erheblich langsamer als frühes Binden. Der Bindungszeitpunkt hat nichts damit zu tun, wann und wie die Instanziierung der Objekte durchgeführt wird.

**Bindungs-
zeitpunkt**

Frühes Binden ist nur in VB6/VBA möglich. VBS verwendet immer spätes Binden!



Benennung

- objX** Die übliche Benennung von Objektvariablen nach der ungarischen Notation besteht aus dem Präfix `obj` und dem nachfolgenden Klassennamen. Oft wird auch einfach ein kleines `o` verwendet. Sofern mehrere Instanzen einer Klasse verwendet werden, gehen Entwickler auch dazu über, den Klassennamen in Form eines Präfixes abzukürzen und den Rest des Namens zur Unterscheidung der Instanzen zu verwenden.

3.3 Instanziierung von COM-Klassen

Für die Instanziierung von COM-Klassen gibt es in Visual Basic zwei grundsätzliche Möglichkeiten:

- ▶ Erzeugung einer neuen Instanz einer COM-Klasse mit dem `new`-Operator
- ▶ Erzeugung einer neuen Instanz einer COM-Klasse mit der `CreateObject()`-Funktion.

GetObject()

Auf bestehende Instanzen aus der Running Objects Table (ROT) oder auf eine persistente Instanz greift man in Visual Basic mit der `GetObject()`-Funktion zu. Dies wird erst in Kapitel Zugriff auf bestehende Instanzen behandelt.

Intrinsic Objects

Einige Umgebungen instanziierten bereits bei ihrem Start bestimmte COM-Objekte automatisch. Innerhalb der Umgebung kann man auf diese Objekte ohne explizite Instanziierung zugreifen, z.B.

- ▶ Wscript im Windows Script Host
- ▶ Response in den Active Server Pages
- ▶ Debug in Visual Basic 6.0

Diese automatisch instanziierten COM-Objekte werden *eingebaute Objekte* oder *Intrinsic Objects* genannt.

3.3.1 Instanziiieren mit CreateObject()

CreateObject()

`CreateObject()` kann nur in Zusammenhang mit der `Set`-Anweisung benutzt werden.

```
set o = CreateObject  
(komponente.klasse.version  
[,computername])
```

Beispiele

```
Set o = CreateObject("Scripting.FileSystemObject")
Set o = CreateObject("MAPI.Session")
Set o = CreateObject("Word.Application")
Set o = CreateObject("Word.Application.8")
```

Listing 3.1: Beispiele für die lokale Instanziierung

3.3.2 Fernaktivierung mit DCOM

Ab Visual Basic 6.0, VBA 6.0 und VBS 5.0 unterstützt Visual Basic den zusätzlichen Parameter `Computername`. Somit ist es möglich, auf einfache Weise DCOM-Aufrufe zu realisieren.

Fernaktivierung

```
set o = CreateObject("Word.Application", "192.168.0.1")
```

Listing 3.2: Beispiel 1 für die entfernte Instanziierung einer COM-Klasse

```
set o = CreateObject("Scripting.FileSystemObject "
, "\\sonne2000")
```

Listing 3.3: Beispiel 2 für die entfernte Instanziierung einer COM-Klasse

Die auf den veränderten Instanziierungsbefehl folgende Verwendung der Klasse unterscheidet sich durch nichts von der Verwendung einer lokalen Instanz.

```
' === DCOM Beispiel
Sub DCOM()
Set fso = CreateObject("Scripting.FileSystemObject", "\\sonne2000")
For Each oDrive In fso.Drives
    say "DriveLetter: " & oDrive.DriveLetter
Next
End Sub
```

Listing 3.4: Verwendung einer COM-Klasse nach der Fernaktivierung
[CD:/code/Kapitel3_COMPraxis/Verschiedenes/CKH_Kapitel3_Verschiedenes.vbp]

Voraussetzung für das Funktionieren der Fernaktivierung ist jedoch, dass für die COM-Klasse (hier: `Scripting.FileSystemObject`) ein *Surrogat-Prozess* definiert wurde und die Zugriffsrechte ausreichen. Die einfachste Möglichkeit, einen Surrogat-Prozess zu definieren, ist die Aufnahme der Klasse in ein MTS-Package bzw. in eine COM+-Anwendung.

**Surrogat-Prozess
wird benötigt**

Für COM-EXE-Komponenten, die immer Out-process-Komponenten sind, ist natürlich kein Surrogat-Prozess notwendig. COM-EXE-Komponenten können auch gar nicht im MTS bzw. in COM+ konfiguriert werden. Viele Komponenten (z.B. Distributed Management Object – DMO, Windows Management Instrumentation – WMI, ActiveX Data Objects – ADO) besitzen eine eingebaute Möglichkeit, sich mit entfernten Systemen bzw. Datenbeständen zu verbinden. Bei diesen Komponenten ist es nicht nötig, DCOM zur entfernten Instanziierung zu nutzen.

Andere Fälle



Voraussetzung für die erfolgreiche Fernaktivierung ist auch das Vorliegen entsprechender Zugriffsrechte des Aufrufers auf dem Zielsystem, auf dem die Klasse instanziiert werden soll. Die DCOM-Rechte konfiguriert man mit dem Werkzeug DCOMCNFG.EXE.

3.3.3 Instanziiieren mit dem NEW-Operator in VB6/VBA

New Der New-Operator kann zur Erzeugung neuer Instanzen von COM-Klassen nur in VB6/VBA benutzt werden. In VBScript können mit `new` nur VBS-Klassen instanziiert werden. `New` benötigt immer Informationen aus der Typbibliothek der Komponente. Diese muss also zuvor eingebunden werden.

Dabei kann `New` entweder bei der Deklaration einer Variablen

```
Dim o as new Komponente.Klasse
```

oder mit einer Objektzuweisung mit `Set` verwendet werden.

```
Dim o As Komponente.Klasse
Set obj = new Komponente.Klasse
```



Die erste Variante ist marginal langsamer als die zweite, da Visual Basic intern bei jedem Objektzugriff eine zusätzliche Überprüfung der Objektvariablen durchführt.

Tabelle 3.2:
Beispiele zur
Anwendung des
New-Operators

Verwendung in Deklaration	Dim obj As new Word.Application
Verwendung mit Set	Dim obj As Word.Application Set obj = new Word.Application

3.3.4 New vs. CreateObject()

New versus CreateObject()

Die Instanziierung einer COM-Klasse mit `New` oder mit `CreateObject()` unterscheidet sich in folgenden Punkten:

- In VBS kann zur Instanziierung von COM-Klassen nur `CreateObject()` verwendet werden. Mit `New` können in VBS nur VBS-Klassen instanziiert werden. Ausnahmen bilden COM-Klassen, die in der gleichen Komponente wie die Scripting Engine selbst interpretiert sind. Eine solche Klasse ist `RegExp` (siehe Kapitel 22).
- Mit `CreateObject()` können nur COM-Klassen, aber keine VB-Klassen instanziiert werden.

Instanziierung von COM-Klassen

- Bei `New` sind der Komponenten- und der Klassenname im Programmcode statisch festgelegt. `CreateObject()` erwartet einen String, für den auch eine zuvor zur Laufzeit belegte Variable verwendet werden kann.

```
Komponente = "Word"
Klasse = "Application"
ProgID = Komponente & "." & Klasse
Set o = CreateObject(ProgID)
```

- `CreateObject()` erlaubt alternativ auch die Angabe einer CLSID.
- `CreateObject()` sucht die angegebene ProgID bzw. CLSID direkt in der Registry unter `HKEY_CLASSES_ROOT`. `New` verlangt, dass die angegebene ProgID in einer in das VB6/VBA-Projekt eingebundenen Typbibliothek definiert ist. Sofern die Typbibliothek nicht den Registry-Einträgen entspricht (vgl. Ausführungen zum Taskscheduler in Kapitel 10), weicht die bei `New` zu verwendende ProgID von der ProgID bei `CreateObject()` ab. Sofern der Klassenname innerhalb der eingebundenen Typbibliothek eindeutig ist, ist die Angabe des Komponentennamens bei `New` nicht notwendig.
- Bei der Verwendung innerhalb eines Objekts, dessen Klasse im Microsoft Transaction Server 2.0 installiert ist, erzeugt `CreateObject()` eine Instanz in einem neuen Objektkontext und `New` eine Instanz ohne Objektkontext.
- Für COM-Klassen, die in mehreren Versionen installiert sind, kann mit `CreateObject()` explizit definiert werden, welche Version instanziiert werden soll. `New` verwendet immer die aktuelle Version.
- Mit `CreateObject()` können auch Instanzen auf entfernten Rechnern erzeugt werden.

	VB	VBA	VBS
New-Operator	Ja	Ja	Nein!
<code>CreateObject()</code> (lokal)	Ja	Ja	Ja
<code>CreateObject()</code> (entfernt)	Ja, ab v6.0	Ja, ab v6.0	Ja, ab v5.0
<code>GetObject()</code>	Ja	Ja	Ja

Tabelle 3.3:
Aktivierungsmöglichkeiten von COM-Objekten in VB-Dialekten im Vergleich

3.3.5 Frühes oder spätes Binden

Oft wird behauptet, der Unterschied zwischen dem `New`-Operator und der `CreateObject()`-Funktion sei der Bindungszeitpunkt. Dies ist falsch, weil der Bindungszeitpunkt in Visual Basic lediglich durch die Deklaration bestimmt wird.

Bindungszeitpunkt

- Wird in der Deklaration der Variablen der Typ genannt, erfolgt frühe Bindung mit `QueryInterface()` und `vTables`.

- Nur wenn die Variable nicht typisiert (also vom Typ *Variant*) oder `As Object` deklariert ist, erfolgt späte Bindung mit `IDispatch`.



Visual Basic Script, das keine Typendeklaration kennt, verwendet folglich immer spätes Binden (mit `IDispatch`)!

Die Typisierung einer Objektvariablen auf eine COM-Klasse erfordert immer die Einbindung der Typbibliothek der Komponente. Ohne eine Typbibliothek sind VB6/VBA nicht besser dran als VBS und können ebenfalls nur spät binden.

```
Dim obj As new TestKomponente.TestKlasse
Set obj = CreateObject("TestKomponente.TestKlasse")
```

Listing 3.5: Frühe Bindung mit `CreateObject()`

```
Dim obj As TestKomponente.TestKlasse
Set obj = new TestKomponente.TestKlasse
```

Listing 3.6: Frühe Bindung mit `New`

```
Dim obj
Set obj = CreateObject("TestKomponente.TestKlasse")
```

Listing 3.7: Späte Bindung mit `CreateObject()`

```
Dim obj As Object
Set obj = new TestKomponente.TestKlasse
```

Listing 3.8: Späte Bindung mit `New`



TypeName()

Die VB-Funktion `TypeName()` funktioniert nicht nur bei elementaren Datentypen, sondern auch bei COM-Klassen und liefert den Klassennamen der COM-Klasse.

3.4 Zugriff auf bestehende Instanzen

GetObject() Während `CreateObject()` und `new` eine neue Instanz einer Klasse erzeugen, ist mit `GetObject()` der Zugriff auf bereits bestehende Instanzen einer Klasse aus der Running Objects Table (ROT) oder auf eine persistente Instanz (z.B. aus dem Dateisystem) möglich.

Parameter `GetObject()` hat zwei Parameter, von denen mindestens einer angegeben sein muss: Entweder ein Moniker für die gewünschte Instanz oder eine ProgID für die gewünschte Klasse.

```
set o = GetObject([moniker] [,ProgID])
```


Wird nur ein Moniker angegeben, sucht `GetObject()` die entsprechende Instanz und gibt bei Erfolg einen Zeiger darauf zurück. Sofern bei `GetObject()` der in diesem Fall optionale Parameter `ProgID` nicht angegeben ist, wird die Datei immer mit der Anwendung geöffnet, mit der die Dateiextension verknüpft ist. Die Verknüpfung ist in der Registry (in `HKEY_CLASSES_ROOT`) hinterlegt.

Moniker

Wird nur eine `ProgID` angegeben, sucht `GetObject()` irgendeine Instanz dieser Klasse in der Running Objects Table. Wird eine Instanz gefunden, wird der Zeiger darauf zurückgeliefert. Wird keine Instanz gefunden, verhält sich `GetObject()` wie `CreateObject()` und erzeugt eine neue Instanz. Dieses Feature funktioniert leider nicht in älteren VBScript-Implementierungen.

Class

Wenn sowohl ein Moniker als auch eine `ProgID` angegeben sind, wird die Instanz gesucht und mit der angegebenen Klasse gestartet – unabhängig davon, was in der Registry konfiguriert wurde. Soll die Datei durch eine andere als die in der Registry benannte Anwendung geöffnet werden, kann diese über die `ProgID` spezifiziert werden.

File-Extension

```
Set o = GetObject("d:\buch\docs\test.xls")
Set o = GetObject(, "Word.Application")
Set o = GetObject("d:\buch\docs\test.xls", "Excel.Sheet")
```

Listing 3.9: Beispiele zur Verwendung von `GetObject()`
[CD:/code/Kapitel3_COMPraxis/Verschiedenes/CKH_Kapitel3_Verschiedenes.vbp]

3.5 Arbeit mit COM-Objekten

Erst nach der Instanziierung können COM-Objekte verwendet werden. Alle Zugriffe erfolgen über die Objektvariable, die einen Zeiger auf das Objekt enthält. Die Objektvariable wird jeder Anweisung vorangestellt, danach folgt – getrennt durch einen Punkt (.) – der Name des Attributs oder der Methode.

Beispiel

Im folgenden Beispiel wird eine COM-Klasse verwendet:

```
Dim m As Mathe.Grundrechenarten
Set m = New Mathe.Grundrechenarten
m.x = 5
m.y = 10
MsgBox m.add
```

Listing 3.10: Beispiel zur Verwendung einer COM-Klasse
[CD:/code/Kapitel3_COMPraxis/Verschiedenes/CKH_Kapitel3_Verschiedenes.vbp]

3.5.1 Methodenaufrufe

Für Methodenaufrufe gelten die gleichen Regeln wie für VB-Prozeduren und VB-Funktionen.

Methoden ohne Rückgabewert

Ohne Rückgabewert Eine Methode ohne Rückgabewert wird aufgerufen, ohne die Parameter in Klammern zu setzen.

```
obj.Methodenname Parameter1,Parameter2,...
```

Call Die Angabe von runden Klammern zur Abgrenzung der Parameterliste führt in diesem Fall zu einem Fehler. Wenn die Klammern verwendet werden, muss dem Methodenaufruf das Schlüsselwort `Call` vorangestellt werden.

```
Call obj.Methodenname(Parameter1,Parameter2,...)
```

Methoden mit Rückgabewert

Mit Rückgabewert Der Aufruf von Methoden mit Rückgabewerten muss immer mit Klammern erfolgen.

```
var = obj.Methodenname(Parameter1,Parameter2,...)
```

3.5.2 Attributzugriffe

Attribute Der Zugriff auf Attribute erfolgt wie der Zugriff auf normale Variablen.

```
obj.Attributname = wert ' Schreiben eines Attributs
var = obj.Attributname ' Attribut lesen
```

3.5.3 Zugriff auf Unterobjekte

Unterobjekte Über die Punktnotation kann auch auf Unterobjekte zugegriffen werden. Wenn `obj` ein Zeiger auf ein Objekt ist, das ein Attribut `UnterObj` besitzt, welches wiederum ein Zeiger auf ein Objekt ist, dann kann auf die Methode `TueEtwas()` des `UnterObj` wie folgt zugegriffen werden:

```
obj.UnterObj.TueEtwas()
```

Dieses Verfahren ist über beliebig viele Hierarchieebenen in einem Objektmodell möglich:

```
obj.UnterObj.UnterUnterObj.UnterUnterUnterObj.TueEtwas()
```

Tip Wenn mehrfach auf ein Unterobjekt zugegriffen werden soll, dann ist es nicht empfehlenswert, jeweils von einem Oberobjekt zu dem Unterobjekt herunterzureichen. Dieses Vorgehen macht den Programmcode unübersichtlich und ist zudem aus Performance-Gesichtspunkten schlecht. Es ist besser für ein Unterobjekt, das mehrfach verwendet werden soll, eine eigene Objektvariable bereitzustellen und über diese zuzugreifen.

```
set uo = o.UnterObj.UnterUnterObj.UnterUnterUnterObj
uo.TueEtwas()
```

3.5.4 Arbeit mit Schnittstellen

Mehrfachschnittstellen sind unter COM ein wichtiges Thema. Ein Wechsel der Schnittstelle entspricht in der COM-Welt einem Aufruf von `IUnknown::QueryInterface()`.

QueryInterface().

Ein expliziter Aufruf von `QueryInterface()` ist in Visual Basic jedoch nicht vorgesehen. Er wäre zwar über einen direkten Aufruf der entsprechenden API-Funktion aus der COM-Bibliothek möglich – dieser schwierige Weg ist jedoch nicht nötig. VB ruft `QueryInterface()` immer dann neu auf, wenn eine Objektzuweisung an eine Objektvariable erfolgt. Wenn ein Objekt über mehrere Schnittstellen verfügt, dann hat VB immer die Schnittstelle im Zugriff, die dem Typ der Objektvariablen entspricht.

3.5.4.1 Schnittstellenwahl bei der Instanziierung

Wenn bei einer Objektzuweisung der Typ der Objektvariablen dem Klassennamen entspricht, zeigt die Objektvariable immer auf die Standardschnittstelle der Klasse.

Standard-schnittstelle

```
Dim k as KlassenName
Set k = new KlassenName
```

gleicher Typ

Listing 3.11: Zugriff auf die Standardschnittstelle

Wenn der Typ der Objektvariablen nicht dem Klassennamen entspricht, wird nach einer Schnittstelle dieses Namens in der Klasse gesucht. Das nötige `IUnknown::QueryInterface()` führt die VB-Laufzeitumgebung automatisch aus.

Andere Schnittstellen

```
Dim s as SchnittstellenNa
Set s = new KlassenName
```

verschiedener Typ

Listing 3.12: Zugriff auf eine andere Schnittstelle

3.5.4.2 Schnittstellenwechsel

Die beiden vorherigen Beispiele zeigten die Schnittstellenauswahl direkt bei der Instanziierung der Klasse. Bei einem Objekt mit Mehrfachschnittstellen kann die Schnittstelle jederzeit gewechselt werden, indem eine Wertzuweisung zwischen zwei Objektvariablen unterschiedlichen Typs erfolgt.

Wechsel durch Zuweisung

```
Dim y as SchnittstellenName1 ' alte Schnittstelle
Dim x as SchnittstellenName2 ' neue Schnittstelle
...
set x = y
```

Hierbei ist *y* ein Zeiger auf die alte Schnittstelle, und *x* muss als eine Objektvariable vom Typ der gewünschten Schnittstelle im gleichen Objekt deklariert worden sein. Nach `set x = y` zeigt dann *x* auf eine andere Schnittstelle im gleichen Objekt.

3.5.4.3 Einschränkungen in VBScript

**Keine Mehrfach-
schnittstellen
in VBS**

Da VBScript nur `IDispatch` verwenden kann, bleibt die Welt der Mehrfachschnittstellen VBScript auch hier vorenthalten. Das kann man positiv oder negativ sehen: Einerseits ist man von der Komplexität der Mehrfachschnittstellen entlastet, andererseits kann VBScript nur automationsfähige COM-Klassen nutzen und innerhalb dieser Klassen auch nur die Methoden, die über `IDispatch` bereitgestellt werden. Sind nicht alle Methoden aller benutzerdefinierten Schnittstellen auch in `IDispatch` einbezogen, bleibt VBScript ein Teil der Funktionalität verborgen.

Zum Glück werden heute viele – aber leider nicht alle – COM-Klassen mit dualen Schnittstellen ausgestattet, die sowohl Aufrufe über `IUnknown` als auch über `IDispatch` unterstützen.

Ausnahme: CDO 3.0

**Mehrfach-
schnittstellen
beim Scripting**

Mit den Collaboration Data Objects (CDO) in der Version 3.0 ist Microsoft einen anderen Weg gegangen, um Skriptsprachen den Umgang mit Mehrfachschnittstellen zu ermöglichen: Dort hat jede Schnittstelle eine eigene Implementierung von `IDispatch` und besitzt darüber hinaus eine Methode, die einen Zeiger auf eine andere Schnittstelle zurückliefern kann. So kann auch eine nicht `vTable`-fähige Sprache Mehrfachschnittstellen nutzen. Microsoft preist dieses Vorgehen als Vereinheitlichung der COM-Nutzung zwischen verschiedenen Sprachen (Visual Basic, C++,...) an, die allerdings auf Kosten der Konsistenz mit vielen anderen Komponenten geht.

3.5.4.4 Schnittstellentest

TypeOf-Operator

Den Test, ob ein Objekt eine bestimmte Schnittstelle unterstützt, kann man nur unter VB6/VBA durchführen. Eine Möglichkeit besteht darin, den Schnittstellenwechsel mit `Set` zu versuchen und einen möglichen Fehler abzufangen. Eleganter ist die Verwendung des `TypeOf`-Operators, der sich durch das folgende Beispiel erklärt. Das Beispiel verwendet Klassen aus der Windows Script Host Runtime Library (vgl. Kapitel 4).

Arbeit mit COM-Objekten

```
Set o = New IWSHNetwork_Class
' Ausgabe des Typnamens
say TypeName(o) ' IWSHNetwork2

' Prüfung des Typs
If TypeOf o Is IWSHNetwork_Class Then _
    say "IWSHNetwork_Class" ' Ja!
If TypeOf o Is IWSHNetwork Then _
    say "IWSHNetwork" ' Ja!
If TypeOf o Is IWSHShell Then _
    say "IWSHShell" ' Nein!
```

Listing 3.13: Testen einer Instanz auf die Unterstützung von Schnittstellen
[CD:/code/Kapitel3_COMPraxis/Verschiedenes/CKH_Kapitel3_Verschiedenes.vbp]

Das Beispiel zeigt, dass die Klasse `IWSHNetwork_Class` die gleichnamige Schnittstelle `IWSHNetwork_Class` und die Schnittstelle `IWSHNetwork` besitzt, aber nicht die Schnittstelle `IWSHShell`.

3.5.5 Ereignisse empfangen

COM bietet eine Infrastruktur an, um COM-Objekte auf Ereignisse reagieren zu lassen, die in anderen COM-Objekten passieren. Ein Ereigniskonsument muss dabei in der Regel eine andere Klasse sein. Eine einfaches VB-Modul kann nicht Ereigniskonsument sein. Einige Active Scripting Hosts (z.B. der WSH) ermöglichen es auch Active Scripts, auf COM-Ereignisse zu reagieren.

Eine VB-Klasse, die die Ereignisse von einem COM-Objekt abonnieren will, gibt bei der Deklaration der Objektvariablen für diese Klasse an, dass sie sich für deren Ereignisse grundsätzlich interessiert. Die Objektvariable muss eine globale (öffentliche oder private) Variable in der Klasse sein. Sie wird dadurch Ereigniskonsument (Event Subscriber) dieses Event Publishers. Der Ereigniskonsument definiert danach Ereignisbehandlungsroutinen für die Ereignisse, die ihn interessieren und die er behandeln möchte. Der Ereigniskonsument ist nicht verpflichtet, alle Events des Publishers zu behandeln. Ereignisbehandlungsroutinen werden nicht wie Attribute und Methoden durch einen Punkt, sondern durch einen Unterstrich (»_«) von dem Namen der Objektvariablen getrennt.

WithEvents

```
Dim WithEvents o As clsEventServer
Private Sub o_jetztistwaslos()
    MsgBox "Da war was los!"
End Sub
```

Listing 3.14: Abo eines Ereignisses in VB6/VBA (nicht in VBS!)

In VBScript gibt es das Schlüsselwort `WithEvents` nicht. VBScript ist aber in der Lage, auf Ereignisse zu reagieren, sofern der jeweilige Scripting Host dafür eigene Mechanismen anbietet.

VBScript

3.5.6 Tipps zur Objektverwendung

Diese Unterkapitel enthält drei wertvolle Tipps zum Umgang mit COM-Komponenten in Visual Basic.

3.5.6.1 Vermeidung tiefer Pfade



Beherrzigen Sie bei der Verwendung von Objektmodellen aus COM-Komponenten folgenden Tipp: Für jeden Punkt in einem Statement müssen einmal `QueryInterface()` oder – noch schlimmer – die `IDispatch`-Methoden `GetIDsOfNames()` und `Invoke()` aufgerufen werden. Vermeiden Sie also die Angabe von tiefen Pfaden, in dem Sie eine Objektvariable als Zwischenstufe verwenden.

Das ist schlecht:

```
obj.UnterObj.UnterUnterObj.TueEtwas()  
obj.UnterObj.UnterUnterObj.UnterUnterUnterObj.TueNochEtwas()  
obj.UnterObj.UnterUnterObj.UnterUnterUnterObj.TueDasAuch()
```

Das ist besser:

```
Set obj2 = obj.UnterObj.UnterUnterObj  
obj2.TueEtwas()  
set obj3 = obj2.UnterUnterUnterObj  
obj3.TueNochEtwas()  
obj3.TueDasAuch()
```

Die zweite Variante ist nicht nur schneller, sondern auch übersichtlicher.

With-Statement

**Tipparbeit sparen
und Übersichtlich-
keit gewinnen**

VB6, VBA und VBS ab Version 5.0 verfügen auch über ein Sprachkonstrukt, um eleganter mit (Unter-)Objekten arbeiten zu können: Das `With`-Statement setzt die nachfolgenden Objektzugriffe in den Kontext eines bestimmten Objekts.

```
With o.UnterObj.UnterUnterObj.UnterUnterUnterObj  
    .TueEtwas()  
    x = .Attribut  
    .Attribut = wert  
    MsgBox anderesObjekt.GibInformation()  
End With
```

Listing 3.15: Beispiel zur Verkürzung mit With

Innerhalb eines `With`-Blocks beziehen sich alle Ausdrücke, die mit einem Punkt beginnen, auf das nach `With` bezeichnete Objekt. Innerhalb des `With`-

Blocks dürfen auch andere Befehle vorkommen. Objektausdrücke, die nicht mit einem Punkt, sondern mit einer Objektvariablen beginnen, werden nicht auf das With-Statement bezogen.

Ein With-Block sollte nicht mit einem Sprungbefehl verlassen werden.

3.5.6.2 Datentypprobleme

In einigen Fällen kommt es bei der Verwendung von COM-Komponenten zu leichten Problemen mit Datentypen: Wenn ein Attribut oder eine Methode in Form eines Parameters einen bestimmten Datentyp erwarten, werden sie mit dem Datentyp *Variant* nicht immer glücklich sein.

Typprobleme

Dazu ein Beispiel.

```
Const FTPSERVER = "ftp://1.1.1.20"
Set oiC = CreateObject("InetCtls.Inet")
oiC.Execute FTPSERVER, "DIR"
Do While oiC.StillExecuting
    DoEvents ' oder WScript.Sleep()
Loop
```

Listing 3.16: Ein Beispiel aus der Komponente »Microsoft Internet Control«

Obige Befehle werden problemlos ausgeführt. Wenn jedoch der Aufruf von `Execute()` und die darauf folgende Warteschleife gekapselt werden sollen, kommt es zu einem Fehler.

```
Const FTPSERVER = "ftp://1.1.1.20"
inet_doit oiC, FTPSERVER, "DIR"
'--- Unterroutine
Sub inet_doit(oiC, url, command)
    oiC.Execute url, command
    Do While oiC.StillExecuting
        DoEvents ' oder WScript.Sleep()
    Loop
End Sub
```

Listing 3.17: Dieses Skript hat Probleme mit den Datentypen.
[CD: /code/komponenten/INET/inet_ftp.vbs]

Durch die Übergabe des Strings an eine Unteroutine geht die Information verloren, dass es sich um einen Untertyp *String* handelt, so dass `Execute()` den angegebenen URL als »falsch formatiert« zurückweist. Abhilfe schafft in solchen Fällen eine explizite Typumwandlung, hier mit `CStr()`.



```
oiC.Execute CStr(url), CStr(command)
```

3.6 Objektoperationen

Es sind folgende Operationen mit Objektvariablen möglich:

- ▶ Wertzuweisung
- ▶ Vergleich
- ▶ Typüberprüfung

3.6.1 Wertzuweisung

Wertzuweisung Eine Wertzuweisung erfolgt mit dem Gleichheitszeichen, erfordert aber im Gegensatz zur Wertzuweisung für den elementaren Datentyp ein zusätzliches Schlüsselwort: *Set*. Dieses Schlüsselwort ist notwendig, um den Zugriff auf das Objekt selbst von dem Zugriff auf das Default Property des Objekts zu unterscheiden.

```
Set obj2 = obj1
```



Wichtig: Diese Wertzuweisung verdoppelt nicht das Objekt, sondern kopiert nur den Zeiger. Es existiert anschließend nach wie vor nur ein Objekt, aber jetzt verweisen zwei Zeiger darauf. Eine Kopier-Operation für Objekte gibt es nicht. Diese muss der Entwickler bei Bedarf selbst für jede Klasse individuell (als eine Methode der Klasse) erstellen.

3.6.2 Objektvergleich

Vergleich Ein Vergleich zwischen zwei Objektvariablen kann mit dem Schlüssel *Is* durchgeführt werden. Dabei wird verglichen, ob die beiden Objektvariablen auf das gleiche Objekt zeigen (Referenzidentität). Wenn die Zeiger ungleich sind, ist die Aussage *falsch*. Ein Vergleich der in zwei verschiedenen Objekten gespeicherten Werte (also eine Prüfung auf Wertidentität) findet nicht statt.

```
If obj1 is obj2 Then ...
```

3.6.3 Typüberprüfung

TypeName() und VarType() Die Visual Basic-Funktion *TypeName()* funktioniert auch mit Objektvariablen und liefert den Klassennamen der übergebenen Objektinstanz. *TypeName()* liefert *Nothing*, wenn die Objektvariable auf keine Instanz verweist. Die Funktion *VarType()* liefert dagegen bei Objektvariablen stets 9 (*vbObject*), da es nicht für jede Klasse eine Typkonstante geben kann.

Objektooperationen

```
Set o = New Mathe
MsgBox typename(o)' liefert "Mathe"
MsgBox vartype(o)' liefert 9 = vbObject
```

Listing 3.18: Ermittlung des Klassennamens einer Objektvariablen

Nur in VB6/VBA gibt es zusätzlich den `TypeInfo`-Operator, mit dem eine Typüberprüfung in Ausdrücken stattfinden kann:

```
If TypeInfo obj Is KlassenName Then ...
```

TypeInfo

3.6.4 Beispiel

Das folgende Beispiel demonstriert alle drei vorgenannten Operationen. Die Funktion `vergleiche()` führt den Vergleich aus und gibt eine entsprechende Meldung aus. Im Hauptprogramm werden zuerst zwei Instanzen einer Klasse erzeugt, so dass der erste Vergleich natürlich eine Ungleichheit feststellt. Nachdem dann die Zeiger `o1` und `o2` gleichgesetzt wurden, liefert der zweite Vergleich nun das Ergebnis, dass die beiden Zeiger auf ein und dasselbe Objekt verweisen.

Beispiel

```
' Deklaration
Dim o1
Dim o2
' Zwei Instanzen erzeugen
Set o1 = New Mathe
Set o2 = New Mathe
' --- Erster Vergleich
vergleiche o1, o2' Ergebnis: sind ungleich!
' --- Wertzuweisung
Set o1 = o2
' -- Zweiter Vergleich
vergleiche o1, o2' Ergebnis: sind gleich!
' -- Typüberprüfung
If typename(o1) = "Mathe" Then MsgBox "ja!"
' If TypeInfo o1 Is Mathe Then MsgBox "ja!" ' Nur VB6/VBA

' ### Vergleich zwischen zwei Objektvariablen
Function vergleiche(o1, o2)
If o1 Is o2 Then
    MsgBox "Zeiger weisen auf das gleiche Objekt!"
Else
    MsgBox "Zeiger weisen auf verschiedene Objekte!"
End If
End Function
```

Listing 3.19: Beispiel zur Arbeit mit Objektvariablen

[CD:/code/Kapitel3_COMPraxis/Verschiedenes/CKH_Kapitel3_Verschiedenes.vbp]

3.7 Objektmengen (Collections)

Objektmengen Alle Visual Basic-Dialekte können in COM-Komponenten definierte Collections nutzen. VB6/VBA können darüber hinaus eigene Collections erzeugen. Die Collections können heterogen sein, d.h., sie können Instanzen verschiedener Klassen enthalten. Ebenso kann ein und dieselbe Instanz mehrfach enthalten sein.

Praktische Beispiele zur Arbeit mit Collections finden Sie in der Beschreibung der COM-Komponenten ab Kapitel 4.

3.7.1 Lesezugriff auf Collections

Der Zugriff auf Collections ist in allen VB-Dialekten gleich. Es gibt drei Möglichkeiten des Zugriffs auf die enthaltenen Objekte:

- ▶ Zugriff über einen Schlüsselwert
- ▶ Zugriff über den fortlaufenden Index
- ▶ Zugriff über Iteration über alle Elemente

3.7.1.1 Zugriff über einen Schlüsselwert

Item() VB-Collections besitzen ebenso wie die meisten COM-Collections die Methode `Item()`. Bei `Item()` kann ein String, der Schlüsselwert, angegeben werden.

Voraussetzung für den Zugriff über einen Schlüsselwert ist, dass ein eindeutiger Schlüsselwert für alle Objekte in der Objektmenge beim Aufbau der Collection angegeben wurde.

```
set obj = col.Item("key")
```

Diese Schreibweise lässt sich in der Regel verkürzen auf `col("key")`, da `Item` das Standardattribut ist. In VB6/VBA ist eine weitere Verkürzung auf `col!key` möglich. Damit ist der Schlüsselname dann aber zur Entwicklungszeit vorgegeben. Bei der Schreibweise mit Klammern kann dagegen auch ein Variablenname angegeben werden.

```
key = "ID"  
set obj = col.Item(key)
```

`Item()` wird oft auch als Attribut mit Parametern (siehe Kapitel 2) definiert.

3.7.1.2 Zugriff über den vorlaufenden Index

Bei `Item()` kann in der Regel auch eine Zahl (numerischer Index) angegeben werden, um die enthaltenen Objekte gemäß ihrer Reihenfolge anzusprechen.

Index

```
set obj = col.Item(1)' Das erste Element  
set obj = col.Item(7)' Das siebte Element
```

Collections in Visual Basic 6.0 beginnen bei 1 zu zählen. Viele Collections in COM-Objektmodellen starten dagegen bei 0!



3.7.1.3 Anzahl der Elemente

`Count()` liefert die Anzahl der in einer Collection enthaltenen Elemente.

Count()

3.7.2 Iteration über die Elemente einer Collection

Wenn die Schlüssel nicht bekannt sind und ein Element gesucht werden soll oder alle Elemente der Collection aufgelistet werden sollen, dann ist eine Iteration über alle Elemente notwendig.

For...Each

Visual Basic bietet mit `For...Each` eine sehr elegante Sprachkonstruktion an, um die Iteration über eine Collection durchzuführen. Diese Möglichkeit funktioniert oft auch dann, wenn die Collection keine oder unzuverlässige Informationen über die Anzahl der enthaltenen Objekte anbietet.

```
For each obj in col  
    obj.DoSomething  
Next
```

For...Each

For...Next

Eine alternative Möglichkeit ist die Iteration mit `For...Next`, sofern die Collection ein `Count()`-Attribut besitzt und numerische Indizes unterstützt.

```
For i = 0 To col.Count-1  
    col.item(i).DoSomething  
Next
```

For...Next

3.7.3 Veränderung einer Collection

Add() Meistens erlauben Collections das Hinzufügen und Entfernen von Objekten.

Hinzufügen

Das Hinzufügen neuer Elemente zu einer Collection erfolgt über die Methode `Add()`:

```
col.Add obj,[key],[Before],[After]
```

Erster Parameter ist ein Zeiger auf das Objekt, das der Collection hinzugefügt werden soll. Optional ist die Angabe eines Schlüssels in Form einer Zeichenkette. Einige VB-Collections und einige COM-Collections bieten darüber hinaus auch die Möglichkeit, mit zwei weiteren Schlüsseln anzugeben, an welcher Stelle der Liste das Objekt eingefügt werden soll.

Die in Visual Basic eingebaute Klasse `Collection` erlaubt heterogene Objektmengen. Will man sicherstellen, dass die Objektmenge homogen ist, so muss man dies durch eine eigene Mengen-Klasse implementieren.

Entfernen

Remove() Zum Entfernen aus der Liste ist der Schlüssel notwendig.

```
col.Remove key
```

3.7.4 Beispiel

In diesem Beispiel wird eine Liste erstellt, mit der ein Autor seine Veröffentlichungen verwalten kann. Die Objektmenge besteht aus den Klassen `Buch` und `Loseblattwerk` und ist daher eine heterogene Objektmenge. Die Klassen besitzen jeweils drei Attribute:

- ▶ Name
- ▶ Verlag
- ▶ Erscheinungsjahr

In der Routine `publikationsliste()` werden vier Bücher und zwei Loseblattwerke in die Routine aufgenommen. Ein Buch wird wieder entfernt.

Dann wird gezeigt, wie auf einzelne Elemente oder die ganze Menge zugegriffen werden kann.

```
Sub publikationsliste()  
Dim l As Loseblattwerk  
Dim b As Buch
```

Objektmengen (Collections)

```
Dim i As Integer
Dim name As String
Dim obj As Object

' --- Erzeugung der Objektmenge
Dim pub As New Collection

' --- Füllen der Objektmenge
Set b = New Buch
b.name = "Windows- und BackOffice-Scripting"
b.verlag = "Addison-Wesley"
b.erscheinungsjahr = 2000
pub.Add b, "WBS"

' --- Füllen der Objektmenge
Set b = New Buch
b.name = "COM-Komponenten-Handbuch"
b.verlag = "Addison-Wesley"
b.erscheinungsjahr = 2001
pub.Add b, "CKH"

' --- Füllen der Objektmenge
Set b = New Buch
b.name = "Windows Scripting, 2. Auflage"
b.verlag = "Addison-Wesley"
b.erscheinungsjahr = 2001
pub.Add b, "WS2"

' --- Füllen der Objektmenge
Set b = New Buch
b.name = "Die Windows Scripting-Box"
b.verlag = "Addison-Wesley"
b.erscheinungsjahr = 2001
pub.Add b, "BOX"

' --- Füllen der Objektmenge
Set l = New Loseblattwerk
l.name = "Praxishandbuch Windows NT"
l.verlag = "Interest Verlag"
l.erscheinungsjahr = 1998
pub.Add l, "NT4"

' --- Füllen der Objektmenge
Set l = New Loseblattwerk
l.name = "Praxishandbuch Windows 2000"
l.verlag = "Interest Verlag"
l.erscheinungsjahr = 2000
pub.Add l, "W2K"
```

COM-Praxis mit Visual Basic und VBScript

```
' --- Anzahl der Veröffentlichungen
say "In der Liste sind " & pub.Count & " Veröffentlichungen."

' --- Eine alte Veröffentlichung raus
pub.Remove "WBS"

' --- Anzahl der Veröffentlichungen
say "In der Liste sind " & pub.Count & " Veröffentlichungen."

' --- Ausgabe einzelner Objekte
ausgabe pub.Item("BOX")
ausgabe pub.Item(2) ' 2. Element
name = "CKH"
ausgabe pub.Item(name)

' -- Iteration über Collection
say "-- Liste von vorne:"
For Each obj In pub
    ausgabe obj
Next
say "-- Liste von hinten:"
For i = pub.Count() To 1 Step -1
    ausgabe pub(i)
Next

End Sub

' --- Hilfsroutinen
Sub ausgabe(p As Object)
    Dim alter, s
    alter = Year(Now) - p.erscheinungsjahr
    s = p.name
    If alter = 0 Then
        s = s & " ist dieses Jahr"
    ElseIf alter = 1 Then
        s = s & " ist letztes Jahr"
    Else
        s = s & " ist vor " & alter & " Jahren"
    End If
    s = s & " bei " & p.verlag & " erschienen"
    say s
End Sub

Sub say(s)
    Debug.Print s
End Sub
```

*Listing 3.20: Hauptprogramm zu diesem Beispiel
[CD:/code/3_COMPraxis/Publikationsliste/Publikationsliste.vbg].*

Objektvernichtung

Ausgabe des Beispiels

Das Beispiel liefert folgende Ausgabe:

```
In der Liste sind 6 Veröffentlichungen.  
In der Liste sind 5 Veröffentlichungen.  
Die Windows-Scripting Box ist dieses Jahr bei Addison-Wesley  
erschieden  
Windows-Scripting, 2. Auflage ist dieses Jahr bei Addison-Wesley  
erschieden  
COM-Komponenten-Handbuch ist dieses Jahr bei Addison-Wesley  
erschieden  
-- Liste von vorne:  
COM-Komponenten-Handbuch ist dieses Jahr bei Addison-Wesley  
erschieden  
Windows-Scripting, 2. Auflage ist dieses Jahr bei Addison-Wesley  
erschieden  
Die Windows-Scripting Box ist dieses Jahr bei Addison-Wesley  
erschieden  
Praxishandbuch Windows NT ist vor 3 Jahren bei Interest Verlag  
erschieden  
Praxishandbuch Windows 2000 ist letztes Jahr bei Interest Verlag  
erschieden  
-- Liste von hinten:  
Praxishandbuch Windows 2000 ist letztes Jahr bei Interest Verlag  
erschieden  
Praxishandbuch Windows NT ist vor 3 Jahren bei Interest Verlag  
erschieden  
Die Windows-Scripting Box ist dieses Jahr bei Addison-Wesley  
erschieden  
Windows-Scripting, 2. Auflage ist dieses Jahr bei Addison-Wesley  
erschieden  
COM-Komponenten-Handbuch ist dieses Jahr bei Addison-Wesley  
erschieden
```

3.8 Objektvernichtung

COM verfügt – im Gegensatz zum .NET-Framework – nicht über eine automatische Entsorgung des von nicht mehr benötigten Variablen belegten Speicherplatzes (Garbage Collection). Alle Visual Basic-Dialekte verfügen jedoch über eine Garbage Collection in ihren Laufzeitumgebungen.

**Garbage
Collection**

Die Vernichtung einer Objektinstanz erfolgt, wenn es keinen Zeiger auf eine Objektinstanz mehr gibt. Nicht mehr benötigte Speicherbereiche werden so automatisch freigegeben, wenn der Gültigkeitsraum (Scope) einer Objektvariablen verlassen wird.

Set obj = Nothing

Auch wenn oft das Gegenteil behauptet wird: die explizite Vernichtung mit der Anweisung

```
Set obj = Nothing
```

ist optional (siehe dazu [WES99b]).

Vorzeitige Freigabe

Sie können das Konstrukt `Set obj = Nothing` jedoch dazu nutzen, ein Objekt vor dem Verlassen des Gültigkeitsbereiches wieder freizugeben. Wenn mehrere Objektvariablen auf die betreffende Instanz verweisen, müssen all diese Objektvariablen auf »*Nothing*« gesetzt werden. Erst dann gibt Visual Basic die Instanz frei.

Beispiel

```
' Deklaration
Dim o1
Dim o2
Dim o3
' Zwei Instanzen erzeugen
Set o1 = New Mathe
' Wertzuweisungen
Set o2 = o1
Set o3 = o2
' Zeiger zurücksetzen...
Set o1 = Nothing
Set o2 = Nothing
' Objekt lebt noch!
o3.About
' Endgültige Objektvernichtung
Set o3 = Nothing
```

Listing 3.21: Beispiel zur Lebensdauer von Objekten [oo_vernichtung.vbs]

3.9 Erzeugung von COM-Komponenten

Komponenten erstellen

Es gibt inzwischen viele Sprachen und Entwicklungsumgebungen, die die Erzeugung von COM-Komponenten unterstützen. Exemplarisch zeigt dieses Kapitel zwei Methoden, die auf den bisher in diesem Buch vorgestellten Techniken aufbauen: Dies ist zum einen die Entwicklung von Komponenten

Erzeugung von COM-Komponenten

mit Visual Basic 6.0 und zum anderen die Erzeugung von Windows Script Components. Auch mit Visual Basic for Application 6.0 (als Teil von Office 2000) können mit Einschränkungen COM-Komponenten erstellt werden.

	VB 6.0/7.0	VBCCE	VBA 6.0	VBS 5.0/5.5
Sprachinterne Klassendefinition	Ja	Ja	Ja	Ja
Erzeugung von nicht-visuellen COM-Komponenten	Ja, DLL + EXE	Nein	tlw. DLL	Scriptlets
Erzeugung von ActiveX-Steuer-elementen	Ja	Ja	Nein	Nein

Tabelle 3.4:
Erzeugung von
Klassen und COM-
Komponenten in den
verschiedenen
VB-Dialekten und
-Entwicklungs-
umgebungen

Erzeugung von Komponenten mit Visual Basic 6.0

Die Visual Studio-Entwicklungsumgebung bietet komfortable Möglichkeiten, verschiedenartige COM-Komponenten zu erstellen. Die Ausführungen in diesem Kapitel basieren auf der Version 6.0 (als Teil der Visual Studio 98-Produktfamilie); die Funktionalitäten sind jedoch fast identisch auch in der Vorgängerversion 5.0 vorzufinden.

**Komponenten
programmieren
mit VB6**

Erzeugung von Komponenten mit VBScript

In Form sogenannter Scriptlets können auch COM-Komponenten mit VBScript erzeugt werden. Dieses Thema wird in diesem Buch jedoch nicht behandelt. Dazu sei an dieser Stelle auf das folgende Buch verwiesen:

Scriptlets

Holger Schwichtenberg:

Windows Scripting

Automatisierte Systemadministration

mit Visual Basic/VBScript und COM/.NET

ISBN 3-8273-1843-2

3.9.1 Projekt und Komponente

Eine COM-Komponente entspricht einem Projekt in Visual Basic. Ein Komponentenprojekt kann ebenso wie ein normales VB-Projekt aus verschiedenen Elementen (Formularen, Modulen, verschiedenen Designern) bestehen.

In der Visual Basic-Entwicklungsumgebung gibt es mehrere vorgefertigte Projekttypen für COM-Objekte:

Projekttypen

- ▶ ActiveX EXE
- ▶ ActiveX DLL
- ▶ ActiveX Control (ActiveX-Steuerelement)

- ActiveX Document EXE
- ActiveX Document DLL und
- Add-In.

Abbildung 3.1:
Projekttypen in
Visual Basic 6.0

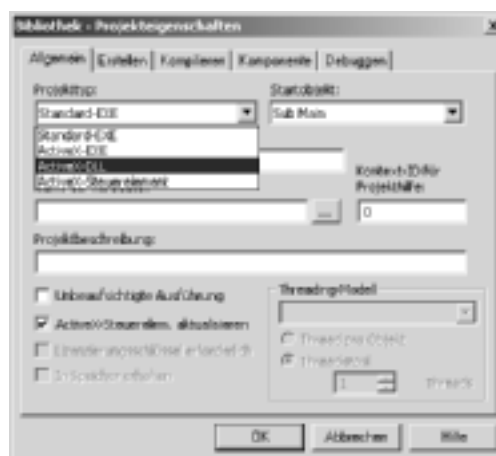


**ActiveX EXE,
ActiveX DLL**

Der Name *ActiveX* ist in Bezug auf die ersten beiden Projekttypen wieder einmal eine Benennung, über die man streiten kann. Auf einen Punkt können Sie sich jedoch verlassen: Alle mit VB6 erzeugten Komponenten unterstützen IDispatch und sind daher COM-automationsfähig.

Die Erstellung von COM-Add-Ins, ActiveX-Dokumenten und Steuerelementen ist nicht Thema dieses Buches. In diesem Kapitel wird es ausschließlich um die Erstellung von In-process-Komponenten (in Form von *ActiveX DLLs*) und Out-process-Komponenten (in Form von *ActiveX EXEs*) gehen.

Abbildung 3.2:
Der Projekttyp kann
auch nachträglich in
den Projekteigen-
schaften in VB6
noch beeinflusst
werden.



Zur Vereinfachung wird in diesem Buch der Begriff *Komponentenprojekt* als Oberbegriff über alle oben genannten Projekttypen verwendet.

Erstellen einer Komponente

Die Schritte zum Erstellen einer Komponente in Visual Basic 6.0 sind:

1. Starten Sie die Visual Basic-Entwicklungsumgebung und entscheiden Sie sich im Auswahldialog für den passenden Komponentenprojekttyp. Sie können zwar nachträglich den Projekttyp noch ändern; dies kann jedoch zu Problemen führen, wenn Sie die Komponente bereits kompiliert haben. Normalerweise werden Sie eine ACTIVEX DLL erstellen. **Projekttypauswahl**
2. Sie erhalten ein Projekt mit dem Namen *Projekt1* und ein Klassenmodul mit dem Namen *Class1*. Vergeben Sie immer direkt am Anfang einen sprechenden Namen für das Projekt und die Klasse, indem Sie die Namen in den Eigenschaftendialogen des Projekts und des Klassenmoduls ändern. Diese Entscheidung ist sehr wichtig, denn der Projektname wird zum Komponentennamen. **Benennung**
3. Stellen Sie die *Instancing*-Eigenschaft der Klasse ein. Die verschiedenen Optionen werden später in diesem Kapitel erklärt.
4. Deklarieren Sie innerhalb des Klassenmoduls die Attribute (öffentliche globale Variablen) und die privaten globalen Variablen. **Implementierung**
5. Deklarieren Sie innerhalb des Klassenmoduls die Ereignisse der Klasse.
6. Deklarieren und implementieren Sie die Methoden und privaten Unter-routinen der Klasse.
7. Kompilieren Sie die Komponente mit dem Menüpunkt DATEI/ERSTELLEN. Visual Basic führt dabei die Registrierung der Komponente auf dem lokalen System automatisch aus, so dass die Komponente direkt verwendet werden kann. **Kompilierung**

Beispielumgebung

Als Beispiel wird meistens in diesem Kapitel eine Benutzerverwaltung verwendet, weil diese viele anschauliche Anwendungsfälle für COM-Konzepte bietet.

Bitte beachten Sie, dass diese Klassen nur didaktische Beispiele sind: Diese Klassen führen keinerlei Veränderungen am System aus. Das Einzige, was die Klassen ausführen, ist die Ausgabe von Meldungen.



Das Beispiel ist realisiert in Form von zwei VB6-Projekten (siehe Tabelle). Beide Projekte sind zu einer Projektgruppe in der Datei BENUTZERBEISPIEL.VBG. Diese Datei finden Sie auf der Buch-CD-ROM unter [CD:/code/3_COMPraxis/BenutzerBeispiel/BenutzerBeispiel.vbg].

Projektname	Projekttyp	Projektdateiname	Dateiname (kompiliert)
Benutzer-Manager	ActiveX-DLL	BENUTZERMANAGER.VBP	BENUTZER-MANAGER.DLL
Benutzer-Manager Client	Standard-EXE	BENUTZERMANAGERCLIENT.VBP	BENUTZER-MANAGER-CLIENT.EXE

3.9.2 Klassendefinition

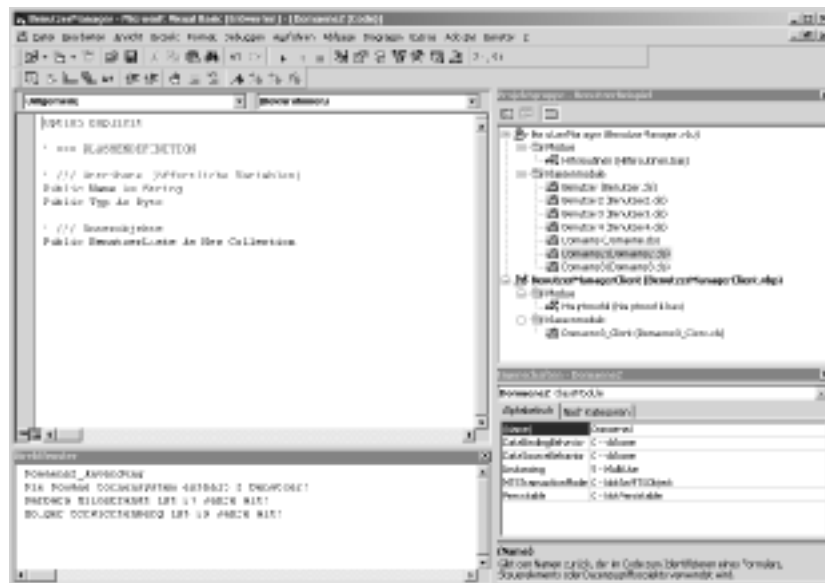
Klassenmodule

Die Klassen, aus denen die Komponente bestehen soll, werden in VB6 in Form von Klassenmodulen angelegt. Ein VB6-Komponentenprojekt muss immer aus mindestens einem Klassenmodul bestehen, da jede COM-Komponente aus mindestens einer Klasse bestehen muss.



Vergeben Sie immer einen sprechenden Namen für das Projekt und die Klasse. Diese Entscheidung ist sehr wichtig, denn der Projektname wird zum Komponentennamen.

Abbildung 3.3:
Die Projektgruppe
»BenutzerBeispiel.vbg« in der
VB6-Entwicklungs-
umgebung



Sichtbarkeit und Instanzierbarkeit

Optionen

In dem Eigenschaftsdialog eines Klassenmoduls kann unter dem Eintrag INSTANCING festgelegt werden, welche Sichtbarkeit und Instanzierbarkeit die COM-Klasse haben soll. Die nachstehende Tabelle zeigt die verfügbaren Optionen.

Erzeugung von COM-Komponenten

Klassen-Typ	Interner Wert	Erläuterung
Private	1	COM-Clients können die Klasse nicht nutzen. Die Klasse kann nur von anderen Klassen innerhalb der Komponente genutzt werden. Es wird daher keine CoClass und keine ProgID für diese Klasse angelegt.
PublicNot-Creatable	2	COM-Clients können die Klasse zwar nutzen, aber keine neue Instanz dieser Klasse erzeugen. Eine neue Instanz kann nur von einer anderen Instanz innerhalb der Komponente erstellt werden. Es wird eine CoClass angelegt, diese ist aber als <i>noncreatable</i> markiert. Es wird auch eine ProgID angelegt, obwohl dies eigentlich überflüssig ist, da mit <code>CreateObject()</code> keine Instanz erzeugt werden kann. Eine Instanz kann innerhalb der Komponente selbst nur mit dem <code>New</code> -Operator erzeugt werden.
SingleUse	3	Ein COM-Client kann eine Instanz erzeugen. Allerdings wird für jede Instanz der Klasse eine neue Instanz der Komponente erzeugt. Diese Einstellung ist nur bei COM-EXEs erlaubt.
MultiUse	5	Ein COM-Client kann eine Instanz erzeugen. Innerhalb einer Instanz der Komponente können beliebig viele Instanzen dieser Klasse laufen.
Global-SingleUse	4	In Erweiterung der Eigenschaft <i>SingleUse</i> wird hier bei der Einbindung der Typbibliothek automatisch eine Instanz erzeugt. Die Attribute und Methoden der Klasse stehen wie globale Variablen bzw. Funktionen zur Verfügung. Dies ist ähnlich wie bei Intrinsic Objects in Scripting Hosts. Diese Einstellung ist nur bei COM-EXEs erlaubt.
GlobalMulti-Use	6	Analog zu <i>GlobalSingleUse</i> , allerdings mit Mehrfachverwendung einer Komponenteninstanz (vgl. <i>MultiUse</i>)

Tabelle 3.5:
Option Instanzierbarkeit und Sichtbarkeit von mit VB6 implementierten COM-Klassen

Abbildung 3.4:
Eigenschaften eines
Klassenmoduls in
einer ActiveX DLL



3.9.3 Schnittstellen und Mitglieder

Schnittstelle

Jede COM-Klasse hat in Visual Basic 6.0 in der Grundform genau eine Schnittstelle, die den Namen der Klasse mit einem vorangestellten Unterstrich »_« trägt. Diese Standardschnittstelle muss nicht explizit definiert werden. Sie wird implizit für jedes Klassenmodul angelegt. Wenn eine Klasse Ereignisse definiert, so packt Visual Basic 6 diese Ereignisse automatisch in eine eigene Schnittstelle, die den Namen der Klasse mit zwei vorangestellten Unterstrichen »__« trägt.

Die folgende Grafik zeigt eine Komponente mit sechs Klassen und sieben Schnittstellen. Die Klasse `Domaene3` besitzt eine Schnittstelle für Attribute und Methoden und eine für Ereignisse.

Abbildung 3.5:
Anzeige der
Komponente
`BenutzerManager.dll` im
`com-TLBrowser`



Mitglieder

Die Definition der Mitglieder einer Schnittstelle geht so:

- ▶ Globale Variablen des Klassenmoduls werden zu Attributen.
- ▶ Öffentliche Unterroutinen des Klassenmoduls werden zu Methoden. Dabei werden – wie üblich in Visual Basic – Methoden mit Rückgabewert durch `Function()...End Function` deklariert, Methoden ohne Rückgabewert durch `Sub()...End Sub`.
- ▶ Ereignisse werden durch das vorangestellte Schlüsselwort `Event` deklariert.

Attribute

Methoden

Ereignisse

Bei der Benennung von Attributen und Methoden sollten Sie berücksichtigen, dass die Namen dann für den Anwender einprägsamer sind, wenn Sie für Attributnamen Adjektive und für Methodennamen Verben verwenden.



Kapselung

Mit den Schlüsselwörtern `Public` und `Private` wird gekennzeichnet, ob die Variable bzw. Unterroutine Teil der Schnittstelle der Klasse oder nur intern zugreifbar sein soll. Als `Public` deklarierte Unterroutinen sind Methoden der Klasse. Als `Public` deklarierte globale Variablen sind Attribute der Klasse.

Public und Private

Dabei gibt es leider eine Inkonsistenz hinsichtlich der Syntax:

- ▶ Bei Unterroutinendefinitionen wird `Public` oder `Private` den Schlüsselwörtern `Sub` oder `Function` vorangestellt. Dabei ist `Public` optional: Eine nicht mit `Private` oder `Public` spezifizierte Unterroutine ist also `Public`.
- ▶ Bei Variablendefinitionen ersetzen `Public` oder `Private` das Schlüsselwort `Dim`. `Dim` kann als Alternative zu `Private` verwendet werden. Es ist nicht erlaubt, `Public Dim x` oder `Private Dim x` zu schreiben.

Klassenbestandteil	VB-Dialekt	Deklaration
Attribut	VB6/VBA	<code>Public Name As Datentyp</code>
Attribut	VBScript/VB6/VBA	<code>Public Name</code>
Nichtöffentliche, globale Variable	VB6/VBA	<code>Private Name As Datentyp</code> oder: <code>Dim Name As Datentyp</code>
Nichtöffentliche, globale Variable	VBScript/VB6/VBA	<code>Private Name</code> oder: <code>Dim Name</code>

Tabelle 3.6:
Syntax zur Definition verschiedener Mitglieder

Tabelle 3.6:
Syntax zur Defini-
tion verschiedener
Mitglieder
(Forts.)

Klassen- bestandteil	VB- Dialekt	Deklaration
Methode mit Rückgabewert	VB6/VBA	Public Function FuncName() As Datentyp oder: Function FuncName() As Datentyp
Methode mit Rückgabewert	VBScript/ VB6/VBA	Public Function FuncName() oder: Function FuncName()
Methode ohne Rückgabewert	alle	Public Sub SubName() oder: Sub SubName()
Nichtöffentliche Unterroutine mit Rückgabewert	VB6/VBA	Private Function FuncName() As Datentyp
Nichtöffentliche Unterroutine mit Rückgabewert	VBScript/ VB6/VBA	Private Function FuncName()
Nichtöffentliche Unterroutine ohne Rückgabewert	alle	Private Sub SubName()
Öffentliche Konstanten (nur in Form von Konstantenlisten erlaubt)	alle	Enum NameDerListeKonstantenName = Wert End Enum
Nichtöffentliche Konstanten	VB6/VBA	Const KonstantenName = Wert

Beispiel für eine Klassendefinition

Das nachfolgende Listing zeigt die Definition einer Klasse mit Namen Benutzer. Die Klasse dient der Verwaltung von Benutzern in einem Netzwerk.

Attribute Die Klasse hat drei Attribute:

- BName: Benutzername
- VName: vollständiger Name
- Geb: Geburtsdatum des Benutzers

Methoden Die Klasse hat drei Methoden:

- SetzeKennwort(NeuesKennwort) legt das Kennwort des Benutzers neu fest. Die Methode ist eine Prozedur, die keinen Rückgabewert hat. Die Methode prüft die Kennwortlänge und gibt eine Rückmeldung über Dialogboxen.

Erzeugung von COM-Komponenten

- ▶ `AendereKennwort(AltesKennwort, NeuesKennwort)` erwartet im Gegensatz zur vorherigen Methode auch das alte Kennwort. Die Methode ist eine Prozedur, die keinen Rückgabewert hat. Die Methode vergleicht das bestehende Kennwort mit dem übergebenen Kennwort, prüft die Kennwortlänge und gibt eine Rückmeldung über Dialogboxen.
- ▶ `Alter()` ist eine Methode mit Rückgabewert. Die Funktion liefert das Alter des Benutzers, das auf Basis des aktuellen Datums und des Geburtsdatums errechnet wird.

Weiterhin enthält die Klassendefinition eine interne Variable und eine interne Funktion: **Interna**

- ▶ Die globale Variable `Kennwort` speichert das Kennwort des Benutzers. Dieser Wert sollte kein Attribut der Klasse sein, damit Anwender der Klasse es nicht lesen können und auch nicht in der Lage sind, einen beliebigen Wert in dieses Attribut zu schreiben.
- ▶ Die interne Funktion `pruefeKennwort(KennwortZeichenkette)` prüft, ob die übergebene Zeichenkette zwischen 4 und 11 Zeichen lang ist. Sie liefern `True` oder `False` zurück.

Option Explicit

```
' === KLASSEDEFINITION

' /// Interne Variablen
Public Kennwort As String

' /// Attribute (öffentliche Variablen)
Public BName As String
Public VollerName As String
Public Geb As Date

' /// Methoden (öffentliche Unterrouinen)
Public Sub SetzeKennwort(NeuesKennwort As String)

If pruefeKennwort(NeuesKennwort) Then
    Kennwort = NeuesKennwort
    MsgBox "Der Benutzer hat ein neues Kennwort!"
Else
    MsgBox "Das neue Kennwort entspricht nicht den
Kennwortrichtlinien!"
End If

End Sub

Public Sub AendereKennwort(AltesKennwort As String, NeuesKennwort As
String)
```

COM-Praxis mit Visual Basic und VBScript

```

If Kennwort = AltesKennwort Then
    ' Neues Kennwort setzen
    SetzeKennwort NeuesKennwort
Else
    MsgBox "Das alte Kennwort ist nicht richtig!"
End If

End Sub

Public Function Alter() As Byte

Alter = DateDiff("yyyy", Geb, Now())

End Function

' /// Interne, nicht-öffentliche Unterroutinen

Private Function pruefeKennwort(KennwortZeichenkette As String)

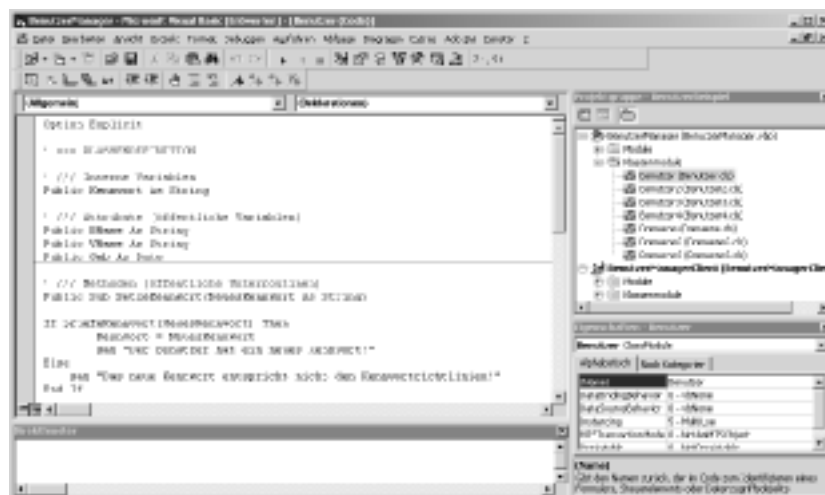
If Len(KennwortZeichenkette) > 3 And _
    Len(KennwortZeichenkette) < 12 Then
    pruefeKennwort = True
Else
    pruefeKennwort = False
End If

End Function

```

Listing 3.22: Klassenmodul »Benutzer« im Projekt [CKH_Kapitel3.vpb]

Abbildung 3.6:
Eingabe der Klasse
in der VB6-Ent-
wicklungsum-
gebung



Beispiel für eine Klassenverwendung

Das folgende Listing zeigt, wie diese Klasse verwendet werden kann. Sofern dieser COM-Client nicht Teil des gleichen VB6-Projekts ist wie der COM-Server, muss von dem COM-Client-Projekt aus ein *Verweis* (siehe PROJEKT/VERWEISE in der Entwicklungsumgebung) auf das Server-Projekt gesetzt werden.

```
' === Anwendungsbeispiel für die Klasse "Benutzer"
Sub Benutzer_Anwendung()

' Instanzieren
Dim u As Benutzer
Set u = New Benutzer

' Basisdaten setzen
u.Name = "Barbara Hildebrandt"
u.Geb = #1/3/1974#

' Attribute abfragen und Methoden aufrufen
MsgBox u.Name & " ist " & _
u.Alter & " Jahre alt!"
u.SetzeKennwort "HolgerSchwichtenberg" ' zu lang!
u.SetzeKennwort "Holger" ' OK
u.AendereKennwort "Holger", "Holgi" ' OK
u.AendereKennwort "Holger", "HolgerS" ' falsches altes Kennwort!

End Sub
```

*Listing 3.23: Benutzer_Anwendung() in
[CD:\Code\Kapitel3_COMPraxis\BenutzerBeispiel\
BenutzerBsp_Client\BenutzerManagerClient.vbp]*

Ausgabe des Beispiel

Das Beispiel liefert folgende Ausgabe:

Ausgabe

Barbara Hildebrandt ist 27 Jahre alt!

Das neue Kennwort entspricht nicht den Kennwortrichtlinien!

Der Benutzer hat ein neues Kennwort!

Der Benutzer hat ein neues Kennwort!

Das alte Kennwort ist nicht richtig!

3.9.4 Unterobjekte

Objekt- assoziationen

Ein Objekt kann Unterobjekte besitzen, wenn ein Attribut eines Objekts einen Zeiger auf ein anderes Objekt darstellt. Da Visual Basic-Objektvariablen die Objekte nicht enthalten können, sondern immer nur Zeiger auf Objekte sind, können in Visual Basic nur Assoziationen zwischen Objekten geschaffen werden, aber keine Aggregationen (vgl. Anhang A).

Beispiel

Mit Hilfe eines Unterobjekts soll modelliert werden, dass ein Benutzer zu einer Domäne gehört. Dazu benötigt man also eine weitere Klasse mit Namen `Domaene`.

Klassendefinition

```
' === KLASSENDEFINITION

' /// Attribute (öffentliche Variablen)
Public Name As String
Public Typ As Byte

Listing 3.24: Klassenmodul »Domaene« in
[CD:\Code\Kapitel3_COMPraxis\BenutzerBeispiel\
BenutzerBsp_Server\BenutzerManager.vbp]
```

Die Klasse `Benutzer` aus dem vorherigen Beispiel wird um das Attribut `Dom` erweitert. Die Klasse bekommt daher den Namen `Benutzer2`. Das folgende Listing zeigt nur einen Ausschnitt aus dem Code des Klassenmoduls, da der Rest gleich ist wie bei der Klasse `Benutzer`.

Klassendefinition

```
' /// Attribute (öffentliche Variablen)
Public BName As String
Public VName As String
Public Geb As Date
Public Dom As Domaene

Listing 3.25: Ausschnitt aus dem Klassenmodul »Benutzer2« in
[CD:\Code\Kapitel3_COMPraxis\BenutzerBeispiel\
BenutzerBsp_Server\BenutzerManager.vbp]
```

Anwendungsbeispiel

Das Anwendungsbeispiel zeigt die Zuordnung eines Objekts des Typs `Domaene` zu einem Objekt des Typs `Benutzer2`.

Klassen- verwendung

```
' === Anwendungsbeispiel für die Klasse "Benutzer2"
Sub Benutzer2_Anwendung()

' -- Instanzieren
Dim d As Domaene
Set d = New Domaene
d.Name = "Sonnensystem"
```

Erzeugung von COM-Komponenten

```
' -- Instanziiieren
Dim u As Benutzer2
Set u = New Benutzer2

' Basisdaten setzen
u.BName = "Babsi"
u.VName = "Barbara Hildebrandt"
u.Geb = #1/3/1974#

' -- Unterobjekt setzen
Set u.Dom = d

' -- Unterobjekt verwenden
say "Der Benutzer " & u.BName & _
" gehört zur Domäne " & _
u.Dom.Name

End Sub
```

*Listing 3.26: Benutzer2_Anwendung() in
[CD:\Code\Kapitel3_COMPraxis\BenutzerBeispiel\
BenutzerBsp_Client\BenutzerManagerClient.vbp]*

Ausgabe des Beispiels

Das obige Beispiel liefert folgende Ausgabe:

Ausgabe

Der Benutzer Babsi gehört zur Domäne Sonnensystem

3.9.5 Selbstreferenzierung

Jede Instanz kann über das Schlüsselwort *Me* einen Zeiger auf sich selbst erhalten. *Me* wird nicht benötigt, um Methoden des Objekts aus dem Objekt selbst heraus aufzurufen: Hier wird implizit ein *Me* vor den Aufruf gesetzt. *Me* kann aber benutzt werden, um anderen Instanzen einen Zeiger auf das aktuelle Objekt zu übermitteln.

Me

3.9.6 Property-Routinen

Der Nachteil bei einer Attributdefinition in einer COM-Klasse besteht darin, dass der Benutzer des Objekts das Attribut mit jedem beliebigen Wert aus dem Gültigkeitsbereich des Attributdatentyps beschreiben kann. Bei Attributen des Typs *Variant* kann der Benutzer also jeden beliebigen Wert übergeben. Aber auch in typisierten Umgebungen kann der Benutzer unerwünschte Werte schreiben, z.B.:

**Gültigkeits-
bereiche**

- Ein Attribut *Alter*, das als Datentyp *Byte* deklariert ist, kann von dem Objektbenutzer ohne Probleme mit dem Wert 250 beschrieben werden,

auch wenn bisher nur Methusalem mit seinen 969 Jahren dieses Alter erreicht hat.

- Ein Attribut `Kennwort`, das als *String* deklariert ist, kann von einem Benutzer mit jedem String belegt werden. Es gibt keine Möglichkeit, Kennwortrichtlinien (z.B. Mindestlänge) durchzusetzen.

Berechnete Attribute

Ein weiterer, mit einfachen Attributen nicht realisierbarer Punkt sind berechnete Attribute. Berechnete Attribute sind Daten, die nicht im Objekt gespeichert, sondern beim Zugriff jeweils erst aus dem internen Zustand des Objekts berechnet werden.

Attributzugriffe kapseln

Eine Lösung dafür wäre, Methodenaufrufe anstelle von Attributen zu verwenden. Dies führt jedoch dazu, dass immer zwei Methoden pro Eigenschaft benötigt werden (eine zum Lesen, eine zum Setzen).

Visual Basic bietet eine Möglichkeit an, Attributzugriffe so in Methoden zu kapseln, dass der Benutzer der Klasse dies nicht bemerkt. Beim lesenden oder schreibenden Zugriff auf ein Attribut wird nicht direkt ein Speicherbereich beschrieben; vielmehr wird der Zugriff von einer Methode abgefangen. Der eigentliche Wert ist intern woanders, z.B. in einer privaten globalen Variablen gespeichert. Dies ist das übliche Vorgehen in COM, das aber auch für VB-Klassen angeboten wird.

Property-Routinen

Das Instrument für die Attributkapselung sind sogenannte Property-Routinen. Eine Property-Routine ist immer ein Codeblock, der mit dem Schlüsselwort `Property` beginnt und mit `End Property` abgeschlossen werden muss.

3.9.6.1 Varianten von Property-Routinen

In VBScript/VB6 und VBA gibt es drei Varianten von Property-Routinen:

- `Property Get` zum Auslesen von Attributwerten
- `Property Let` zum Setzen von Attributen auf elementare Datentypen
- `Property Set` zum Setzen von Attributen auf Objektzeiger

Property Get

Lesen von Attributen

`Property Get` hat keine Parameter, aber einen Rückgabewert. Dieser wird aus einer privaten globalen Variablen ausgelesen. `Property Get` kann auch Zeiger auf Objekte zurückliefern; dann muss allerdings die Wertzuweisung an den Funktionsnamen mit `Set` erfolgen.

```
Public Property Get attributname() [As Variant]
    attributname = internesAttribut
End Property
```

Property Let und Property Set

Bei Property Let und Property Set erhält die Property-Routine den dem Attribut zugewiesenen Wert als Parameter, der dann – nach einer optionalen Prüfung – in die private globale Variable gespeichert werden kann. In VB6/VBA sollten die Schlüsselwörter ByVal und As Variant dabei angegeben werden. In VBS ist diese Spezifikation nicht erlaubt.

Setzen von Attributen

```
Public Property Let attributname([ByVal] NeuerWert [As Variant])
    internesAttribut = NeuerWert
End Property
```

Property Set ist ähnlich aufgebaut; bei der Wertzuweisung ist jedoch Set zu verwenden.

```
Public Property Set attributname (NeuerZeiger [As Object])
    set internesAttribut = NeuerZeiger
End Property
```

Mit Property-Routinen kann auch erreicht werden, dass ein Attribut nicht beschrieben (read-only) oder aber nicht gelesen (write-only) werden kann, da nicht zwingend vorgeschrieben ist, dass stets beide Property-Routinen implementiert werden müssen:

- ▶ Wenn ein Attribut vom Objektnutzer nur gelesen werden, aber der Wert von dem Nutzer nicht veränderbar sein soll (read-only), implementieren Sie Property Get, aber nicht Property Let/Set.
- ▶ Wenn ein Attribut vom Objektnutzer zwar geschrieben werden, aber er den Wert danach nicht mehr auslesen können soll (write-only), implementieren Sie Property Let/Set, aber nicht Property Get.

Read-only

Write-only

Ein Anwendungsfall für *Read-only-Attribute* ist beispielsweise die Versionsnummer einer Klasse. *Write-only-Attribute* sind seltener. Ein typischer Anwendungsfall sind Passwörter: Diese kann der Benutzer setzen, aber später nicht mehr auslesen, da sie nicht im Klartext abgespeichert, sondern üblicherweise in Form eines Hash-Codes abgelegt werden, aus dem das Klartext-Passwort nicht mehr ermittelbar ist.

Anwendungsbeispiele

3.9.6.2 Beispiel

Das Beispiel zur Veranschaulichung der Arbeitsweise der Property-Routinen ist naturgemäß etwas länger. Die Klasse Benutzer3 besitzt vier Attribute:

Die Klasse »Benutzer« ist mit Property-Routinen realisiert

- ▶ Name ist ein normales Attribut, das ohne Property-Routinen realisiert ist und daher beliebig gelesen und beschrieben werden kann.
- ▶ Geb kann gelesen und beschrieben werden. Beim Schreibzugriff findet jedoch eine Konsistenzprüfung statt. Bei einem Alter von über 150 Jahren wird nachgefragt. Der Wert wird intern in der Variablen Intern_Geburtstag gespeichert.

- ▶ Alter ist ein berechnetes Attribut, das nur gelesen werden kann. Alter berechnet sich aus der Differenz des Geburtstags und des aktuellen Datums.
- ▶ Kennwort ist ein Attribut, das nur gesetzt werden kann. Es wird überprüft, ob das Kennwort mindestens drei Buchstaben lang ist.

Zusätzlich besitzt die Klasse eine Methode Info(), die Name und Alter ausgibt.

Klassendefinition

```
' === KLASSEDEFINITION

' /// Interne Variablen
Private intern_Geburtstag As Date
Private intern_Kennwort As String
' /// Öffentliche Attribute (normale Attribute)
Public Name As String
' /// Öffentliche Attribute (Property-Routinen)

' --- Attribut Alter (write-only)
Public Property Let Geb(ByVal vNewValue)
Dim Jahre
If Not IsDate(vNewValue) Then _
    MsgBox "Ungültiges Datum!": Exit Property
Const Nachfrage = _
    "Sie sind sicher, dass Sie älter als 150 Jahre sind?"
Jahre = DateDiff("yyyy", vNewValue, Now())
If Jahre > 150 Then
    If MsgBox(Nachfrage, vbYesNo) = vbYes Then
        intern_Geburtstag = vNewValue
    Else
        intern_Geburtstag = Now
    End If
Else
    intern_Geburtstag = vNewValue
End If
End Property
' --- Attribut Alter (read-only)
Public Property Get Alter() As Integer
    Alter = DateDiff("yyyy", intern_Geburtstag, Now())
End Property
' --- Attribut Kennwort (write-only)
Public Property Let Kennwort(ByVal vNewValue)
If Len(vNewValue) < 3 Then
    MsgBox "Kennwort zu kurz!"
Else
    intern_Kennwort = vNewValue
End If
End Property
```


Erzeugung von COM-Komponenten

```
' /// Methoden
Sub info()
MsgBox "Benutzer " & Name & " ist " & Alter & " Jahre alt!"
End Sub
```

*Listing 3.27: Klassenmodul »Benutzer3« in
[CD:\Code\Kapitel3_COMPraxis\BenutzerBeispiel\
BenutzerBsp_Server\BenutzerManager.vbp]*

Verwendung der Klasse

Das folgende Listing zeigt die Verwendung der obigen Klasse.

```
' === Anwendungsbeispiel für die Klasse "Benutzer3"
Sub Benutzer3_Anwendung()

Dim u As New Benutzer3
Set u = New Benutzer3
u.Name = "Methusalem"
u.Geb = #1/1/1031# ' Property Let
MsgBox "Alter = " & u.Alter ' Property Get -> "Alter = 969"
u.info ' Methodenaufruf -> "Methusalem ist 969 Jahre alt!"
u.Kennwort = "123" ' Property Set ' -> Kennwort zu kurz!
' MsgBox u.Kennwort ' Property Get -> Nicht erlaubt!

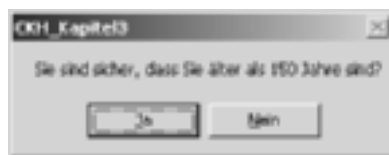
End Sub
```

*Listing 3.28: Benutzer3_Anwendung() in
[CD:\Code\Kapitel3_COMPraxis\BenutzerBeispiel\
BenutzerBsp_Client\BenutzerManagerClient.vbp]*

**Klassen-
verwendung**

Ausgabe des Beispiels

Das Beispiel zeigt zunächst die folgende Dialogbox an:



Ausgabe

*Abbildung 3.7:
Dialogbox von
Benutzer3_
Anwendung()*

Wenn JA geklickt wird, wird folgender Text ausgegeben:

Alter = 970

Benutzer Methusalem ist 970 Jahre alt!

Wenn NEIN geklickt wird, bleibt der Wert Alter auf 0 stehen und es wird folgender Text ausgegeben:

Alter = 0

Benutzer Methusalem ist 0 Jahre alt!

3.9.6.3 Das Default Property-Problem

Standardattribut Property-Routinen bieten auch die Möglichkeit, ein Standardattribut festzulegen, das immer dann verwendet wird, wenn auf eine Objektvariable ohne Angabe eines Mitgliedsnamens zugegriffen und zugleich ein elementarer Datentyp erwartet wird. Dies ist jedoch sehr schlechter Stil, denn eine Objektvariable bezeichnet normalerweise das ganze Objekt und nicht nur ein Attribut. Als Zeiger auf das ganze Objekt wird eine Objektvariable bei der Zuweisung an eine andere Objektvariable und bei der Übergabe als Parameter an eine Unteroutine verstanden.

Beispiel Gegeben sei eine Klasse `clsProblem` mit dem Standardattribut `Name` und dem zusätzlichen Attribut `Bemerkung`. Folgendes Code-Beispiel macht die Verwirrung deutlich.

```
Public Bemerkung

Dim i_standardattribut

Public Property Get Name() As Variant
    Name = i_standardattribut
End Property

Public Property Let Name(ByVal vNewValue As Variant)
    i_standardattribut = vNewValue
End Property
```

*Listing 3.29: Klassenmodul »clsProblem« in
[CD:\Code\Kapitel3_COMPraxis\Verschiedenes\
CKH_Kapitel3_Verschiedenes.vbp]*

```
Sub attributausgabe(obj)
    MsgBox obj
    MsgBox obj.Bemerkung
End Sub

Sub teste_problem()
    Dim p As New clsProblem
    p.Name = "Default Property-Problem"
    p.Bemerkung = "Bug oder Feature?"
    MsgBox p ' Standardattribut
    x = p ' Standardattribut
    Set y = p ' Zeiger!
    attributausgabe p ' Zeiger !
    Call attributausgabe(p) ' Zeiger!
    attributausgabe (p) ' Standardattribut
End Sub

' ### Unteroutine zum Default Property Problem
Sub attributausgabe(obj)
    MsgBox obj
```

Erzeugung von COM-Komponenten

```
MsgBox obj.Bemerkung  
End Sub
```

*Listing 3.30: b6_oo_defaultpropproblem() in
[CD:\Code\Kapitel3_COMPraxis\Verschiedenes\
CKH_Kapitel3_Verschiedenes.vbp]*

MsgBox p und die Zuweisung x = p sind Zugriffe auf das Standardattribut, die Zuweisung mit Set und die beiden ersten Prozeduraufrufe verwenden den Objektzeiger. Die Prozedur attributausgabe() erhält dabei einen Zeiger auf den Zeiger. MsgBox obj greift also wieder auf das Standardattribut zu.

Erläuterung

Beim letzten Aufruf tritt allerdings wieder ein Problem zu Tage, das es in ähnlicher Form auch bei elementaren Datentypen gibt: Wenn eine Unteroutine nur einen Parameter hat, dieser Parameter in runden Klammern steht und das Unterprogramm keinen Rückgabewert liefert oder das Hauptprogramm den gelieferten Rückgabewert nicht empfängt, dann wird nicht der Zeiger übergeben, sondern das Standardattribut!

Das ist für den ersten Befehl MsgBox obj in der Prozedur attributausgabe() noch kein Problem, aber beim Zugriff auf obj.Bemerkung kommt es zum Programmabbruch, denn der übergebene String hat keine Attribute. Das kann in einer anderen Konstellation noch unangenehmer werden: Wenn das Standardattribut selbst wieder ein Zeiger auf ein Objekt ist, dann erhält die Unteroutine zwar einen Zeiger, aber den falschen!

Achten Sie genau auf die Regeln für die Klammersetzung bei Prozeduren und Funktionen. Sie können das Problem auch vermeiden, indem Sie sich die Mühe machen, Standardattribute immer explizit anzugeben.



3.9.7 Erzeugung von Collections in VB6/VBA

VB6/VBA bringen eine vordefinierte Klasse Collection mit, von der Sie Instanzen erzeugen können, um Objekte zu einer Menge zusammenzufassen.

In VBS können Sie zur Verwendung von Objektmengen die COM-Klasse Scripting.Dictionary verwenden.

Die Klasse Collection definiert vier selbsterklärende Methoden:

Mitglieder

- Sub Add(Item, [Key], [Before], [After])
Fügt einem Collection-Objekt ein Element hinzu
- Function Count() As Long
Gibt die Anzahl der Elemente in einer Collection zurück

► Function Item(Index|Key)

Gibt ein bestimmtes Element eines Collection-Objekts mittels Position oder Schlüssel zurück

► Sub Remove(Index)

Entfernt ein Element aus einem Collection-Objekt

Beispiel

Nun soll jede Domäne eine Liste aller Benutzer speichern, die zu der Domäne gehören. Dazu erhält die in früheren Beispielen verwendete Klasse *Domaene* ein Unterobjekt vom Typ *Collection*. Das Unterobjekt heißt *BenutzerListe*. Die Klasse heißt nun *Domaene2*.

Klassendefinition

```
Option Explicit

' === KLASSENDEFINITION

' /// Attribute (öffentliche Variablen)
Public Name As String
Public Typ As Byte

' /// Unterobjekte
Public BenutzerListe As New Collection
```

Listing 3.31: Klassenmodul »Domaene2«

Anwendung der Klasse

Klassen- verwendung

Im Anwendungsbeispiel wird als Benutzerobjekt wieder auf die Klasse *Benutzer2* zurückgegriffen. Die Routine erzeugt eine Domäne und zwei Benutzer. Nach dem Füllen der Benutzerobjekte werden diese jeweils der Domäne hinzugefügt, indem die *Add()*-Methode der *Collection*-Klasse verwendet wird. Mit Hilfe der *Count()*-Methode wird danach die Anzahl der Mitglieder in der Benutzerliste ausgegeben. Anschließend wird *For...Each* verwendet, um alle Benutzer in der Liste auszugeben.

```
' === Anwendungsbeispiel für Collections
Sub Domaene2_Anwendung()
Dim d As Domaene2
Dim u As Benutzer2

' -- Domäne anlegen
Set d = New Domaene2
d.Name = "Sonnensystem"

' -- Benutzer anlegen
Set u = New Benutzer2
u.BName = "BH"
```

Erzeugung von COM-Komponenten

```
u.VName = "Barbara Hildebrandt"
u.Geb = #1/3/1974#
' -- Benutzer in Domäne aufnehmen
d.BenutzerListe.Add u

' -- Benutzer anlegen
Set u = New Benutzer2
u.BName = "HS"
u.VName = "Holger Schwichtenberg"
u.Geb = #8/1/1972#
' -- Benutzer in Domäne aufnehmen
d.BenutzerListe.Add u

' -- Anzahl der Benutzer ausgeben
say "Die Domäne " & d.Name & _
" enthält " & d.BenutzerListe.Count & _
" Benutzer!"

' -- Alle Benutzer einer Domäne auflisten
For Each u In d.BenutzerListe
    say u.VName & " ist " & _
    u.Alter & " Jahre alt!"
Next

End Sub
```

Listing 3.32: Domaene2_Anwendung()

Ausgabe des Beispiels

Das Beispiel liefert folgende Ausgabe:

Ausgabe

Die Domäne Sonnensystem enthält 2 Benutzer!

Barbara Hildebrandt ist 27 Jahre alt!

Holger Schwichtenberg ist 29 Jahre alt!

3.9.8 Ereignisse

COM unterstützt neben Attributen und Methoden noch eine weitere Art von Schnittstellenmitgliedern: Ereignisse. Ein Ereignis ist weder etwas, das man lesen oder schreiben kann, noch etwas, was man von außen aufrufen kann. Ein Ereignis ist aus der Sicht des COM-Clients ein Ankerpunkt, an den er eigene Routinen anhängen kann. Man sagt auch, ein COM-Client ist ein Ereigniskonsument, der Ereignisse abonniert.

Events

Aus der Sicht des COM-Servers ist ein Ereignis etwas, was man auslösen kann. Bei der Auslösung werden alle Unterroutrinen aufgerufen, die die COM-Clients mit dem bestimmten Ereignis verankert haben. In diesem

Zusammenhang wird ein COM-Server auch Event Publisher und ein COM-Client auch Event Subscriber genannt.

In Visual Basic gibt es zwei Arten von Ereignissen:

- ▶ Zwei implizite Klassenereignisse, die jede COM-Klasse automatisch besitzt
- ▶ selbst definierte Ereignisse.

3.9.8.1 Klassenereignisse

**Class_Initialize()
und
Class_Terminate()**

Oft ist es sinnvoll, dass bei der Erzeugung einer neuen Objektinstanz oder bei deren Vernichtung Vorgänge ausgeführt werden, z.B. die Initialisierung von Variablen bei der Objekterzeugung oder die Freigabe von Ressourcen bei der Objektvernichtung. Sicherlich ist es möglich, dafür jeweils eine Methode anzubieten. Jedoch ist das Objekt dann auf die Disziplin seines Nutzers angewiesen.

Dies kann mit den Ereignissen `Class_Initialize()` und `Class_Terminate()` besser gelöst werden. Diese beiden Ereignisse stellen einen Sonderfall dar, bei dem das Schlüsselwort `Class` für die Selbstreferenz auf die eigene Klasse steht. Event Publisher ist in diesem Fall die VB-Laufzeitumgebung.

```
Private Sub Class_Initialize()
End Sub

Private Sub Class_Terminate()
End Sub
```

**Konstruktoren
und Destruktoren**

Der `Class_Initialize`-Event ist nur ein Teil dessen, was in anderen objekt-orientierten Programmiersprachen als Konstruktor bezeichnet wird. Üblicherweise ist man in der Lage, bei der Objekterzeugung auch Parameter anzugeben, die dann einer bestimmten Routine, Konstruktor genannt, übergeben werden. Erst Visual Basic.NET unterstützt die Möglichkeit, bei der Instanziierung Parameter anzugeben, die `Class_Initialize` empfangen können.

Beispiel

Das nachfolgende Listing zeigt die Klasse `Benutzer4`. Bei der Instanziierung wird das Kennwort auf den Standardwert »123« gesetzt und eine Meldung ausgegeben. Bei der Vernichtung eines Objekts wird ebenfalls eine Meldung ausgegeben.

Klassendefinition

```
Option Explicit

' /// Interne Variablen
Public Kennwort As String
```

Erzeugung von COM-Komponenten

```
' /// Attribute (öffentliche Variablen)
Public BName As String
Public VName As String
Public Geb As Date

' /// Klassenereignisse

Private Sub Class_Initialize()
    MsgBox "Ein neuer Benutzer wurde angelegt!"
    Kennwort = "123"
End Sub

Private Sub Class_Terminate()
    MsgBox "Der Benutzer " & BName & " wurde gelöscht!"
End Sub
```

*Listing 3.33: Klassenmodul »Benutzer4« in
[CD:\Code\Kapitel3_COMPraxis\BenutzerBeispiel\
BenutzerBsp_Server\BenutzerManager.vbp]*

Anwendung der Klasse

Im folgenden Code wird eine Instanz der Klasse Benutzer4 erzeugt und später wieder vernichtet.

```
' === Anwendungsbeispiel für die Klasse "Benutzer4"
Sub Benutzer4_Anwendung()

' -- Instanzieren
Dim u As Benutzer4
Set u = New Benutzer4

' Basisdaten setzen
u.BName = "Babsi"
u.VName = "Barbara Hildebrandt"
u.Geb = #1/3/1974#

' Objekt löschen!
Set u = Nothing

End Sub
```

*Listing 3.34: Benutzer4_Anwendung() in
[CD:\Code\Kapitel3_COMPraxis\BenutzerBeispiel\
BenutzerBsp_Client\BenutzerManagerClient.vbp]*

**Klassenverwen-
dung**

Ausgabe des Beispiels

Ausgabe Das Beispiel liefert folgende Ausgabe:

Ein neuer Benutzer wurde angelegt!

Barbara Hildebrandt ist 27 Jahre alt!

Der Benutzer Babsi wurde gelöscht!

3.9.8.2 Selbst definierte Ereignisse

Event Publisher Eine in VB6 oder VBA entwickelte Klasse kann eigene Ereignisse definieren, auf die die Umwelt des Objekts reagieren kann. Ereigniskonsument muss dabei eine andere Klasse sein. Ein einfaches VB-Modul kann nicht Ereigniskonsument sein.

VBScript erlaubt keine Ereignisdefinitionen. VBScript ist aber in der Lage, auf Ereignisse zu reagieren, sofern der jeweilige Scripting Host dafür eine Unterstützung anbietet.

Ereignisse definieren

Event Eine Klasse wird Event Publisher, indem sie ein Ereignis mit dem Schlüsselwort `Event` definiert und dieses Ereignis irgendwann mit `RaiseEvent()` auslöst.

```
Event jetztistwaslos()  
' ...  
Sub dosomething()  
' ...  
RaiseEvent jetztistwaslos()  
' ...  
End Sub
```

Listing 3.35: Definition und Aussenden eines Ereignisses in VB6 und in VBA (nicht in VBS!)

Beispiel

Das Beispiel `Domaene2_Anwendung()` wird nun so umgebaut, dass die Klasse `Domaene3` die Benutzerliste durchläuft und in Form von COM-Ereignissen an den Client liefert. Die Klasse definiert ein Ereignis `AbfrageErgebnis(s)`, das als Parameter einen String mitliefert. Die Methode `HoleNutzer()` übermittelt auf diesem Wege die Namen der Nutzer an den Client.

```
Option Explicit  
  
' === KLASSENDEFINITION  
  
' /// Attribute (öffentliche Variablen)
```


Erzeugung von COM-Komponenten

```
Public Name As String
Public Typ As Byte

' /// Unterobjekte
Public BenutzerListe As New Collection

' /// Ereignis
Event AbfrageErgebnis(s As String)

' /// Methoden
' --- Asynchrone Rückgabe der Ergebnisse
Public Sub HoleNutzer()
Dim u As Benutzer

' Schleife über alle Nutzer
For Each u In BenutzerListe
    RaiseEvent AbfrageErgebnis(u.VName)
Next
End Sub
```

Listing 3.36: Klassenmodule »Domaene3«

Anwendung der Klasse

Da der Ereigniskonsument selbst eine COM-Klasse oder VB-Klasse sein muss, ist der COM-Client diesmal in einer Klasse zu kapseln. Neben der eigentlichen Hauptroutine, die nach dem Erzeugen der Objekte und deren Füllen mit Werten die Ausgabe der Benutzerliste anstößt, ist eine Ereignisbehandlungsroutine für das Ereignis `AbfrageErgebnis()` notwendig.

```
Option Explicit
Public WithEvents d As Domaene3
Dim fertig As Boolean

' === Anwendungsbeispiel für Ereignisse
Public Sub Domaene3_Anwendung()

Dim u As Benutzer

' -- Domäne anlegen
Set d = New Domaene3
d.Name = "Sonnensystem"

' -- Benutzer anlegen
Set u = New Benutzer
u.BName = "BH"
u.VName = "Barbara Hildebrandt"
u.Geb = #1/3/1974#
```

```
' -- Benutzer in Domäne aufnehmen
d.BenutzerListe.Add u

' -- Benutzer anlegen
Set u = New Benutzer
u.BName = "HS"
u.VName = "Holger Schwichtenberg"
u.Geb = #8/1/1972#
' -- Benutzer in Domäne aufnehmen
d.BenutzerListe.Add u

say "Liste aller Nutzer in " & d.Name & ":"
' -- Start der Asynchronen Abfrage
d.HoleNutzer

End Sub

' === Ergebnisse empfangen
Private Sub d_AbfrageErgebnis(s As String)
    say s
End Sub
```

Listing 3.37: Klassenmodul »Domaene3_Client«

Diese Klasse benötigt wieder einen minimalen Client.

```
' === Anwendungsbeispiel für Ereignisse
Sub Domaene3_Anwendung()
Dim c As New Domaene3_Client
c.Domaene3_Anwendung
End Sub
```

Listing 3.38: Domaene3_Anwendung()

Ausgabe des Beispiels

Ausgabe Das Beispiel liefert folgende Ausgabe:

Liste aller Nutzer in Sonnensystem:

Barbara Hildebrandt

Holger Schwichtenberg

3.9.9 Vererbung

Einer der Hauptkritikpunkte an der Umsetzung objektorientierter Prinzipien in Visual Basic war die fehlende Vererbung.

Schnittstellenvererbung versus Implementierungsvererbung

VB 6.0 und VBA 6.0 unterstützen nur Schnittstellenvererbung, aber keine Implementierungsvererbung. Das heißt, eine Unterklasse wird gezwungen, alle Mitglieder einer Oberklasse ebenfalls zu implementieren. Die Implementierung kann aber aus der Oberklasse nicht automatisch übernommen werden, sondern die Methoden müssen in der Unterklasse reimplementiert werden (siehe auch Anhang A).

Mit Schnittstellenvererbung kann erreicht werden, dass die beiden Klassen polymorph (vgl. Anhang A) zueinander sind; es ist jedoch kein Instrument zur Wiederverwendung.

Erst Visual Basic.NET (7.0) bietet zusätzlich Implementierungsvererbung mit Hilfe des Schlüsselworts `Inherits` an.

VBS unterstützt keine Formen der Vererbung. Dieses Kapitel bezieht sich also nur auf VB6/VBA.

3.9.10 Mehrfachschnittstellen

In VB ab Version 5.0 und VBA ab Version 5.0 können Mehrfachschnittstellen für Klassen implementiert werden. Mehrfachschnittstellen basieren auf der Schnittstellenvererbung.

Visual Basic Script unterstützt weder die Definition noch die Nutzung von Mehrfachschnittstellen, da ein Schnittstellenwechsel in VBS nicht möglich ist. Dies ist aus sprachinterner Sicht kein Problem, denn wo keine Mehrfachschnittstellen definiert werden können, müssen sie auch nicht benutzt werden.



Implements

Jede Klasse besitzt im Standard nur eine Schnittstelle. Sofern die Klasse weitere Schnittstellen aufweisen soll, muss dies mit Hilfe des Statements

Implements

```
Implements Schnittstellenname
```

angezeigt werden.

Dabei kann `Schnittstellenname` entweder

- ▶ eine in einer Typbibliothek definierte COM-Klasse oder
- ▶ eine in einer Typbibliothek definierte COM-Schnittstelle oder
- ▶ eine in dem gleichen Projekt definierte (abstrakte) VB-Klasse

sein. Sofern als `Schnittstellenname` ein Klassenname angegeben wird, bezieht sich `Implements` auf die Standardschnittstelle dieser Klasse.

Mit `Implements` wird von einer Klasse verlangt, alle Member der angegebenen Schnittstelle zu reimplementieren. Visual Basic legt dann neben der Standardschnittstelle für die erbende Klasse eine zusätzliche Schnittstelle mit dem Namen der vererbenden Schnittstelle an.

Abstrakte Klassen

Da die Unterstützung von Mehrfachschnittstellen nachträglich in die Sprache aufgenommen wurde, werden Schnittstellen in Visual Basic auf kleinen Umwegen über die Definition von abstrakten Basisklassen erstellt.

Zunächst muss man für jede gewünschte Schnittstelle eine abstrakte Klasse definieren – also eine Klasse mit Attributdefinition und Methodenrümpfen, aber ohne Implementierung. Erst dann wird die eigentliche Klasse definiert, wobei mit Hilfe von `Implements` Bezug auf die zuvor fertig gestellte abstrakte Basisklasse genommen wird. `Implements` erzwingt, dass alle in der abstrakten Basisklasse definierten Attribute und Methoden hier implementiert werden müssen.

Dabei ist syntaktisch Folgendes zu beachten:

- ▶ Die Attribute müssen mit Property-Routinen implementiert werden. Einfache Attributdefinitionen funktionieren hier leider nicht.
- ▶ Der Name der abstrakten Basisklasse, deren Attribut oder Methode implementiert werden soll, muss durch einen Unterstrich getrennt vorangestellt werden (leider gibt es damit neben der Verwendung in Ereignisbehandlungsroutinen eine zweite Bedeutung dieser Notation).
- ▶ Alle implementierten Attribute und Methoden müssen als `privat` (`Private`) deklariert sein. Wenn Sie sich darüber wundern, sind Sie nicht allein, denn diese Mitglieder sollen ja dem Objektnutzer zur Verfügung stehen. In diesem Fall sind die Mitglieder aber dann trotz der `Private`-Deklaration öffentlich. Ohne Auszeichnung als `Private` wären die implementierten Member zusätzlich Teil der Standardschnittstelle der Klasse – allerdings mit ihrem unnatürlichen Namen gemäß o.g. Namenskonvention. Hier zeigt sich deutlich, dass man versucht hat, mit der bisher bekannten Menge an Schlüsselwörtern neue objektorientierte Konzepte zu realisieren, für die man besser ein neues Schlüsselwort eingeführt hätte.

Beispiel: Definition von Mehrfachschnittstellen

Das folgende Beispiel handelt von einem Multifunktionsgerät, das sowohl als Drucker als auch als Fax und Scanner verwendet werden kann. Das komplette Beispiel finden Sie auf der CD als VB-Projektdatei [CD:/Code/Kapitel3_ComPraxis/vb_mehrfachschnittstellen/geraete.vbp]. Dort sind drei abstrakte Basisklassen `IFax`, `IScanner` und `IPrinter` definiert.

Erzeugung von COM-Komponenten

Das folgende Listing zeigt die Klasse AllInOne:

```
' --- Festlegung der zu implementierenden Schnittstellen
Implements IFax
Implements IScanner
Implements IPrinter
' --- Private Variablen
Dim FaxStatus As Integer
Dim ScanStatus As Integer
' --- Implementierung der Scanner-Schnittstelle
Private Sub IScanner_ScanPage()
MsgBox "Seite wird gescannt..."
ScanStatus = 1
End Sub
Private Property Get IScanner_ScanStatus() As Integer
IScanner_ScanStatus = ScanStatus
End Property
Private Property Let IScanner_ScanStatus(ByVal vNewValue As Integer)
ScanStatus = vNewValue
End Property
' --- Implementierung der Fax-Schnittstelle
Private Sub IFax_SendFax()
MsgBox "Telefax wird gesendet..."
FaxStatus = 1
End Sub
Private Property Get IFax_FaxStatus() As Integer
IFax_FaxStatus = FaxStatus
End Property
Private Property Let IFax_FaxStatus(ByVal vNewValue As Integer)
FaxStatus = vNewValue
End Property
' --- Implementierung der Drucker-Schnittstelle
Private Sub IPrinter_PrintOut()
MsgBox "Dokument wird gedruckt!"
End Sub
```

**Definition einer
Klasse mit
mehreren
Schnittstellen**

Listing 3.39: Implementierung einer Klasse mit Mehrfachschnittstellen (nicht möglich in VBS!)

Abbildung 3.8:
Mehrfachschnittstellen-Beispiel
Multifunktionsgerät

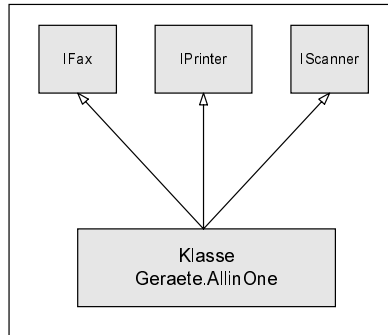
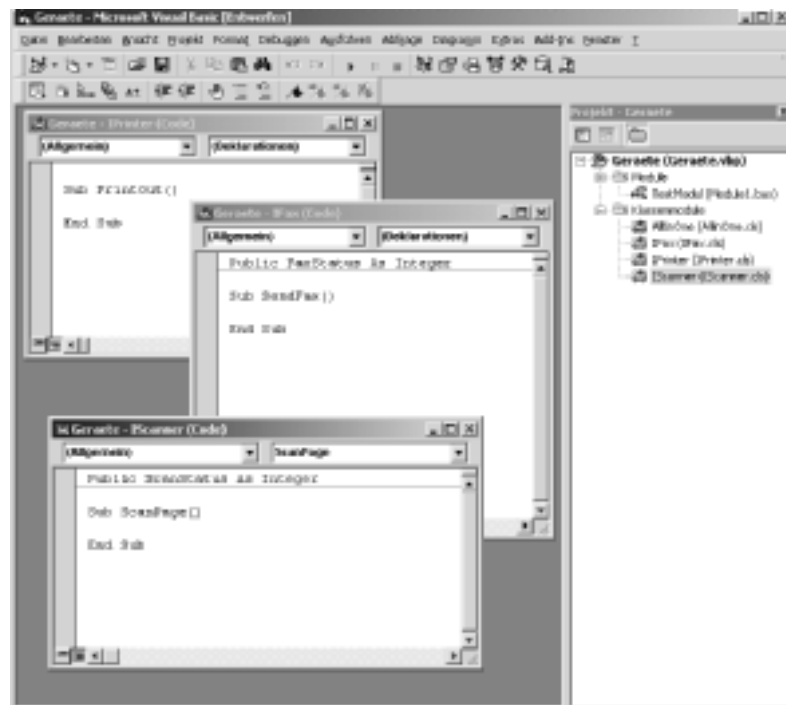


Abbildung 3.9:
Die Abbildung zeigt
die Definition der
drei Schnittstellen
»IFax«, »IPrinter«
und »IScanner«.



Verwendung von Mehrfachschnittstellen

Nutzung von Mehrfachschnittstellen

Außerdem soll hier noch gezeigt werden, wie Mehrfachschnittstellen benutzt werden. Wenn ein Objekt über mehrere Schnittstellen verfügt, dann hat VB immer die Schnittstelle im Zugriff, die dem Typ der Objektvariablen entspricht. Ein Wechsel der Schnittstelle erfolgt also mit:

```

Dim x as neueSchnittstelle
set x = y
    
```

Erzeugung von COM-Komponenten

Hierbei ist *y* ein Zeiger auf die alte Schnittstelle, und *x* muss als eine Objektvariable vom Typ der neuen Schnittstelle deklariert worden sein.

```
Sub main()  
Dim F As IFax  
Dim S As IScanner  
Dim P As IPrinter  
Dim A As App  
' Objekt instanziiieren  
Set F = New AllInOne  
' Aufruf aus Interface IFax  
F.SendFax  
MsgBox F.FaxStatus  
' Schnittstelle wechseln  
Set P = F  
P.PrintOut  
' Schnittstelle wechseln  
Set S = F  
S.ScanPage  
MsgBox S.ScanStatus  
' Fehler: Methode nicht gefunden  
S.SendFax  
' Fehler: Typen unverträglich  
Set A = F  
End Sub
```

Listing 3.40: Verwendung der im vorherigen Listing definierten Mehrfachschnittstellen (nicht möglich in VBS!)

Der obige Programmcode enthält aus didaktischen Gründen zwei Zeilen, die zu Fehler führen:

- Der vorletzte Befehl führt zu einem Fehler, weil die Methode `SendFax()` über die Schnittstelle `IScanner`, auf die `S` zeigt, nicht im Zugriff ist.
- Der letzte Befehl führt zu einem Fehler, weil die Klasse `AllInOne` nicht über eine Schnittstelle `App` verfügt.



4 WSH Runtime Library

In der Literatur zum Windows Scripting Host wird oft nur die *WSH Runtime Library* (kurz: *WSH Runtime*) vorgestellt. Die WSH Runtime ist eine wichtige, aber keineswegs die einzige Komponente für das Scripting. Ebenso wird in Darstellungen (auch in der MSDN Library) oftmals vergessen zu erwähnen, dass diese COM-Komponente keineswegs ein untrennbarer Teil des WSH ist, sondern sehr wohl wie jede andere COM-Komponente auch aus anderen COM-fähigen Umgebungen heraus verwendet werden kann. Oft werden die Begriffe *WSH Objects* oder *WSH-Komponente* synonym für *WSH Runtime Library* verwendet. Die Komponente wird in WSHOM.OCX implementiert, was für heutige COM-Komponenten eine ungewöhnliche Dateiextension ist, der Funktionalität aber nicht schadet. Die WSH Runtime Library existiert komplett unabhängig von dem Intrinsic Object WScript, das in WSCRIPT.EXE bzw. CSCRIPT.EXE implementiert wird.

WSH-Laufzeitkomponente

Name und Abkürzung	WSH Runtime Library (WSH Objects)
Name der Komponentendatei	WSHOM.OCX
Interner Name der Typbibliothek	IWSHRuntimeLibrary
Helpstring der Typbibliothek	Windows Script Host Object Model
Abweichende ProgID	WScript
Hersteller	Microsoft
Lizenzierung	Kostenloses Add-on
Besprochene Version	5.6.0.6330
NT 4 und Windows 95/98/ME	Durch Installation eines Add-ons [CD:/install/WSH]
Windows2000	Version 2.0 enthalten; Version 5.6 als Add-on installierbar [CD:/install/WSH]
Whistler (Windows XP / Windows.NET)	Version 5.6 enthalten
Dokumentation	siehe Microsoft MSDN Library [MSL00]

Tabelle 4.1:
Schnellinfo WSH
Runtime Library

Die Typbibliothek zum WSH 2.0 hatte fälschlicherweise den Helpstring »Windows Script Host Object Model 1.0«.

4.1 Objektmodelle der WSH Runtime Library

ProgIDs Die WSH Runtime enthält nicht ein, sondern zwei unabhängige Objektmodelle. Stammobjekt ist einmal WSHNetwork und zum zweiten WSHShell. WSHShell hat die ProgID »WScript.Shell«, WSHNetwork die ProgID »WScript.Network«.

Abbildung 4.1:
Objektmodell zu
WScript.Network

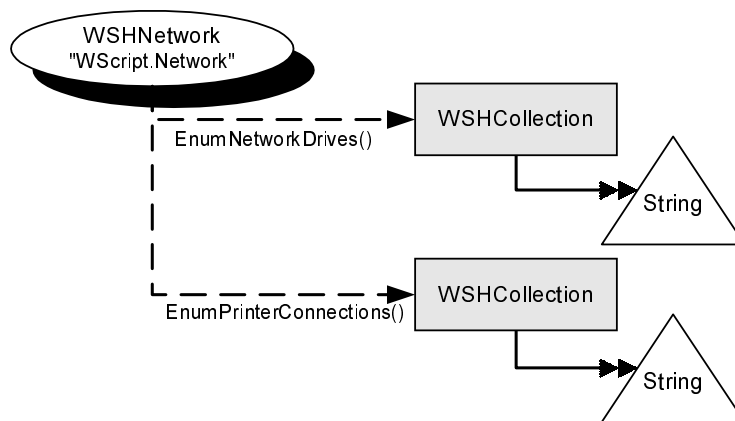
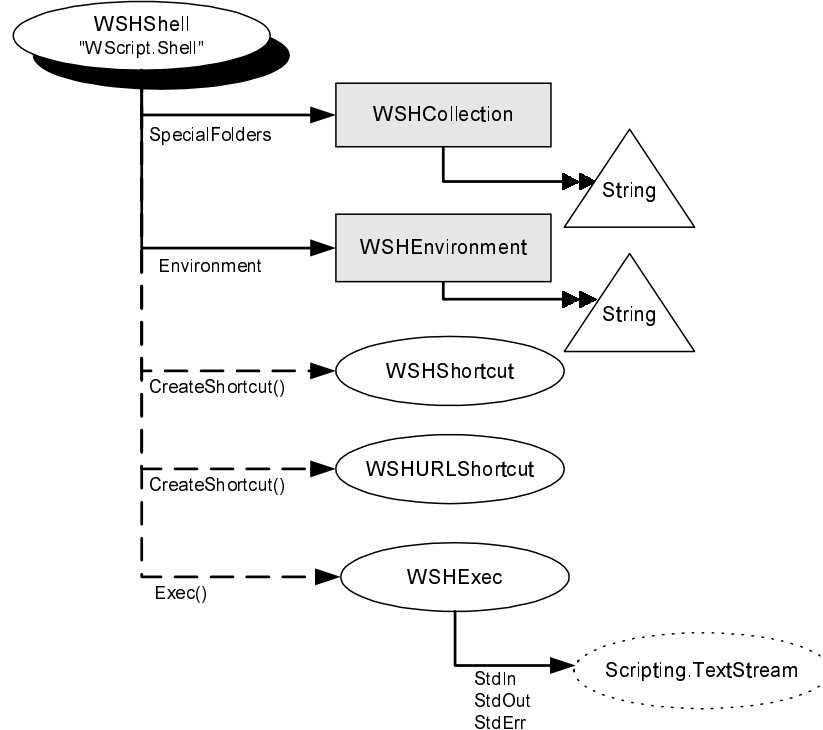


Abbildung 4.2:
Objektmodell zu
WScript.Shell



Die WSH Runtime umfasst die in der Tabelle 4.2 dargestellten Klassen. Die Klasse WSHExec ist neu seit Version 5.6.

Klassen

Klasse	Erläuterung
WSHCollection	Ein WSHCollection-Objekt ist ein Hilfsobjekt zur Verwaltung von Wertemengen, nicht wie üblich von Objekten. Die WSHCollection-Klasse ist nicht von außen instanzierbar, sondern wird von den Klassen WSHShell (im Attribut SpecialFolders) und WSHNetwork (EnumNetworkDrives, EnumPrinterConnections) verwendet.
WSHEnvironment	Dies ist eine Collection-Klasse mit mehr Ähnlichkeit zu WSHCollection als zu einer richtigen Collection-Klasse. Sie speichert Umgebungsvariablen.
WSHShell	Die WSHShell-Klasse ist inzwischen eine Mischung sehr verschiedener Funktionalitäten: Zugriff auf Umgebungsvariablen, Schreiben ins Ereignisprotokoll, Lesen und Schreiben der Registry, Senden von Tastendrücken an Fenster, Erzeugen und Verändern von Verknüpfungen im Dateisystem, Zugriff auf Spezialordner und Ausgabe von Dialogboxen.
WSHNetwork	Mit dieser Klasse wird der Zugriff auf Netzwerkverbindungen und Drucker möglich. Sie ist daher bestens dafür geeignet, die nötigen Verbindungen beim Anmelden eines Benutzers vorzunehmen. WSHNetwork gehört zu den Klassen, von denen eigene Instanzen angelegt werden können.
WSHShortcut	Ein WSHShortcut-Objekt repräsentiert eine Datei- oder Ordnerverknüpfung. Ein Objekt dieser Klasse wird durch die Methode WSHShell.CreateShortcut() erzeugt.
WSHURLShortcut	Ein WSHURLShortcut-Objekt repräsentiert eine spezielle Verknüpfung zu einem Uniform Resource Locator (URL). Ein Objekt dieser Klasse wird durch die Methode WSHShell.CreateShortcut() erzeugt.
WSHExec	Ein WSHExec-Objekt ist das Ergebnis der Methode Exec() auf einem WSHShell-Objekt.

Tabelle 4.2:
Klassen der WSH
Runtime Library

Kuriose Typbibliothek in Version 2.0

In der Version 2.0 der Bibliothek hatte Microsoft alle eigenen Namenskonventionen gebrochen, indem den Klassen Namen der Form I[Name]_Class gegeben wurden. Anstatt die Klasse WSHShell zu nennen, heißt sie in der Typbibliothek IWSHShell_Class. Auch die Komponente selbst hat mit IWSHRuntime Library in der Typbibliothek einen ungewöhnlichen Namen.

Abbildung 4.3:
Ansicht der WSH
2.0-Typbibliothek im
VB6-Objektkatalog



**Verwirrende
Typbibliothek**

In der Typbibliothek erscheinen außer den genannten Klassen auch noch IWSHShell und IWSHNetwork. Dies sind keine Klassen, sondern zusätzliche Schnittstellen der Klassen IWSHShell_Class und IWSHShell_Network. Diese Schnittstellen sind die Schnittstellen der Klassen aus WSH Version 1.0, die aus Kompatibilitätsgründen übernommen wurden. Sie sehen hier ein gutes Beispiel für Versionierung und auch dafür, wie der Microsoft Objektkatalog (siehe Kapitel 2) durch eine eingeschränkte Sicht den Benutzer verwirren kann. Es wäre viel deutlicher, wenn der Objektkatalog anzeigen würde, dass die Klasse IWSHShell_Class zwei Schnittstellen IWSHShell und IWSHShell2 besitzt, anstatt die eine Schnittstelle zur Klasse zu erheben und die andere ganz zu verbergen.

Tabelle 4.3:
Klassennamen in
Version 5.6 versus
Version 2.0

Klassenname in der Typbibliothek zur WSH Runtime 5.6	Klassenname in der Typbibliothek zur WSH Runtime 2.0
WSHCollection	IWSHCollection_Class
WSHEnvironment	IWSHEnvironment_Class
WSHShell	IWSHShell_Class
WSHNetwork	IWSHNetwork_Class
WSHShortcut	IWSHShortcut_Class
WSHURLShortcut	IWSHURLShortcut_Class
WSHExec	nicht verfügbar

Typbibliothek der WSH Runtime Version 5.6

Die Klassennamen sind in der Typbibliothek zur WSH Runtime 5.6 richtig, die Typbibliothek selbst heißt immer noch *IWSHRuntime Library*. Die Typbiblio-

thek enthält nun auch die kompletten Typinformationen der Klassen der *Scripting Runtime Library*.



Abbildung 4.4:
Ansicht der WSH
5.6-Typbibliothek im
VB6-Objektkatalog

Collections in der WSH Runtime

Die WSH Runtime Library ist Microsofts Meisterstück hinsichtlich der inkonsistenten Implementierung von Collections. Anstatt die standardisierten Collections zu verwenden, führt Microsoft zwei neue Typen von Collections ein, die auch zueinander inkonsistent sind. Hier scheint eine »heiße Nadel« als Chefprogrammierer beschäftigt gewesen zu sein. Der neue Grundtyp ist dabei die `WSHCollection`: eine Collection, die keine Objekte, sondern String-Werte enthält. Ein `WSHCollection`-Objekt verfügt über drei Attribute:

WSHCollection

- **Item:** Ermöglicht den Zugriff auf ein Element der Collection über einen numerischen – in einigen Fällen auch alphanumerischen – Wert. Da **Item** das Standardattribut ist, kann es weggelassen werden.

Attribute

```
wert = WSHCollection.Item(Index)
WSHCollection(Index) = Wert
```

Bitte beachten Sie, dass ein Zugriff der Form

```
wert = WSHCollection.Item(Index).Value
```

jedoch falsch ist, weil die Elemente der Collection keine Objekte sind.

- Count: Anzahl der Elemente in der Collection
- Length: Äquivalent zu Count. In der Microsoft-Dokumentation heißt es, dieses Attribut sei »aus Kompatibilitätsgründen zu JScript« vorhanden. Natürlich kann JScript das Attribut Count nutzen, es ist einfach nur

üblich, im JScript das Attribut `Length` zu nennen. Dem kritischen Microsoft-Nutzer stellt sich die Frage, warum Microsoft ausgerechnet an dieser Stelle so viel Wert auf Kompatibilität legt.

Weitere Features Die Enumeration mit `For...Each` wird unterstützt, jedoch – wie Sie an den einzelnen Collections sehen werden – mit zum Teil sehr unterschiedlichen Ergebnissen. Von der `WSHCollection` abgeleitet ist die Collection-Klasse `WSHEnvironment`, die zusätzlich die Methode `Remove()` unterstützt. Eine `Add()`-Methode fehlt immer. Das Hinzufügen wird in jedem Fall anders gehandhabt. Ein alphanumerischer Index existiert bei den Collections `WSHShell.Environment` und `WSHShell.SpecialFolders`. Ein numerischer Index existiert bei `WSHNetwork.EnumPrinterConnections` und `WSHNetwork.EnumNetworkDrives`.

Verfügbarkeit von Elementen in verschiedenen Versionen

Die folgende Tabelle enthält alle Klassennamen, Attributnamen und Methodennamen und zeigt an, in welcher Version der WSH-Laufzeitbibliothek sie eingeführt wurden. In allen späteren Versionen sind die Elemente dann auch verfügbar, weil keine Elemente entfernt wurden. Die Liste ist alphabetisch sortiert. Sofern ein Mitglied in mehreren Klassen vorkommt, ist der Klassenname in Klammern genannt.

Tabelle 4.4:
Unterschiede
zwischen den
WSH-Versionen

Element	Typ	WSH 1.0	WSH 2.0	WSH 5.6
AddWindowsPrinterConnection	Methode		x	
AppActivate	Methode		x	
Arguments	Attribut	x		
AtEndOfLine	Attribut		x	
AtEndOfStream	Attribut		x	
Character	Attribut			x
Close	Methode		x	
Column	Attribut		x	
ComputerName	Attribut	x		
ConnectObject	Methode		x	
Count	Methode	x		
CreateObject	Methode	x		
CreateScript	Methode			x
CreateShortcut	Methode	x		
CurrentDirectory	Attribut			x
Description	Attribut	x		
Description (WSHRemote)	Attribut			x
DisconnectObject	Methode	x		

Objektmodelle der WSH Runtime Library

Element	Typ	WSH 1.0	WSH 2.0	WSH 5.6
Echo	Methode	x		
EnumNetworkDrives	Methode	x		
EnumPrinterConnections	Methode	x		
Environment	Attribut	x		
Error (WSHRemote)	Attribut			x
Exec	Methode			x
Execute	Methode			x
Exists	Methode			x
ExitCode	Attribut			x
ExpandEnvironmentStrings	Methode	x		
FullName	Attribut	x		
GetObject	Methode	x		
GetResource	Methode		x	
HotKey	Attribut	x		
IconLocation	Attribut	x		
Item	Attribut	x		
Item (WSHNamed)	Attribut			x
Item (WSHUnnamed)	Attribut			x
Length	Attribut	x		
Line	Attribut		x	
Line (WSHRemote)	Attribut			x
LogEvent	Methode		x	
MapNetworkDrive	Methode	x		
Name	Attribut	x		
Number	Attribut			x
Path	Attribut	x		
Popup	Methode	x		
ProcessID	Attribut			x
Quit	Methode	x		
Read	Methode		x	
ReadAll	Methode		x	
ReadLine	Methode		x	
RegDelete	Methode	x		
RegRead	Methode	x		

Tabelle 4.4:
Unterschiede
zwischen den
WSH-Versionen
(Forts.)

WSH Runtime Library

Tabelle 4.4:
Unterschiede
zwischen den
WSH-Versionen
(Forts.)

Element	Typ	WSH 1.0	WSH 2.0	WSH 5.6
RegWrite	Methode	x		
Remove	Methode	x		
RemoveNetworkDrive	Methode	x		
RemovePrinterConnection	Methode	x		
Run	Methode	x		
Save	Methode	x		
ScriptFullName	Attribut	x		
ScriptName	Attribut	x		
SendKeys	Methode		x	
SetDefaultPrinter	Methode	x		
ShowUsage	Methode			x
Skip	Methode		x	
SkipLine	Methode		x	
Sleep	Methode		x	
Source	Attribut			x
SourceText	Attribut			x
SpecialFolders	Attribut	x		
Status (WSHRemote)	Attribut			x
Status (WSHScriptExec)	Attribut			x
StdErr	Attribut		x	
StdErr (WSHScriptExec)	Attribut			x
StdIn	Attribut		x	
StdIn (WSHScriptExec)	Attribut			x
StdOut	Attribut		x	
StdOut (WSHScriptExec)	Attribut			x
TargetPath	Attribut	x		
Terminate (WSHScriptExec)	Methode			x
UserDomain	Attribut	x		
UserName	Attribut	x		
Version	Attribut	x		
WindowStyle	Attribut	x		
WorkingDirectory	Attribut	x		
Write	Methode		x	
WriteBlankLines	Methode		x	

Element	Typ	WSH 1.0	WSH 2.0	WSH 5.6
WriteLine	Methode		x	
WScript	Objekt	x		
WSHArguments	Objekt	x		
WSHController	Objekt			x
WSHEnvironment	Objekt	x		
WSHNamed	Objekt			x
WSHNetwork	Objekt	x		
WSHRemote	Objekt			x
WSHRemoteError	Objekt			x
WSHScriptExec	Objekt			x
WSHShell	Objekt	x		
WSHShortcut	Objekt	x		
WSHSpecialFolders	Objekt	x		
WSHUnnamed	Objekt			x
WSHUrlShortcut	Objekt	x		

Tabelle 4.4:
Unterschiede
zwischen den
WSH-Versionen
(Forts.)

4.2 WSHNetwork und untergeordnete Klassen

Mit der Klasse `WSHNetwork` wird der Zugriff auf Netzwerk- und Druckerverbindungen möglich. Sie ist daher dazu geeignet, die nötigen Verbindungen beim Anmelden eines Benutzers vorzunehmen.

WSHNetwork

Informationen über den Computer und den Benutzer

`WSHNetwork` bietet drei Attribute an, mit denen Sie Informationen über den Computer und den angemeldeten Benutzer erhalten:

- ▶ **ComputerName:** Diese Eigenschaft liefert den NetBIOS-Namen des Computersystems als Zeichenkette zurück.
- ▶ **UserDomain:** Diese Eigenschaft enthält den Namen der Anmeldedomäne des Benutzers.
- ▶ **UserName:** Diese Eigenschaft enthält den Anmeldenamen des Benutzers.

Computername

Domäne

Benutzername

```
Set WSHNetwork = CreateObject("WScript.Network")
say "Dieser Computer heißt:", WSHNetwork.ComputerName
say "Der angemeldete Benutzer hat sich an folgender Domain
angemeldet:", WSHNetwork.UserDomain
```

```
say "Der angemeldete Benutzer hat diesen Benutzernamen:", _
WSHNetwork.UserName
```

Listing 4.1: Informationen via WSHNetwork [WSH_networkinfos.wsf]

4.2.1 Druckerverbindungen verwalten

Drucker Mit WSHNetwork können Sie Folgendes tun:

- ▶ die eingerichteten Drucker auflisten (die Methode `EnumPrinterConnections()` liefert einen Zeiger auf eine `WSHCollection`)
- ▶ neue Drucker einrichten (Methoden `AddPrinterConnection()` und `Add WindowsPrinterConnection()`)
- ▶ eine Druckereinrichtung löschen (Methode `RemovePrinterConnection()`)
- ▶ den Standarddrucker einstellen (Methode `SetDefaultPrinter()`)

Druckerverbindungen auflisten

EnumPrinterConnections()

`EnumPrinterConnections()` liefert ein `WSHCollection`-Objekt, in dem abwechselnd die Druckerports und die Druckernamen enthalten sind. Dies führt zu einer vergleichsweise komplizierten Iteration. Die Definition einer eigenen Klasse `WSHPrinter` und der Einsatz einer echten `Collection` hätten die Verwendung intuitiver gemacht und zudem die Möglichkeit geboten, weitere druckerbezogene Attribute und Methoden zu implementieren.



Um die korrekte Anzahl der Druckerverbindungen zu ermitteln, muss der Wert der `Count`-Eigenschaft der `WSHCollection` durch zwei geteilt werden.

```
Dim WSHNetwork ' As IWSHNetwork_Class
Dim DruckerAnzahl ' As Integer
Dim DruckerListe ' As IWSHCollection_Class
Dim i ' As Integer
Set WSHNetwork = CreateObject("WScript.Network")
Set DruckerListe = WSHNetwork.EnumPrinterConnections
' -- Collection enthält abwechselnd Port und Druckername!
DruckerAnzahl = DruckerListe.Count / 2
If DruckerAnzahl = 0 Then
    say "Es gibt keine Druckerverbindungen!"
Else
    say "Anzahl der eingerichteten Netzdrucker: " & DruckerAnzahl
    For i = 0 To DruckerListe.Count - 1 Step 2
        say DruckerListe(i) & " " & DruckerListe(i + 1)
    Next
End If
Set WSHNetwork = Nothing
```

Listing 4.2: Liste der eingerichteten Drucker [WSH_listPrinters.wsf]

Druckerverbindungen verändern

AddWindowsPrinterConnection verbindet einen Drucker, so dass dieser in der Systemsteuerung als Drucker erscheint. Mit AddPrinterConnection() wird dagegen ein Drucker einem Druckeranschluss zugeordnet, so dass dieser von DOS aus nutzbar ist. Dazu werden der Pfad des Druckers und ein Port benötigt. Optional ist die Angabe von Benutzername und Passwort für die Verbindung sowie die Speicherung der Verbindung in dem Benutzerprofil durch Angabe von True als drittem Parameter.

AddPrinter-Connection

```
AddWindowsPrinterConnection Druckerpfad  
AddPrinterConnection Port,Druckerpfad,[ProfilUpdate],  
[Benutzer],[Password]
```

Unter Windows 95/98/ME muss der Druckertreiber vor dem Aufruf von AddWindowsPrinterConnection() schon installiert sein.



Verbindung aufheben

Mit RemovePrinterConnection() wird eine Verknüpfung zu einem Netzdrucker wieder aufgehoben. Mit dem zweiten Parameter kann erreicht werden, dass die Verbindung trotz aktueller Verwendung beendet wird.

RemovePrinter-Connection()

```
RemovePrinterConnection Pfad, [TrotzBenutzung], [ProfilUpdate]
```

Neben der Angabe des Verknüpfungsnamens kann auch angegeben werden, ob die Ressource trotz aktueller Benutzung entfernt werden soll. Ein dritter Parameter bestimmt, ob die Veränderung in das aktuelle Benutzerprofil übernommen wird.

Standarddrucker setzen

SetDefaultPrinter() erwartet als einzigen Parameter einen Druckernamen, wie er in der durch EnumPrinterConnections() zurückgelieferten WSHCollection enthalten ist.

SetDefault-Printer()

4.2.2 Netzlaufwerke verwalten

Hier gibt es die Methoden EnumNetworkDrives(), MapNetworkDrive() und RemoveNetworkDrive(), die wie die Methoden zur Druckereinrichtung arbeiten. Für SetDefaultPrinter() gibt es keine analoge Methode, denn es gibt kein Standardnetzlaufwerk. In der durch EnumNetworkDrives() gelieferten WSHCollection wechseln sich Laufwerksbuchstabe und Netzwerkpfad in UNC-Form ab.

Netzlaufwerke

Liste der verbundenen Netzlaufwerke

Das folgende Skript gibt eine Liste aller verbundenen Netzwerke aus:

```
Dim WSHNetwork ' As IWSHNetwork_Class
Dim LaufwerksAnzahl ' As Integer
Dim LaufwerksListe ' As IWSHCollection_Class
Dim i ' As Integer
Set WSHNetwork = CreateObject("WScript.Network")
Set LaufwerksListe = WSHNetwork.EnumNetworkDrives
' -- Collection enthält abwechselnd Laufwerksbuchstabe und UNC-Pfad
LaufwerksAnzahl = LaufwerksListe.Count / 2
If LaufwerksAnzahl = 0 Then
    say "Es gibt keine Netzlaufwerke!"
Else
    say "Anzahl der eingerichteten Netzlaufwerke: " & LaufwerksAnzahl
    For i = 0 To LaufwerksListe.Count - 1 Step 2
        say LaufwerksListe(i) & " " & LaufwerksListe(i + 1)
    Next
End If
Set WSHNetwork = Nothing
```

Listing 4.3: Liste der Netzlaufwerke [WSH_listnetworkdrives.wsf]

Laufwerke verbinden

Das folgende Beispiel verbindet die Standardfreigabe »c\$« und die Freigabe von *Daten* auf dem Rechner *Sonne* mit den Laufwerksbuchstaben *S:* und *T:* und löst diese Verbindung danach wieder.

```
Dim WSHNetwork ' As IWSHNetwork_Class
Set WSHNetwork = CreateObject("WScript.Network")
list_networkdrives
WSHNetwork.MapNetworkDrive "S:", "\\sonne\c$", True, "HS", "egal"
WSHNetwork.MapNetworkDrive "T:", "\\sonne\Daten", True, "HS", "egal"
list_networkdrives
' Löschen; 1. True=TrotzBenutzung und 2.= mit ProfilUpdate
WSHNetwork.RemoveNetworkDrive "S:", True, True
WSHNetwork.RemoveNetworkDrive "T:", True, True
list_networkdrives
End Sub
```

Listing 4.4: Demo für die Arbeit mit Netzlaufwerkszuordnungen [WSH_networkdrives.wsf]



Wenn der Laufwerksbuchstabe bereits belegt ist, erscheint die Fehlermeldung »Der lokale Gerätenamen wird bereits verwendet« oder »Es wurde versucht, eine bereits gespeicherte Verbindung zu einem Gerät zu speichern.«.

4.3 WSHShell und untergeordnete Klassen

Die WSHShell-Klasse ist eine bunte Sammlung verschiedener Funktionen, die hauptsächlich in Zusammenhang mit der Benutzeroberfläche, den Umgebungsvariablen, der Registry und dem Ereignisprotokoll stehen.

WSHShell

4.3.1 Aktuelles Arbeitsverzeichnis

Neu in WSH 5.6 ist das Attribut `CurrentDirectory`, das das aktuelle Arbeitsverzeichnis zurückliefert. Das Attribut ist auch beschreibbar, um das Verzeichnis zu wechseln.

CurrentDirectory

```
Dim WSHShell
Set WSHShell = CreateObject("WScript.Shell")
say WSHShell.CurrentDirectory
WSHShell.CurrentDirectory = "c:\"
say WSHShell.CurrentDirectory
```

*Listing 4.5: Ausgabe des aktuellen Arbeitsverzeichnisses
[WSH_workingDir.wsf]*

4.3.2 Zeitgesteuerte Dialogboxen

Die Methode `Popup()` aus der WSHShell-Klasse stellt ein Dialogfenster zur Verfügung, das im Gegensatz zur Visual Basic-Funktion `MsgBox()` nach einer bestimmten Wartezeit automatisch abgebrochen werden kann. Es werden alle Optionen von `MsgBox()` unterstützt. Als zusätzlichen Rückgabewert liefert `Popup()` `-1`, wenn die Dialogbox wegen Zeitüberschreitung beendet wurde.

Popup()

```
Ergebnis = WSHShell.Popup(strText [, Wartezeit], [Titel], [Typ])
```

Der folgende Programmcode kann am Anfang eines Skripts stehen, dessen Ausführung kritisch ist. Wenn der Benutzer nicht innerhalb von zehn Sekunden nach Aufruf des Skripts OK anklickt, bricht das Skript ab, ohne die weiteren Befehle auszuführen.

```
Set WSHShell = CreateObject("WScript.Shell")
' Meldungsfenster mit Frage, maximal 10 Sekunden lang zeigen
Ergebnis = WSHShell.Popup("Wollen Sie dieses Skript starten?", _
10, "Wichtige Frage", 48 + 4)
' Keine Antwort innerhalb von 10 Sekunden
If Ergebnis = -1 Then
    say "Sie haben sich nicht entschieden! Das Skript bricht ab!"
    WScript.Quit
```

```

End If
' Antwort NEIN
If Ergebnis = 7 Then
    say "Sie haben Nein gedrückt! Das Skript startet also nicht!"
    WScript.Quit
End If
' Antwort JA
If Ergebnis = 6 Then
    say "Skript startet..."
End If

```

Listing 4.6: Dialogbox mit Zeitbegrenzung [WSH_nachfrage.wsf]

4.3.3 Zugriff auf die Registry

Registry-Funktionen

WSHShell bietet drei rudimentäre Funktionen zur Arbeit mit der Registry:

- ▶ RegWrite(Pfad,Wert[,Typ]): Erzeugung von Schlüsseln und Schreiben von Registry-Einträgen
- ▶ RegRead(Pfad): Mit dieser Methode können Registry-Einträge ausgelesen werden.
- ▶ RegDelete(Pfad): Mit dieser Methode können Einträge aus der Registry entfernt werden. Als Argument wird ein Schlüssel oder ein einzelner Eintrag angegeben. Wenn das Argument mit dem Backslash (»\«) endet, wird der ganze Schlüssel entfernt, sonst nur der Wert.

Schlüssel, Einträge und Werte Wertname

Registry-Begriffe

Innerhalb der Registry gibt es eine kleine Begriffsverwirrung. Die Registry besteht aus einer Liste von Registrierungsschlüsseln, die wiederum andere Schlüssel enthalten können. Ein Schlüssel kann neben Unterschlüsseln auch Werte enthalten. Die Registry arbeitet mit so genannten *benannten Werten*, d.h., jeder einzelne Wert hat einen Namen, um ihn von anderen Werten innerhalb desselben Schlüssels unterscheiden zu können. Ein Eintrag in der Registry hat die Form eines Attribut-Wert-Paares, z.B. Build="1085.0005". Leider wird mit dem Begriff *Registry-Wert* in der Literatur einerseits das gesamte Attribut-Wert-Paar bezeichnet, andererseits aber auch der eigentliche Wert (im obigen Beispiel "1085.0005").

In diesem Buch wird die letztere Bedeutung von Wert verwendet. Der erste Teil des Ausdrucks wird als *Wertname* bezeichnet, der gesamte Ausdruck als Registry-Eintrag.

Jeder Schlüssel hat einen Standardwert. Dieser hat keinen Wertnamen, sondern wird über den Schlüsselnamen angesprochen.

Die Registry-Methoden erwarten einen kompletten Pfad zu einem Schlüssel oder einem Wert. Ein Pfad zu einem Schlüssel muss auf einen Backslash enden («\»). Für einige Registry-Wurzelschlüssel (*Hives*) gibt es Abkürzungen:

Hives

- ▶ HKCU = HKEY_CURRENT_USER
- ▶ HKLM = HKEY_LOCAL_MACHINE
- ▶ HKCR = HKEY_CLASSES_ROOT

Beispiel für einen Pfad zu einem Schlüssel:

Beispiele

HKEY_LOCAL_MACHINE\Software\IT-Visions\

Beispiel für einen Pfad zu einem Wert:

HKLM\Software\IT-Visions\Windows-Scripting\Autor

Normalerweise legt `RegWrite()` Wertenamen und Werte an. Wird jedoch statt eines Wertenamens ein Schlüsselname angegeben (also ein String, der mit einem Backslash endet), so wird der Standardwert eines Schlüssels gesetzt. In beiden Fällen werden nicht existierende Schlüssel angelegt, und zwar auch dann, wenn mehrere Ebenen in dem angegebenen Schlüssel nicht existieren.

Standardwert

Die folgenden Datentypen der Registry werden unterstützt: `REG_SZ`, `REG_EXPAND_SZ`, `REG_DWORD` und `REG_BINARY`. Diese Datentypen können als Zeichenketten (sie sind keine Konstanten, die Zahlen repräsentieren!) als dritter Parameter übergeben werden. Wird ein anderer Datentyp als Argument übergeben, gibt die Funktion den Fehler »Type Mismatch« zurück.

Datentypen

Die WSH-Registry-Funktionen sind tolerant gegenüber Datentypfehlern, sofern eine automatische Konvertierung möglich ist. Aber Achtung: Während "1", "REG_DWORD" den gewünschten Wert 1 ergibt, steht nach "1.0", "REG_DWORD" eine 10 in der Registry. Binärwerte müssen in Form eines Array of Variant übergeben werden.



```
Dim WSHShell 'As IWSHRuntimeLibrary.IWSHShell_Class
Const firmenkey = "HKEY_LOCAL_MACHINE\Software\IT-Visions\"
' Objekt instanziiieren
Set WSHShell = CreateObject("WScript.Shell")
' Wert eintragen
WSHShell.RegWrite firmenkey & "Windows-Scripting\Autor", _
"Holger Schwichtenberg"
WSHShell.RegWrite firmenkey & "Windows-Scripting\Version", _
"1", "REG_DWORD"
WSHShell.RegWrite firmenkey & "Windows-Scripting\Test\TestWert", _
"test", "REG_SZ"
' Das wäre falsch: Type Mismatch
' WSHShell.RegWrite firmenkey & "Windows-Scripting\Version", _
"Version 1.0", "REG_DWORD"
' Defaultwert eines Schlüssels setzen
```

WSH Runtime Library

```
WSHShell.RegWrite firmenkey & "Website\","www.IT-Visions.de"
' Wert auslesen
MsgBox "Der Autor heißt:" & WSHShell.RegRead(firmenkey & _
"Windows-Scripting\Autor")
' Einen Wert löschen
WSHShell.RegDelete firmenkey & "Windows-Scripting\Test\TestWert"
' kompletten Schlüssel löschen
WSHShell.RegDelete firmenkey & "Windows-Scripting\Test\"
```

Listing 4.7: Demo verschiedener Registry-Operatoren [WSH_reg1.wsf]



Es ist mit der Methode `RegDelete()` allerdings nicht möglich, Registry-Einträge rekursiv zu löschen. Das bedeutet, dass der zu löschende Schlüssel keine Unterschlüssel mehr enthalten darf, wenn er gelöscht werden soll. Wenn dies versucht wird, dann kommt es zum Fehler Nr. 8007005 »*Unable to Remove Registry Key*«.

CLSID_from_ProgID()

CLSID_from_ProgID()

Ein nützliches Anwendungsgebiet zeigt die Funktion `CLSID_from_ProgID()`: Sie ermittelt aus einer übergebenen ProgID die zugehörige CLSID. Im Fehlerfall wird eine Fehlermeldung zurückgegeben.

```
Function CLSID_from_ProgID(progid)
Dim WSHShell ' As IWSHRuntimeLibrary.IWSHShell_Class
Set WSHShell = CreateObject("WScript.Shell")
On Error Resume Next
CLSID_from_ProgID = WSHShell.RegRead("HKCR\" & progid & "\clsid\")
If Err <> 0 Then
    CLSID_from_ProgID = "Fehler: " & Err.Description
End If
End Function
```

Listing 4.8: Hilfsroutine zur Ermittlung der CLSID anhand einer ProgID

```
say "word.application = " & _
CLSID_from_ProgID("word.application")
say "quatsch.mitSoBe = " & _
CLSID_from_ProgID("quatsch.mitSoBe") ' -- Fehler
```

Listing 4.9: Test für CLSID_from_ProgID() [WSH_testeCLSID_from_ProgID.wsf]

Schwächen

Die Registry-Funktionen der `WSHShell`-Klasse haben einige Schwächen. Mit der `WSHShell`-Klasse ist es nicht möglich, alle Unterschlüssel eines Schlüssels aufzulisten. Oft sind aber die Namen der zu lesenden Schlüssel nicht bekannt. So sind z.B. Mengen von Werten häufig als Unterschlüssel gespeichert. Die WSH Runtime kann daher beispielsweise nicht alle installierten COM-Klassen auflisten. Auch kann in der Registry nicht nach Schlüssel, Wertnamen oder Werten gesucht werden. Die in Kapitel 11 vorgestellte Komponente *RegCol* ist besser!

4.3.4 Schreiben in das Anwendungsprotokoll

Seit Version 2.0 gibt es eine Methode in der WSHShell-Klasse, um Einträge in das NT-Anwendungsprotokoll vorzunehmen. Allerdings gibt es nach wie vor keine Methode, um in andere Ereignisprotokolle zu schreiben oder das EventLog auszulesen. In Kapitel 12 wird dafür die Komponente *STMAdmin* vorgestellt. Auf einem Windows-95/98/ME-System wird der Eintrag mangels EventLog in eine Datei mit dem Namen WSH.LOG im Windows-Verzeichnis geschrieben.

LogEvent()

```
WSHShell.LogEvent(Typ, Nachricht [,Ziel])
```

Typ ist einer der in der nächsten Tabelle genannten Typen. Nachricht ist ein beliebiger String. Auf NT-Systemen kann der Eintrag wahlweise in ein entferntes System erfolgen. Ziel ist ein Rechnername oder eine IP-Adresse.

Typ	Wert
0	Erfolg
1	Fehler
2	Warnung
4	INFORMATION
8	AUDIT_SUCCESS
16	AUDIT_FAILURE

Tabelle 4.5:
Eintragstypen für
Ereignisprotokolle

```
Dim WSHShell ' As IWSHRuntimeLibrary.IWSHShell_Class
Set WSHShell = CreateObject("WScript.Shell")
WSHShell.LogEvent 1, "Demo-Fehler"
```

Listing 4.10: Eintrag ins Ereignisprotokoll [WSH_log.wsf]

Das obige Skript führt auf einem deutschen Windows-98-System zu folgendem Eintrag in die WSH.LOG:

```
01.08.00 22:32:22 Fehler: Demo-Fehler
```

4.3.5 Programme ausführen

Mit der WSH Runtime 1.0 und der WSH Runtime 2.0 gab es nur eine Methode zur Ausführung von DOS- und Windows-Programmen aus einem Skript heraus: `Run()`. In WSH Runtime 5.6 gibt es eine bessere Methode: `Exec()`. Mit beiden Methoden wird ein neuer Prozess gestartet, in dem das angegebene DOS- oder Windows-Programm ausgeführt wird. Werden in dem Kommandopfad Umgebungsvariablen angegeben, werden diese vorher ausgewertet.



Mit `Run()` und `Exec()` können nur lokale Programme gestartet werden. Um entfernte Programme zu starten, besteht ab WSH 5.6 die Möglichkeit, über *WSHControllerLibrary* ein entferntes Skript zu starten, das seinerseits ein Programm startet. Remote Scripting wird in [SCH01c] beschrieben.

Run()

`Run` hat einen Pflichtparameter und zwei optionale Parameter und gibt einen *Long*-Wert zurück.

Run()

```
Function Run(Command As String, [WindowStyle], [WaitOnReturn]) As Long
```

Parameter

Die Erscheinungsform des Programmfensters kann durch den zweiten Parameter gesteuert werden. Gültig sind hier Werte von 0 bis 10 (siehe Tabelle 4.6). Der dritte Parameter gibt an, ob `Run()` auf die Beendigung des gestarteten Programms warten soll. Im Fall von `bWarten=True` liefert `Run()` den Rückgabewert des Programms.

Tabelle 4.6:
Fensterarten für die
`Run()`-Methode

Konstante	Erläuterung
0	unsichtbar
1	normale Größe, Fenster wird aktiviert (Standardeinstellung)
2	minimiert und aktiviert
3	maximiert und aktiviert
4	letzte Größe, Fenster wird nicht aktiviert
5	letzte Größe, aktiviert
6	minimiert, Aktivierung des nächsten Fensters in der Fensterliste
7	minimiert, aber nicht aktiviert
8	letzte Größe, aber nicht aktiviert
9	wie 1: Diese Option soll für Fenster angewendet werden, die vorher minimiert waren.
10	belässt den derzeitigen Zustand des Anwendungsfensters

Beispiele

Beispiele Das folgende Skript öffnet seinen eigenen Quelltext im Editor *Notepad*. Es ist nur im WSH lauffähig, da es auf das Intrinsic Object `WScript` zugreift, um den Pfad zu dem Skript zu ermitteln.

```
' === Ausführung einer EXE mit Run()
Dim WSHShell ' As IWSHRuntimeLibrary.IWSHSHELL_Class
Set WSHShell = CreateObject("WScript.Shell")
WSHShell.Run ("%windir%\notepad.exe " & WScript.ScriptFullName)
```

Listing 4.11: Ausführung einer EXE mit `Run()`
[_WSH_run_notepad.vbs]

WSHShell und untergeordnete Klassen

Auch DOS-Batch-Dateien können aufgerufen werden. Das folgende Skript wartet, bis die Batch-Datei komplett ausgeführt wurde, und meldet, ob ein Fehler aufgetreten ist.

```
Dim WSHShell ' As IWSHRuntimeLibrary.IWSHSHELL_Class
Dim e ' Ergebnis
Set WSHShell = CreateObject("WScript.Shell")
e = WSHShell.Run("d:\buch\dos\test.bat", , True)
' -- Ergebnistest
If e = 0 Then
    say "Batch-Routine erfolgreich ausgeführt!"
Else
    say "Fehler in der Batch-Routine: " & e
End If
```

Listing 4.12: Ausführung einer DOS-Batchdatei mit Run()
[WSH_run.bat.wsf]

Exec()

Wesentlich verbessert hat Microsoft in Version 5.6 die Fähigkeit des WSH, andere lokale Skripte oder andere lokale Programme zu starten. Bisher gab es dafür schon die Methode Run() in dem Objekt WSHShell. Die neue Methode Exec() bietet im Gegensatz zu Run() die Möglichkeit, den Status des erzeugten Prozesses (»läuft noch« oder »ist beendet«) abzufragen. Außerdem kann der Entwickler auf die Standardein- und -ausgabe des Kindprozesses zugreifen. Der Vaterprozess darf in die Standardeingabe des Kindprozesses schreiben und die Standardeingabe des Kindprozesses lesen. Der Vaterprozess darf den Kindprozess auch vorzeitig beenden.

Exec() liefert zu diesem Zweck ein Objekt vom Typ WSHExec zurück.

WSHExec

Function Exec(Command As String) As WSHExec

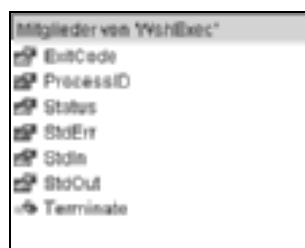


Abbildung 4.5:
Mitglieder
der Klasse
»WSHExec«

Beispiel 1: Ausgaben des Prozesses auswerten

Ping Im folgenden Beispiel wird das DOS-Programm PING.EXE aufgerufen. Die Ausgaben werden über das WSHExec-Objekt abgefangen und ausgewertet. Das Skript zählt die Anzahl der korrekten bzw. fehlerhaften Antworten.

```
' === Ausführen eines Pings und zählen, wie viele Pings einen Fehler
lieferten!
```

```
Dim WSHShell
Dim WSHExec
Dim Counter
Dim CountOK
Dim CountFehler
Dim CountAnzahl
```

```
countOK = 0
CountFehler = 0
```

```
set WSHShell = WScript.CreateObject("WScript.Shell")
```

```
' -- Ping-Prozess starten...
set WSHExec = WSHShell.Exec("ping 192.168.0.1")
```

```
WScript.Echo "----- Ausgabe von Ping.exe:"
```

```
' -- Schleife über Ausgaben
```

```
While Not WSHExec.StdOut.AtEndOfStream
```

```
    ' --- Ausgabe des Kindprozesses einlesen
```

```
    Output= WSHExec.StdOut.Readline()
```

```
    ' --- Ausgabe auswerten
```

```
    if Instr(Output,"Antwort von") > 0 then ' nur Antworten zählen!
```

```
    if Instr(Output,"Zeit") = 0 then
```

```
        countFehler = CountFehler +1
```

```
    else
```

```
        countOK = CountOK +1
```

```
    end if
```

```
    end if
```

```
    ' --- Ausgabe weiterreichen
```

```
    WScript.Echo output
```

```
Wend
```

```
CountAnzahl = CountOK + CountFehler
```

```
WScript.Echo "----- Auswertung:"
```

```
WScript.Echo countFehler & " von " & CountAnzahl & _
```

```
    " Pings waren fehlerhaft!"
```

Listing 4.13: Einsatz der Methode Exec()

[_WSH56_exec.vbs]

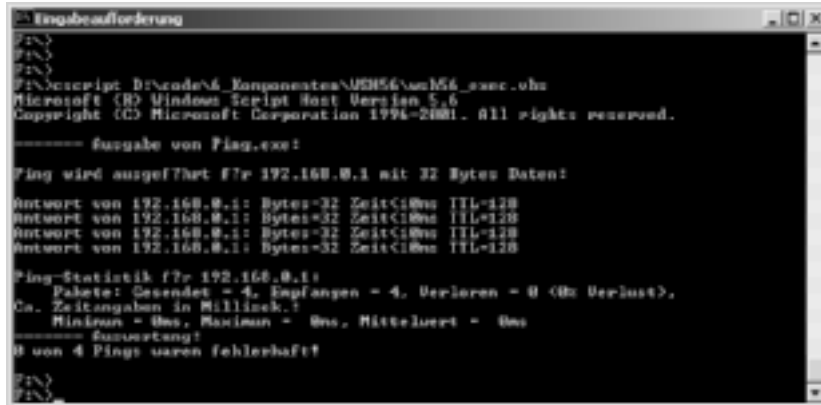


Abbildung 4.6:
Ausgabe des Skripts
_wh56_exec.vbs:
Die Umlaute wer-
den verstümmelt.

Beispiel 2: Prozess gewaltsam beenden

Das folgende Beispiel ist eine Modifikation des obigen Beispiels. Das Skript enthält eine Unterroutine Ping(), die mit Hilfe eines Aufrufs von Ping.exe prüft, ob ein Host durch genau zehn Pings mindestens einmal erreicht werden kann. Sofern der Host erreicht wird, wird PING.EXE sofort beendet. Dabei kommt die Methode Terminate() der Klasse WSHExec zum Einsatz. PING.EXE führt im Standard nur vier Pings aus. Mit der Option »-t« werden die Pings endlos ausgeführt. Ohne den Einsatz von Terminate() würde das Skript erst enden, wenn die WSH-TimeOut-Zeit erreicht ist.

Ping()-Hilfsroutine

```
' === Prüfen, ob ein Rechner erreichbar ist
dim ergebnis
Const ZIELHOST = "192.168.1.1"
ergebnis = Ping(ZIELHOST)

if ergebnis then
    WScript.echo "Zielhost " & ZIELHOST & " gefunden!"
else
    WScript.echo "Zielhost " & ZIELHOST & " nicht erreichbar!"
end if

' ### Liefert True/False, ob Host mindestens 1x erreicht werden kann!
function ping(host)
Dim WSHShell
Dim WSHExec
Dim Fehler
Dim Count
Dim Output

Ping = False
Count = 0
```

WSH Runtime Library

```
set WSHShell = WScript.CreateObject("WScript.Shell")

' -- Ping-Prozess starten...
set WSHExec = WSHShell.Exec("ping " & host)

' -- Schleife über Ausgaben
do while Not WSHExec.StdOut.AtEndOfStream
    ' --- Ausgabe des Kindprozesses einlesen
    Output= WSHExec.StdOut.Readline()
    ' Antwort?
    if Instr(Output,"Antwort von") > 0 then
' nur Antworten auswerten
        count = count + 1
        ' gefunden ?
        if Instr(Output,"Zeit") > 0 then
            Ping = True
        end if
    end if

    ' Abbruch?
    if Count = 10 or ping = True then
        WSHExec.Terminate ' sofort beenden
    exit do
    end if

Loop
End Function
```

*Listing 4.14: Nützliche Routine zum Testen, ob ein Rechner erreichbar ist
[_WSH56_exec_Ping2.vbs]*

Beispiel 3: Warten auf Prozessende

Das Attribut `Status` zeigt an, ob der Kindprozess noch läuft. 0 bedeutet, der Prozess läuft. 1 bedeutet, der Prozess ist beendet. Im dritten Beispiel wird `NOTEPAD.EXE` gestartet. Das Skript wartet solange, bis der Editor wieder geschlossen wird.

```
' === Warten, bis ein Prozess beendet ist

Dim WSHShell
Dim WSHExec

set WSHShell = WScript.CreateObject("WScript.Shell")

' --- Editor-Prozess starten...
set WSHExec = WSHShell.Exec("notepad.exe")
```

WSHShell und untergeordnete Klassen

```
' --- Warten auf das Prozessende  
While Not WSHExec.status = 1  
Wend
```

```
WScript.echo "Notepad wurde beendet!"
```

Listing 4.15: Warten auf Prozessende [*_WSH56_notepad.vbs*]

Kritik

Die Auswertung der Ausgaben einer DOS-Anwendung wird auch als Screen Scraping bezeichnet. Die gleiche Bezeichnung wird auch verwendet, wenn Computeranwendungen den Inhalt von HTML-basierten Webseiten analysieren. Die Gefahr beim Screen Scraping ist, dass die aufgerufene Anwendung bzw. Webseite das Ausgabeformat ändern und das Screen Scraping dann zu unerwarteten Ereignissen führen kann. Dieses Problem ergibt sich gerade in multinationalen Unternehmen, bei denen die Betriebssystemsprache nicht eindeutig ist. Wenn Sie dann auf die oben dokumentierte Weise die Ausgaben einer DOS-Anwendung auslesen, müssen Sie jede Sprachversion, die vorkommen kann, einzeln behandeln.



4.3.6 Fernsteuerung von Windows-Fenstern

Die WSH Runtime enthält seit Version 2.0 Methoden, um Anwendungen über simulierte Tastatureingaben fernzusteuern. Damit ist es in eingeschränktem Maße möglich, auch Anwendungen ohne COM-Schnittstellen zu automatisieren.

Fenstersteuerung

- ▶ `AppActivate(fenstername)` aktiviert ein Windows-Fenster.
- ▶ `SendKeys(tastenfolge)` sendet eine bestimmte Tastenfolge an das aktive Fenster.

Wenn bei `AppActivate()` kein Fenster gefunden wird, das exakt den angegebenen Namen enthält, so wird zunächst ein Fenster gesucht, das mit dem Namen beginnt. Ein letzter Versuch besteht darin, ein Fenster zu finden, das auf diesen Namen endet. Es wird nur das erste Fenster aktiviert, das nach dieser Regel gefunden wird.

AppActivate()

`SendKeys()` kann neben den üblichen ASCII-Zeichen auch Sondertasten übermitteln. Diese werden durch Kürzel in geschweiften Klammern umschrieben (vgl. Tabelle 4.7). `[⇧]` (Umschalttaste) wird durch `»+«` dargestellt, `[Strg]` durch `»^«` und `[Alt]` durch `»%«`. Mit runden Klammern können Tasten gruppiert werden, so dass diese als gleichzeitig gedrückt gelten. Sofern ein ASCII-Zeichen gesendet werden soll, das einem der Symbole (`~`, `+`, `^`, `&`, `,`, `)`) entspricht, muss dieses in geschweifte Klammern gesetzt werden. Die Angabe einer Zahl innerhalb der geschweiften Klammern bedeutet eine Wiederholungsanzahl (Beispiel: `{x 10}` bedeutet zehn mal `»x«`).

SendKeys()

Tabelle 4.7:
Umschreibungen
von Sondertasten
für SendKeys()

Taste	Umschreibung
	{BACKSPACE}, {BS}, oder {BKSP}
	{BREAK}
	{CAPSLOCK}
 / 	{DELETE} oder {DEL}
	{DOWN}
	{END}
	{ENTER} oder ~
	{ESC}
	{HELP}
	{HOME}
 or 	{INSERT} oder {INS}
	{LEFT}
	{NUMLOCK}
	{PGDN}
	{PGUP}
	{PRTSC}
	{RIGHT}
	{SCROLLLOCK}
	{TAB}
	{UP}
 ,  , usw.	{F1}, {F2}, usw.

Weiterhin gibt es folgende Sonderbefehle:

- ^{ESC} bedeutet das Aufklappen des Startmenüs.
- ^+{F10} bedeutet das Aufklappen des Kontextmenüs.



Das Problem beim Einsatz der `SendKeys()`-Funktion liegt darin, dass man die Tastatursteuerung sehr exakt nachbilden und sich auf die gleichbleibende Reihenfolge verlassen muss. Wenn ein Fenster geöffnet wird, muss das Skript einige Zeit warten, bis es eine Eingabe für das neue Fenster senden kann, da `SendKeys()` keinerlei Feedback empfangen kann oder abwartet. Außerdem kann der Anwender während der Abarbeitung der `SendKeys()`-Befehle das aktive Fenster wechseln und die Tastatureingaben in das falsche Fenster lenken. Die Folgen können katastrophal sein. `SendKeys()` ist also mit großer Vorsicht zu genießen.

WSHShell und untergeordnete Klassen

```
Dim WSHShell ' As IWSHRuntimeLibrary.IWSHShell_Class
Set WSHShell = CreateObject("WScript.Shell")
' - 5. Eintrag des Startmenüs auswählen
WSHShell.SendKeys "^{ESC}{UP 5}~"
```

Listing 4.16: Öffnet aus dem Startmenü den fünften Eintrag von unten [WSH_tasten.wsf]

4.3.7 Umgebungsvariablen lesen und bearbeiten

Der Zugriff auf Umgebungsvariablen erfolgt über die Collection-Klasse **WSHEnvironment**, die eine Erweiterung einer **WSHCollection** ist. Auch hier gibt es eine Zuordnung von zwei Strings, genau wie bei den Netzlaufwerken und den Druckern. Microsoft hat hier aber eine ganz andere Implementierung gefunden: Innerhalb der Collection sind die einzelnen Einträge über den Namen der Umgebungsvariablen ansprechbar und liefern den Wert zurück. Wird die Collection jedoch mit `For...Each` durchlaufen, liefert sie nicht die Namen der Umgebungsvariablen zurück, sondern einen String der Form »Umgebungsvariable=Wert«. Diese Zeichenkette muss manuell getrennt werden.

Das Attribut `Environment()` (wieder einmal ein Attribut mit Parametern!) liefert einen Zeiger auf eine **WSHEnvironment**-Collection. Da es mehrere Rubriken von Umgebungsvariablen gibt, erwartet `Environment()` einen Parameter, der den Rubriknamen angibt. Es ist aber nicht möglich, die vorhandenen Rubriken von Umgebungen aufzulisten. Es gibt vier Umgebungen: *System*, *User*, *Volatile*, *Process*. Die Routine `list_env()` listet alle Variablen einer bestimmten Rubrik auf. Dabei wird die VB-Funktion `Split()` eingesetzt, um Name und Wert zu trennen.

WSHEnvironment

List_env()

```
Sub list_env(rubrik) ' as String
Dim WSHShell ' As IWSHRuntimeLibrary.IWSHShell_Class
Dim envvar, wert ' As String
Dim objEnv 'As IWSHRuntimeLibrary.IWSHEnvironment_Class
Set WSHShell = CreateObject("WScript.Shell")
' Zugriff auf Umgebung
Set objEnv = WSHShell.Environment(rubrik)
say "Anzahl der Umgebungsvariablen: " & objEnv.Count
' Iteration über alle U-Variablen
For Each envvar In objEnv
    wert = Split(envvar, "=")
    say wert(0) & " = " & wert(1)
Next
End Sub
```

Listing 4.17: Auslesen aller Environment-Variablen einer Rubrik [list_env.wsf]

```
Dim WSHShell ' As IWSHRuntimeLibrary.IWSHShell_Class
Dim envvar, neuevar, wert ' As String
Set WSHShell = CreateObject("WScript.Shell")
```

WSH Runtime Library

```
Dim objEnv ' As IWSHRuntimeLibrary.IWSHEnvironment_Class
' - Auswahl der Umgebung
Set objEnv = WSHShell.Environment("PROCESS")
' -- Zugriff auf Path-Variable
envvar = "Path"
wert = objEnv.Item(envvar)
say envvar & " hat den Wert " & wert
' -- Ändern bzw. Anlegen einer neuen Variablen
say "-- Neue Variable..."
envvar = "ScriptingBuch"
objEnv.Item(envvar) = "d:\buch"
wert = objEnv.Item(envvar)
list_env ("PROCESS")
' -- Entfernen einer Variablen
say "-- Wieder entfernt..."
objEnv.Remove "ScriptingBuch"
list_env ("PROCESS")
```

Listing 4.18: Demo zur Arbeit mit Environment-Variablen [WSH_createEnv.wsf]

Wenn eine Umgebungsvariable nicht existiert, kommt es nicht zu einem Fehler. Als Wert wird lediglich ein Leerstring zurückgeliefert. Der Schreibzugriff ist möglich. Wenn die Variable nicht existiert, wird sie neu angelegt. WSHEnvironment bietet gegenüber WSHCollection zusätzlich die Methode Remove() an. Remove() liefert keine Fehlermeldung, wenn die Variable, die angesprochen wurde, gar nicht existierte!

```
say "Alle Umgebungsvariablen: (Stand" & Now & ")"
list_env ("SYSTEM")
list_env ("USER")
list_env ("PROCESS")
list_env ("VOLATILE")
```

Listing 4.19: Ausgabe aller Umgebungsvariablen mit Hilfe der Routine list_env() [WSH_listEnv.wsf]

ExpandEnvironmentStrings()

Eine Auflösung von Umgebungsvariablen ist auch möglich über WSHShell.ExpandEnvironmentStrings(), wobei dabei in einem String durch »%« eingeschlossene Namen als Umgebungsvariablen interpretiert und aufgelöst werden. Sofern die Umgebungsvariable nicht existiert, unterbleibt die Auflösung.

```
Dim WSHShell 'As IWSHRuntimeLibrary.IWSHShell_Class
Set WSHShell = CreateObject("WScript.Shell")
say WSHShell.ExpandEnvironmentStrings("Windows ist installiert im %windir% !")
say WSHShell.ExpandEnvironmentStrings("Sie nutzen ein %OS%-System auf einem %PROCESSOR_ARCHITECTURE%") ' Nur NT!
say WSHShell.ExpandEnvironmentStrings("%SYSTEMROOT%\System32")
```

Listing 4.20: Verwendung von ExpandEnvironmentStrings() [WSH_expandEnv.wsf]

4.3.8 Zugriff auf Sonderordner

Die Standorte der Sonderordner des Betriebssystems sind über eine `WSHCollection` ermittelbar. Sie erhalten einen Zeiger auf die Collection über das Attribut `WSHShell.SpecialFolders`.

SpecialFolders

```
Dim WSHShell ' As IWSHRuntimeLibrary.IWSHShell_Class
Set WSHShell = CreateObject("WScript.Shell")
Dim OName ' As String
say "Anzahl der Sonderordner: " & WSHShell.SpecialFolders.Count
For Each OName In WSHShell.SpecialFolders
    say OName
Next
End Sub
```

Listing 4.21: Liste aller Sonderordner [*WSH_spezFoldList.wsf*]

`Item()` ermöglicht den Zugriff auf den Pfad eines Sonderordners, sofern dessen interner Name bekannt ist. Leider gibt es keine Möglichkeit, alle verfügbaren internen Namen aufzulisten!

```
say "Hier liegen Ihre Favoriten:", _
WSHShell.SpecialFolders.Item("Favorites")
```

Die folgende Liste enthält die internen Namen der Windows-Sonderordner. Dabei sind nicht alle diese Sonderordner in jedem Betriebssystem vorhanden. Der Aufruf von `Item()` mit einem nicht vorhandenen Sonderordnernamen liefert einen Leerstring zurück. Achtung: Ein Leerstring kommt auch zurück, wenn man sich verschreibt!

AllUsersDesktop	Desktop	NetHood
AllUsersStartMenu	Favorites	PrintHood
AllUsersPrograms	Fonts	Programs
AllUsersStartup	MyDocuments	Recent
SendTo	StartMenu	Templates

Tabelle 4.8:
Liste der internen
Namen der
Windows-
Sonderordner

4.3.9 Verknüpfungen erstellen

Die Klasse `WSHShell` bietet eine Methode `CreateShortcut()`, die entweder ein Objekt der Klasse `WSHShortcut` oder `WSHURLShortcut` liefert. Die `WSHShortcut`-Klasse ermöglicht die Erstellung von Datei- und Verzeichnisverknüpfungen sowie den Zugriff auf bestehende Verknüpfungen. Ein `WSHURLShortcut`-Objekt repräsentiert eine Verknüpfung zu einem *Uniform Resource Locator* (URL).

**Verknüpfungen
verwalten**

Verknüpfungen zu Ordnern und Dateien

WSHShortcut Die Klasse bietet folgende Attribute:

- ▶ **FullName:** In dieser Eigenschaft steht der vollständige Suchpfad zum Shortcut-Objekt.
- ▶ **TargetPath:** Der Pfad zum Zielobjekt der Verknüpfung steht in dieser Eigenschaft.
- ▶ **Hotkey:** Hier steht die Definition der Abkürzungstaste zum Aufruf der Verknüpfung.
- ▶ **Description:** Hier steht eine kurze textliche Beschreibung des Verknüpfungsobjekts.
- ▶ **Arguments:** Mit dieser Eigenschaft werden einem Verweis Aufrufargumente zugeordnet.

```
WSHShortcut.Arguments = "Arg1 Arg2 Arg3"
```

Auch wenn der Name dieses Attributs es suggeriert – `WSHShortcut.Arguments` ist keine Collection und darf nicht mit `WScript.Arguments` verwechselt werden. `WSHShortcut.Arguments` ist ein einfaches Zeichenketten-Attribut, das die Kommandozeilenparameter, die in der Verknüpfung abgelegt werden sollen, in Form einer durch Leerzeichen getrennten Zeichenkette erwartet.

- ▶ **IconLocation:** Ort, an dem das Programmsymbol der Verknüpfung zu finden ist. Die Angabe erfolgt als Paar »Pfad, Index«.

```
WSHShortcut.IconLocation = "Pfad,Index"
```

- ▶ **WindowStyle:** Der Stil, mit dem das Programmfenster geöffnet wird, wenn die Aktivierung der Verknüpfung erfolgt, steht in dieser Eigenschaft. Der Stil wird durch einen Zahlwert dargestellt: 1 = normal, 3 = maximiert, 7 = minimiert.
- ▶ **WorkingDirectory:** Arbeitsverzeichnis der Verknüpfung

Speichern von Verknüpfungen `WSHShortcut` unterstützt nur eine Methode: `Save()`. Mit dieser Methode wird das Verweisobjekt an dem im `FullName` angegebenen Ort gespeichert.

Die Erzeugung neuer Instanzen eines `WSHShortcut`-Objekts erfolgt mit Hilfe der Methode `CreateShortcut()` aus der `WSHShell`-Klasse.

```
Set shortcut_obj = WSHShell.CreateShortcut(strFullName)
```

Als Parameter `strFullName` muss eine Zeichenkette angegeben werden, die den vollständigen Pfad des Ausgangspunkts der Verknüpfung enthält. Das Ziel der Verknüpfung wird dann jedoch über den Zugriff auf die Attribute des `WSHShortcut`-Objekts festgelegt. `FullName` wird beim Erzeugen automatisch auf den Namen gesetzt, der bei `CreateShortcut()` als Parameter angegeben wurde. Danach kann `FullName` noch gelesen und neu beschrieben werden.

Wenn bereits eine Verknüpfung unter diesem Namen existiert, wird sie ohne Vorwarnung überschrieben.



Die Definition eines Hotkeys besteht aus der Kombination einer Sondertaste und eines ASCII-Zeichens. Die Sondertasten werden umschrieben durch Zeichenkombinationen (z.B. *ALT+*, *CTRL+*, *SHIFT+*, *EXT*, *ESC*, *ENTER*, *TAB*, *SPACE*, *PRINT SCREEN*, *BACKSPACE*, *TAB*, *CLEAR*).

**Festlegung
eines Hotkeys**

Um die Umschreibungen zu ermitteln, legen Sie manuell eine Verknüpfung an und ordnen dieser über das Eigenschaftsfenster eine Tastenkombination manuell zu. Dann lesen Sie diese mit Hilfe eines Skripts aus!



Beispiel

Das folgende Skript erstellt eine Verknüpfung zu dem bei allen Windows-Versionen mitgelieferten einfachen Editor auf dem Desktop. Die Collection `SpecialFolders` wird benutzt, um den Pfad des Desktops zu ermitteln, in dem dann mit `CreateShortcut()` die Verknüpfung erstellt wird.

**Notepad-
Verknüpfung**

```
Set WSHShell = CreateObject("WScript.Shell")
strDesktop = WSHShell.SpecialFolders("Desktop")
Set ObjShortcut =
WSHShell.CreateShortcut(strDesktop & "\notepad.lnk")
ObjShortcut.TargetPath = "%windir%\notepad.exe"
ObjShortcut.Hotkey = "ALT+CTRL+F"
ObjShortcut.Arguments = "d:\buch\docs\test.txt"
ObjShortcut.Description = "Ich bin ein sehr einfacher Editor"
ObjShortcut.Save
say "Die Verknüpfung zu Notepad wurde erstellt."
```

Listing 4.22: Erstellen einer Verknüpfung [WSH_newshortcut.wsf]

Vorhandene Shortcuts bearbeiten

Nicht bei Microsoft dokumentiert ist die Möglichkeit, auf bestehende Verknüpfungen zuzugreifen und sie zu verändern. Wenn `CreateShortcut()` mit dem Pfad einer bestehenden Verknüpfung aufgerufen wird, enthält das zurückgegebene Objekt dessen Eigenschaften. Diese Funktionalität mit `CreateShortcut()` zu ermöglichen ist natürlich ungeschickt. Dazu wäre eine zweite Methode namens `GetShortcut()` besser gewesen (vgl. `CreateObject()` und `GetObject()`). Diese Funktionalität ist aber mit Vorsicht zu genießen, da nicht klar ist, ob es sich um eine absichtliche Implementierung oder einen ungewollten Seiteneffekt handelt.



```
Set WSHShell = CreateObject("WScript.Shell")
strDesktop = WSHShell.SpecialFolders("Desktop")
Set ObjShortcut = WSHShell.CreateShortcut(strDesktop & "\notepad.lnk")
say "FullName:" & objshortcut.FullName
```

WSH Runtime Library

```
say "TargetPath:" & objshortcut.TargetPath
say "Arguments:" & objshortcut.Arguments
say "WindowStyle:" & objshortcut.WindowStyle
say "HotKey:" & objshortcut.Hotkey
say "Description:" & objshortcut.Description
say "WorkingDirectory:" & objshortcut.WorkingDirectory
say "IconLocation:" & objshortcut.IconLocation
```

Listing 4.23: Auslesen einer bestehenden Verknüpfung [WSH_shortcutlesen.wsf]

URL-Verknüpfungen

WSHURLShortcut Wenn der Parameter bei CreateShortcut() die Endung .URL hat, ist das Ergebnis von CreateShortcut() kein WSHShortcut-Objekt, sondern ein WSHURLShortcut-Objekt. Der Ziel-URL wird in TargetPath geschrieben. Mit Hilfe einer URL-Verknüpfung können Sie komplexe URLs auf einfache Weise im Dateisystem speichern und wie eine Datei weitergeben.

```
Set WSHShell = CreateObject("WScript.Shell")
strDesktop = WSHShell.SpecialFolders("Favorites")
Set objshortcut = WSHShell.CreateShortcut(strDesktop & _
"\ Die Website zum Buch.url")
objshortcut.TargetPath = "http://www.windows-scripting.de"
objshortcut.Save
say "Dieser Link wurde gespeichert in: " & objshortcut.FullName
say "Der Link verweist auf " & objshortcut.TargetPath
```

Listing 4.24: Anlegen einer URL-Verknüpfung [WSH_urlshortcut.wsf]

5 Scripting Runtime Library

Die *Scripting Runtime Library* (zum Teil auch nur kurz *Script Runtime* genannt) ist eine bunte Mischung von Funktionen aus folgenden Bereichen:

- ▶ Zugriff auf Dateien und Ordner (komplexe Objekthierarchie mit der Stammklasse `FileSystemObject`). Dieser Teil der Scripting Runtime Library wird auch *File System Objects (FSO)* genannt.
- ▶ Zugriff auf die Textdateien sowie die Standardein- und -ausgabe (Klasse `Textstream`)
- ▶ Speicherung beliebiger Daten in verketteten Listen (Klasse `Dictionary`)
- ▶ Verschlüsselung von Skript-Quellcode (Klasse `Encoder`). Diese Klasse wird in [SCH01c] besprochen.

Die Scripting Runtime Library ist längst nicht mehr nur eine Unterstützungsbibliothek für das Scripting, sondern auch im großen Visual Basic eine häufig genutzte Komponente.

Bestandteile der Scripting Runtime Library

Name und Abkürzung	Scripting Runtime Library / File System Objects (FSO)
Name der Komponentendatei	SCRRUN.DLL
Interner Name der Typbibliothek	Scripting
Helpstring der Typbibliothek	Microsoft Scripting Runtime
Hersteller	Microsoft
Lizenzierung	Bestandteil des Betriebssystems / kostenloses Add-on
Besprochene Version	5.6.0.6330
NT4 und Windows 95/98/ME	Installation WSH 5.6 [CD:/install/wsh] oder [MSS00]
Windows2000	Version 5.1 enthalten; Update auf 5.6 durch WSH 5.6-Setup [CD:/install/wsh] oder [MSS00]
Whistler (Windows XP/ Windows.NET)	Version 5.6 enthalten
Dokumentation	siehe MSDN Library [MSL00]

Tabelle 5.1:
Schnellinfo
Scripting Runtime
Library

5.1 Die Dictionary-Klasse

Dictionaries Ein Dictionary-Objekt ist ein Container zur Speicherung einer beliebigen Art von Daten. Da jedem Eintrag nicht nur eine Zahl, sondern auch ein String als Schlüssel zugeordnet werden kann, entspricht ein Dictionary-Objekt den assoziativen Arrays in der Skriptsprache PERL bzw. einem Teil der Funktionalität dynamischer Objekte in JavaScript.

Instanziierung Die Klasse Dictionary wird von der Scripting Runtime-Komponente angeboten:

```
Set objDic = CreateObject("Scripting.Dictionary")
```

Abbildung 5.1:
Ansicht der
Dictionary-Klasse
im VB6-
Objektkatalog



Dictionary versus Collection

Vergleich zu Collections Ein Dictionary-Objekt ähnelt den Visual Basic Collections. Die Unterschiede sind folgende:

- ▶ Eine Collection kann nur Objektzeiger aufnehmen, ein Dictionary-Objekt kann Objektzeiger und jeden beliebigen Datentyp aufnehmen.
- ▶ Die Performance des Dictionary-Objekts ist wesentlich höher als die Performance einer Visual Basic Collection!
- ▶ Ein Dictionary-Objekt implementiert einige zusätzliche Methoden und Attribute.
- ▶ Die Iteration mit `For...Each` hat eine andere Semantik.

Die Dictionary-Klasse

Folgende Dictionary-Funktionalitäten entsprechen den VB-Collections:

Gemeinsame Features

- Das Einfügen neuer Elemente erfolgt mit der Methode `add()`, wobei zuerst ein Schlüssel und dann der Wert übergeben werden muss.

```
objDic.Add key, item
```

- Das Entfernen von Elementen erfolgt mit `Remove()`.

```
objDic.Remove key
```

- Das **Count-Attribut** liefert die Anzahl der enthaltenen Elemente.

```
objDic.Count
```

- Der Zugriff auf ein bestimmtes Element über den Schlüssel ist realisiert mit Hilfe von `Item()`.

```
objDic.Item(key)
```

Darüber hinaus werden folgende Funktionen nur in Dictionary-Objekten bereitgestellt:

Weitere Features der Dictionary-Klasse

- die Überprüfung, ob ein bestimmter Schlüssel existiert

```
objDic.Exists(key)
```

- das nachträgliche Ändern eines Schlüssels

```
objDic.Key(key) = newkey
```

- die Übergabe aller Schlüssel in Form eines Arrays

```
objDic.Keys
```

- die Übergabe aller Werte in Form eines Arrays

```
objDic.Items
```

- das Löschen aller Einträge mit einem Befehl

```
objDic.RemoveAll
```

- die Festlegung, ob die Schlüssel case-sensitive oder case-insensitive sind. Groß- und Kleinschreibung wird folgendermaßen unterschieden:

```
objDic.CompareMode = 0 'vbBinaryCompare
```

Und so gibt es keinen Unterschied zwischen Groß- und Kleinschreibung:

```
objDic.CompareMode = 1 'vbTextCompare
```



Ein großer Unterschied besteht bei der Verwendung von **For...Each**: Während in VB-Collections damit eine Iteration über die enthaltenen Objektzeiger erfolgt, liefert das **Dictionary**-Objekt immer die **Schlüssel** – nicht die Werte! – in Form von elementaren String-Variablen zurück. Das passiert also auch dann, wenn das **Dictionary**-Objekt Objektzeiger enthält. Die Iteration muss demnach folgendermaßen aussehen:

```
For Each key In objDic
    say key & " = " & objDic.Item(key)
Next
```

Dictionary-Beispiel

Beispiel Das Beispiel zeigt die Verwendung von `Add()`, `Remove()`, `Item()` und `Count`: Zunächst werden fünf Elemente eingefügt, davon bewusst zwei mit gleichem Text, aber unterschiedlicher Groß-/Kleinschreibung. Alle Elemente werden mit `For...Each` ausgegeben. Danach wird ein Element gelöscht. Daraufhin meldet das Skript, dass sich nun noch vier Elemente in dem **Dictionary**-Objekt befinden. Am Ende werden alle Elemente gelöscht, so dass die letzte Ausgabe »0 Elemente« ist.

```
Dim objDic ' As Scripting.Dictionary
Dim key
' --- Instanzieren
Set objDic = CreateObject("Scripting.Dictionary")
' --- Vergleichsmodus setzen: Unterscheidung ein
objDic.CompareMode = 0 'vbBinaryCompare
' --- Füllen
objDic.Add "Buch", "Windows- und BackOffice-Scripting"
objDic.Add "Autor", "Holger Schwichtenberg"
objDic.Add "Verlag", "Addison-Wesley"
objDic.Add "bemerkung", "erschieden im Jahr 2000"
objDic.Add "Bemerkung", "Ein gutes Buch"
' --- Ausgabe
For Each key In objDic
    say key & " = " & objDic.Item(key)
Next
' --- Ein Element löschen
objDic.Remove ("Bemerkung") ' Selbstbeweihräucherung raus ;-)
' --- Ausgabe
MsgBox "Jetzt sind noch " & objDic.Count & " Elemente drin!"
' --- Alle Elemente löschen
objDic.RemoveAll
' --- Ausgabe
MsgBox "Jetzt sind noch " & objDic.Count & " Elemente drin!"
```

Listing 5.1: Jonglieren mit Strings in einem Dictionary-Objekt [dic_test.wsfl]

File System Objects (FSO)

```
Buch = Windows- und BackOffice-Scripting
Autor = Holger Schwichtenberg
Verlag = Addison-Wesley
bemerkung = erschienen im Jahr 2000
Bemerkung = Ein gutes Buch
Selbstbeweihräucherung raus... ;-)
Jetzt sind noch 4 Elemente drin!
Jetzt sind noch 0 Elemente drin!
```

Listing 5.2: Ausgabe des obigen Skripts

5.2 File System Objects (FSO)

Die *File System Objects (FSO)* sind eine Ansammlung von COM-Klassen innerhalb der Scripting Runtime-Komponente. Sie bieten einen eleganten Zugang zum Windows-Dateisystem. FSO ist ein Objektmodell im Rahmen der Scripting Runtime-Komponente; seinen Namen hat es von seiner Stammklasse.

**File System
Objects**

Während das große Visual Basic und VBA seit jeher über eigene Dateisystemzugriffsfunktionen verfügen (z.B. `Dir()`, `Open()`,...), wurden diese in Visual Basic Script bewusst nicht implementiert. Das erste Anwendungsfeld für VBS war der Internet Explorer, und dort sollte aus Sicherheitsgründen kein Zugriff auf das lokale Dateisystem des Webclients möglich sein. Andernfalls hätte ein Internet Content Provider leicht ein Skript schreiben können, das den Inhalt der Festplatte des Surfers scannt und beim nächsten Seitenabruf gezielt Informationen an den Webserver überträgt.

Für andere Anwendungsgebiete des Scriptings (z.B. ASP, WSH) ist jedoch der Zugriff auf das Dateisystem ein wichtiges Feature. Mit FSO stellt Microsoft eine Komponente zur Verfügung, die einen objektbasierten Zugriff auf das Dateisystem ermöglicht und damit weit komfortabler ist als die traditionellen Zugriffsmethoden aus VB und VBA. Daher ist seit den Versionen 6.0 FSO auch die empfohlene Vorgehensweise für den Dateizugriff im großen VB und VBA.

5.2.1 Funktionsüberblick

FSO bietet folgende Möglichkeiten:

Features

- ▶ Direkter Zugriff auf einzelne Laufwerke, Ordner und Dateien
- ▶ Direkter Zugriff auf Sonderordner
- ▶ Iteration über Laufwerke und Ordner
- ▶ Zusammensetzung und Aufspaltung von Pfadangaben
- ▶ Anlegen, Verschieben, Kopieren und Löschen von Ordnern
- ▶ Verschieben, Kopieren und Löschen von Dateien jeden Typs
- ▶ Anlegen, Lesen und Beschreiben von Textdateien

- ▶ Lesen und Verändern von Laufwerks-, Ordner- und Dateieigenschaften
- ▶ Direkter Zugriff auf Sonderordner
- ▶ Zugriff auf die Standardein- bzw. -ausgabe
- ▶ Ändern von Dateiattributen
- ▶ Zugriff auf Dateilänge und Daten (DateCreated, DateLastAccessed, DateLastModified)
- ▶ Versionsinformationen von DLLs

Fehlende Features Folgende Funktionen, die in Zusammenhang mit dem Dateisystem anfallen, deckt FSO jedoch nicht ab:

- ▶ Anlegen, Lesen und Beschreiben von binären Dateien
- ▶ Suchfunktion über das Dateisystem
- ▶ Zugriff auf den Sperrstatus einer Datei
- ▶ Zugriff auf Sicherheitsinformationen
- ▶ Zugriff auf Verzeichnisfreigaben
- ▶ Zugriff auf erweiterte Dateiattribute (z.B. Autorennamen bei Word-Dokumenten)
- ▶ Zugriff auf die Kontextmenüeinträge einer Datei
- ▶ Meldung von Dateisystem-Ereignissen (neue Datei, Dateiänderung, etc.)

Andere Komponenten für das Dateisystem

Die Unterstützung binärer Dateien ist für eine kommende Version von FSO geplant. Für den Zugriff auf die Sicherheit gibt es mit ADSSECURITY.DLL inzwischen eine eigene Komponente. Den Zugriff auf Verzeichnisfreigaben ermöglichen die Komponenten *ADSI* und *WMI*.

Den Zugriff auf die Kontextmenüeinträge ermöglicht die Komponente *Shell Objects*. Diese Komponente überschneidet sich hinsichtlich der Funktionalität zum Teil mit FSO, realisiert den Zugriff auf das Dateisystem jedoch in einem etwas anderen Objektmodell.

Auch WMI ermöglicht den Zugriff auf das Dateisystem. Hier sind auch die Suche (auf Basis von WQL) und das Abfangen von Ereignissen möglich.

5.2.2 FSO-Objektmodell

FSO-Objektmodell Zentrale Klasse ist `FileSystemObject`, die das ganze Dateisystem repräsentiert und zentrale Methoden bereitstellt. Von `FileSystemObject` aus gibt es grundsätzlich zwei Wege zu den Unterelementen:

- ▶ das Entlanghangeln anhand der Objekthierarchie
- ▶ den direkten Zugriff auf ein Dateisystemelement über dessen Pfad

`FileSystemObject` ist die Stammklasse und einzige instanziiierbare Klasse im FSO-Objektmodell, so dass deren Instanziierung immer der erste Schritt sein muss. FSO besitzt keine Attribute mit elementaren Werten, sondern neben zahlreichen Methoden nur den Verweis auf vier Unterobjekte.

File System Objects (FSO)

```
DIM FSO ' As Scripting.FileSystemObject
Set fso = CreateObject("Scripting.FileSystemObject")
```

FSO ist leicht zu erlernen, weil es für jeden Windows-Nutzer sehr anschaulich ist: Es spiegelt Elemente wider, die der Anwender jeden Tag an seinem PC vor Augen hat. FSO eignet sich daher gut für Übungen zu Objekten und Collections, insbesondere auch zum Erlernen rekursiver Funktionen.

Die Laufwerke bilden eine `Collection Drives`, die einzelne `Drive`-Objekte enthält. Über das Attribut `RootFolder` gelangt man zu einem `Folder`-Objekt. Ein `Folder`-Objekt enthält zwei `Collections`. In der einen sind die Dateien enthalten (`Files`), in der anderen die Ordner. Dabei weicht hier der Attributname (`Subfolders`) von dem Klassennamen (`Folders`) ab. Eine `Folders`-`Collection` enthält wieder `Folder`-Objekte, so dass eine Rekursion in dem Objektbaum entsteht. Ein `File`-Objekt repräsentiert eine einzelne Datei beliebigen Typs.

Drives, Folders, Files

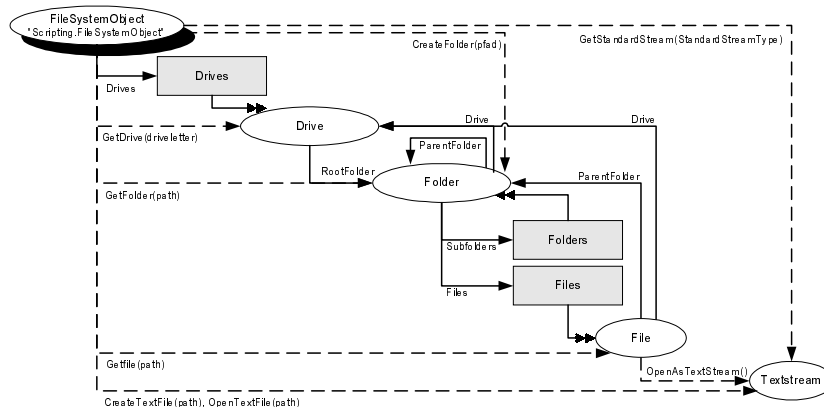


Abbildung 5.2:
Das FSO-
Objektmodell

Klasse	Erläuterung
FileSystemObject	Stammklasse, die als einzige instanzifizierbar ist. Diese Klasse dient auch der Erzeugung neuer Instanzen der anderen Klassen.
Drive	Repräsentiert ein Laufwerk
Drives	Liste aller verfügbaren Laufwerke (nicht nur Festplatten, sondern alle Arten von Laufwerken, die im Windows-Explorer angezeigt werden können), also Diskettenlaufwerk, Festplatte, CD, andere Wechselmedien und auch Laufwerksverknüpfungen
File	Repräsentiert eine Datei
Files	Liste aller Dateien in einem Ordner
Folder	Repräsentiert einen Ordner

Tabelle 5.2:
FSO-Klassen

Tabelle 5.2:
FSO-Klassen
(Forts.)

Klasse	Erläuterung
Folders	Liste aller Ordner in einem Laufwerk oder einem übergeordneten Ordner
TextStream	Repräsentiert eine Textdatei oder einen Standard-I/O-Stream

5.2.3 Lesezugriff auf das Dateisystem

Drives Das erste FSO-Skript listet alle Laufwerke auf. Wie das Beispiel zeigt, gibt es nur wenige Eigenschaften, die auf jeden Fall ausgelesen werden können (DriveLetter, DriveType, ShareName und Path). Der Zugriff auf medienabhängige Eigenschaften ist dagegen nur möglich, wenn sich auch ein lesbares Medium in dem Laufwerk befindet. Dies sollte mit `isready()` überprüft werden, bevor ein Zugriff auf die medienabhängigen Eigenschaften erfolgt.

```
Dim FSO      ' As Scripting.FileSystemObject
Dim oDrive  ' As Scripting.Drive
Set FSO = CreateObject("Scripting.FileSystemObject")
For Each oDrive In FSO.Drives
    say "--- Allgemeine Daten"
    say "DriveLetter: " & oDrive.DriveLetter
    say "DriveType: " & oDrive.DriveType
    say "ShareName: " & oDrive.ShareName
    say "Path: " & oDrive.Path
    If oDrive.IsReady Then
        say "--- Medienabhängige Daten"
        say "IsReady: " & oDrive.IsReady
        say "SerialNumber: " & oDrive.SerialNumber
        say "FileSystem: " & oDrive.FileSystem
        say "VolumeName: " & oDrive.VolumeName
        say "--- Mediengröße"
        say "TotalSize: " & oDrive.TotalSize
        say "FreeSpace: " & oDrive.FreeSpace
        say "AvailableSpace: " & oDrive.AvailableSpace
    End If
Next
```

Listing 5.3: Ausführliche Liste der Laufwerke [fso_drives.wsf]

Ordnerinhalte auflisten

Das Listing zeigt, wie die in einem bestimmten Ordner enthaltenen Dateien und Unterordner aufgelistet werden.

```
Dim FSO ' As Scripting.FileSystemObject
Dim oFolder ' As scripting.folder
Dim oSubfolder ' As scripting.folder
Dim oFile ' As scripting.file
```

File System Objects (FSO)

```
Set FSO = CreateObject("Scripting.FileSystemObject")
Set oFolder = FSO.GetFolder("d:\buch")
say "-- Dateien:"
For Each oFile In oFolder.Files
    say oFile.Name
Next
say "-- Ordner:"
For Each oSubfolder In oFolder.SubFolders
    say oSubfolder.Name
Next
```

*Listing 5.4: Das Skript listet den Inhalt eines Ordners ohne Rekursion auf.
[fso_OrdnerInhalt.wsf]*

Rekursion über Unterordner

Der kleine Olymp der FSO-Programmierung ist die Rekursion über einen ganzen Verzeichnisbaum. Dabei wird nicht nur der Inhalt eines Ordners aufgelistet, sondern – sofern der Ordner Unterordner enthält – auch der Inhalt der Unterordner und der Inhalt der Unterordner der Unterordner usw.

Listfolder()

Das Listing 5.5 ist kürzer, als man vielleicht meint. Die Rekursion steckt in der Methode `Listfolder()`, die sich immer wieder selbst aufruft. Übergeben werden der Pfad eines Ordners und eine Ebenennummer, wobei diese optional ist.

```
Dim start ' As String
Set FSO = CreateObject("Scripting.FileSystemObject")
start = InputBox("Pfad, an dem gestartet werden soll?")
listfolder start, 0

' ### Hilfsroutine: Rekursion über Ordnerinhalte
Sub listfolder(folder, ebene)
Set FSO = CreateObject("Scripting.FileSystemObject")
Set objFolder = FSO.GetFolder(folder)
say "Inhalt von " & folder & " (Ebene: " & ebene & ")"
say "-- Dateien:"
For Each objFile In objFolder.Files
    say objFile.Name
Next
say "-- Ordner:"
For Each objSubFolder In objFolder.SubFolders
    say objSubFolder.Name
    listfolder objSubFolder.Path, ebene + 1
Next
End Sub
```

Listing 5.5: Rekursive Liste von Ordnerinhalten [fso_rekInhalt.wsf]

Direkter Zugriff mit den Get-Methoden

Direkter Abstieg Der ebenenweise Abstieg im Verzeichnisbaum ist dann sinnvoll, wenn eine Datei gesucht oder eine Liste erstellt werden soll. Für den Zugriff auf eine bestimmte Datei, deren Pfad und Namen man kennt, ist der ebenenweise Abstieg vergleichbar mit dem Abstieg in Serpentin von einem Berg, anstatt den Sessellift zu benutzen. Sicherlich hat das seinen Reiz, ist aber nicht besonders effizient.

GetDrive(), GetFolder(), GetFile() Die Klasse `FileSystemObject` stellt daher drei Methoden für den direkten Zugriff auf die Dateisystemeinträge bereit:

- ▶ `GetDrive(laufwerksname)`
- ▶ `GetFolder(pfad)`
- ▶ `GetFile(pfad)`

Das folgende Listing zeigt dementsprechend drei gültige und drei ungültige Aufrufe. In den letzten drei Beispielen entspricht die Spezifikation des Pfads nicht dem Typ des Dateisystemeintrags. Tolerant ist FSO jedoch hinsichtlich dessen, ob Sie einen Laufwerksbuchstaben als einfachen Buchstaben (`D`), mit Doppelpunkt (`D:`) oder mit Doppelpunkt und Backslash (`D:\`) angeben. Grundsätzlich kann die Funktion von `GetDrive()` auch immer mit `GetFolder()` abgebildet werden. Wenn allerdings kein Medium in einem Laufwerk ist, schlägt ein `GetFolder()` auf dieses Laufwerk fehl, während `GetDrive()` keinen Fehler liefert.

```
' - gültige Aufrufe
Set x = FSO.GetDrive("D:\")
Set x = FSO.GetFolder("D:")
Set x = FSO.GetFile("D:/buch/vertrag.txt")
' - ungültige Aufrufe
'Set x = fso.GetDrive("D:/buch")
'Set x = fso.GetFolder("D")
'Set x = fso.GetFile("D:/buch/")
```

Listing 5.6: Verwendung von Pfaden in den Get-Methoden [*fso_pfade.wsf*]



FSO ist tolerant hinsichtlich der Verwendung von Slash (`»/«`) und Backslash (`»\«`) in Pfadangaben. Beide Schreibweisen werden akzeptiert, anders also als bei der DOS-Kommandozeilenebene, die dafür nur den Backslash akzeptiert und die Verwendung eines Slashes als Kommandozeilenparameter interpretiert. Alle von FSO zurückgelieferten Pfadangaben verwenden jedoch immer den korrekten Backslash.

Datei- und Ordneigenschaften

Attribute In der `File`- und in der `Folder`-Klasse gibt es im Wesentlichen die gleichen Attribute für Informationen über die Datei- bzw. Ordneigenschaften. Das folgende Skript funktioniert daher sowohl mit einem `File`- als auch mit einem `Folder`-Objekt. Bitte beachten Sie, dass die Eigenschaft `ParentFolder`

File System Objects (FSO)

nicht den Pfad, sondern den Zeiger auf ein Folder-Objekt liefert. Sie erhalten den Pfad daher über `obj.ParentFolder.Path`.

```
Dim FSO      ' As Scripting.FileSystemObject
Dim obj      ' File oder Folder
' -- FSO erzeugen
Set FSO = CreateObject("Scripting.FileSystemObject")
' -- Bindung an Datei oder Ordner
Set obj = FSO.GetFile("d:\buch\docs\test.xls")
'Set obj = FSO.GetFolder("d:\buch\docs")
' -- Ausgabe der Eigenschaften
say "Standort: " & obj.ParentFolder.Path
say "Typ: " & obj.Type
say "Größe: " & obj.Size
say "Erzeugt am: " & obj.DateCreated
say "Zuletzt geändert am: " & obj.DateLastModified
say "Zuletzt gelesen am: " & obj.DateLastAccessed
say "Attribute: " & fileatt_to_string(obj.Attributes)
```

Listing 5.7: Ausgabe von Dateiattributen [fso_Dateieigenschaften.wsf]

Das Attribut `Attributes` ist eine Intervariable, in der verschiedene Bits für die einzelnen Eigenschaften gesetzt sind. Die Methode `fileatt_to_string()` erledigt die Umwandlung.

```
Function fileatt_to_string(att)
' Att as Long
' -- Array mit Konstanten für FileAttribute
Dim arrFileAttribute(8, 2)
arrFileAttribute(0, 0) = "Normal"
arrFileAttribute(0, 1) = 0
arrFileAttribute(1, 0) = "ReadOnly"
arrFileAttribute(1, 1) = 1
arrFileAttribute(2, 0) = "Hidden"
arrFileAttribute(2, 1) = 2
arrFileAttribute(3, 0) = "System"
arrFileAttribute(3, 1) = 4
arrFileAttribute(4, 0) = "Directory"
arrFileAttribute(4, 1) = 16
arrFileAttribute(5, 0) = "Archive"
arrFileAttribute(5, 1) = 32
arrFileAttribute(6, 0) = "Alias"
arrFileAttribute(6, 1) = 1024
arrFileAttribute(7, 0) = "Compressed"
arrFileAttribute(7, 1) = 2048
' -- Umwandlung
fileatt_to_string = get_from_array_mult(att, arrFileAttribute)
End Function
```

Listing 5.8: Diese Hilfsroutine liefert die Fileattribute als String. [fileatt_to_string.wsf]

Dateiversion Neu in Version 5.1 war eine Funktion zur Ermittlung der Dateiversionen.

```
Dim FSO ' As Scripting.FileSystemObject
Dim strSystemFolder ' As String
Set FSO = CreateObject("Scripting.FileSystemObject")
strSystemFolder = FSO.GetSpecialFolder(1)
say FSO.GetFileVersion(strSystemFolder & "\scrrun.dll")
```

Listing 5.9: Mit diesem Skript ermittelt FSO selbst seine eigene Version. GetSpecialFolder(1) liefert den Pfad zum System-Verzeichnis. [fso_filevers.wsf]



GetSpecialFolder() kennt als weitere mögliche Parameter nur den Windows-Ordner (0) und den Temp-Ordner (2). GetFileVersion() liefert keine Fehlermeldung, sondern einen Leerstring zurück, wenn die Datei nicht existiert.

5.2.4 Dateisystemoperationen

Veränderungen Dateisystemoperationen wie Kopieren, Verschieben oder Löschen sind einerseits auf der Ebene des FileSystemObject, andererseits auf der Ebene der File- bzw. Folder-Klasse möglich. Sowohl die Klasse File als auch die Klasse Folder stellt die Methoden Copy(), Move() und Delete() bereit. Die FileSystemObject-Klasse bietet die Methoden CopyFile(), CopyFolder(), MoveFile(), MoveFolder(), DeleteFile() und DeleteFolder() an.

Tabelle 5.3:
Methoden für
Ordner-Operatoren

Aufgabe	Verfügbare Methoden
Ordner erstellen	FileSystemObject.CreateFolder(Pfad)
Ordner zz	Folder.Delete([Erzwingen]) oder FileSystemObject.Delete Folder(Pfad,[Erzwingen])
Ordner verschieben	Folder.Move(Ziel) oder FileSystemObject.MoveFolder(Quelle,Ziel)
Ordner kopieren	Folder.Copy(Ziel,Überschreiben) oder FileSystemObject.CopyFolder(Quelle,Ziel)

Tabelle 5.4:
Methoden für
Datei-Operatoren

Aufgabe	Verfügbare Methoden
Datei verschieben	File.Delete([Erzwingen]) oder FileSystemObject.DeleteFile(Pfad,[Erzwingen])
Datei kopieren	File.Move(Ziel) oder FileSystemObject.MoveFile(Quelle,Ziel)
Datei löschen	File.Copy(Ziel,Überschreiben) oder FileSystemObject.CopyFile(Quelle,Ziel)

File System Objects (FSO)

Operationen in der File- und Folder-Klasse

Das folgende Skript nutzt die `Delete()`-Methode der `File`-Klasse, um alle Dateien mit den Extensionen `.HTM` und `.HTML` aus `C:\TEMP` zu löschen. **Delete()**

```
Dim FSO      ' As Scripting.FileSystemObject
Dim oFile    ' As Scripting.File
Dim oFolder  ' As Scripting.Folder
' -- FSO erzeugen
Set FSO = CreateObject("Scripting.FileSystemObject")
' -- Bindung Ordner
Set oFolder = FSO.GetFolder("c:\temp")
' -- Iteration
For Each oFile In oFolder.Files
    If UCase(Right(oFile.Name, 4)) = ".HTM" _
    Or UCase(Right(oFile.Name, 5)) = ".HTML" Then ' Löschen
        say "Loesche " & oFile.Name
        oFile.Delete
    End If
Next
```

Listing 5.10: Löschen aller HTML-Dateien in c:\temp [fso_selektivesLoeschen.wsfl]

Operationen in der FileSystemObject-Klasse

In dem folgenden Skript wird eine Instanz von `FileSystemObject` verwendet, um verschiedene Dateisystemoperationen auszuführen. Ergebnis dieses Skripts ist, dass eine Kopie der Datei auf der Buch-CD mitgelieferten `TEXT.TXT` unter einem zufälligen Namen in `C:\TEMP` liegt. Die Parameter der eingesetzten Methoden bedürfen keiner näheren Erklärung, da – wie in DOS-Befehlen – der erste Parameter stets die Quelle und der zweite Parameter stets das Ziel ist. Die Methode `GetTempName()` erzeugt einen zufälligen Namen für eine temporäre Datei in 8.3-Form.

**Ordner und
Dateien kopieren,
verschieben und
löschen**

```
Dim FSO      ' As Scripting.FileSystemObject
Dim temp     ' As String
Const ziel = "d:\buch\docs_kopie"
' --- FSO erzeugen
Set FSO = CreateObject("Scripting.FileSystemObject")
If Not FSO.FolderExists("d:\buch\docs") Then
    say "Testumgebung zu diesem Buch existiert nicht!"
    Exit Sub
End If
' --- Ausführung von Dateisystemoperationen
say "Kopiere Verzeichnis..."
FSO.CopyFolder "d:\buch\docs", ziel
temp = FSO.GetTempName
say "Kopie Datei in neue temporäre Datei: " & temp
FSO.CopyFile ziel & "\test.txt", ziel & "\" & temp
say "Verschiebe Datei in nach c:\temp"
```

```
FSO.MoveFile ziel & "\" & temp, "c:\temp\"
say "Lösche Ordnerkopie"
FSO.DeleteFolder ziel
End Sub
```

Listing 5.11: Ausführung von Dateisystemoperationen mit der FSO-Stammklasse
[FSO_operationen.wsf]

5.3 Arbeit mit Textstreams

Zugriff auf Textdateien

Die Scripting Runtime-Komponente ermöglicht die Bearbeitung von Textdateien über eine so genannte `TextStream`-Klasse. Diese Klasse wird in der Regel als Teil der FSO angesehen, besitzt aber keinen statistischen Link (also einen Link über ein Attribut) zu einer anderen Klasse von FSO und hat inzwischen auch Aufgaben außerhalb des Dateisystems. Eine `TextStream`-Klasse repräsentiert eine geöffnete Textdatei. Es ist zwar möglich, binäre Dateien zu öffnen, es gibt jedoch in FSO keine Verfahren, um binäre Dateien korrekt zu verarbeiten.

Standardein- und Standardausgabe

Seit Version 5.1 der Scripting Runtime Library gibt es ein weiteres Einsatzgebiet der `Textstream`-Klasse. Auch die Standardein- und Standardausgabe werden über `Textstream`-Objekte repräsentiert. Dies macht sich auch der Windows Scripting Host in seinem Intrinsic Object `WScript` zu Nutze.



Eine Textdatei ist eine Datei, in der nur gültige ASCII-Zeichen und ASCII-Steuercodes enthalten sind, so dass eine Textdatei mit einem Texteditor lesbar und veränderbar ist. Als Alternative zum ASCII-Zeichensatz können Textdateien inzwischen auch den neueren Unicode enthalten.

Eine binäre Datei hingegen kann alle möglichen Bytewerte enthalten, auch solche, die auf dem Bildschirm nicht als Zeichen oder Formatierung darstellbar sind.

Wege zu einem Textstream

Es gibt vier Möglichkeiten, ein `Textstream`-Objekt zu erhalten:

- OpenTextFile()** 1. Mit der Methode `OpenTextFile()` auf der Ebene der `FileSystemObject` Klasse kann eine Datei direkt über ihren Pfad geöffnet werden.


```
OpenTextFile(fileName As String, [IOMode As IOMode = ForReading],  
[Create As Boolean = False], [Format As Tristate = TristateFalse])  
As TextStream
```
- CreateTextFile()** 2. Auf der Ebene der FSO-Stammklasse steht mit `CreateTextFile()` ebenso eine Möglichkeit zur Verfügung, eine neue Textdatei zu erzeugen.


```
CreateTextFile(fileName As String, [Overwrite As Boolean = True],  
[Unicode As Boolean = False]) As TextStream
```

Arbeit mit Textstreams

3. Die `File`-Klasse bietet mit `OpenAsTextStream()` eine Möglichkeit an, eine Datei, die bereits in Form eines `File`-Objekts im Zugriff ist, als Textdatei zu öffnen.

OpenAsTextStream()

```
OpenAsTextStream([IOMode As IOMode = ForReading], [Format As Tristate = TristateFalse]) As TextStream
```

4. Die Methode `GetStandardStream(Typ)` in der `FileSystemObject`-Klasse liefert ein Textstream-Objekt für die Standardeingabe (`Typ = 0`), Standardausgabe (1) und Standardfehlerausgabe (2).

GetStandardStream()

`OpenTextFile()` und `CreateTextFile()` erwarten als Pflichtparameter einen Dateinamen. Bei `OpenAsTextStream()` ist die Angabe eines Dateinamens nicht notwendig. Dieser ist bereits durch das `File`-Objekt, auf dem die Methode ausgeführt wird, spezifiziert. Die Angabe, in welchem Modus die Datei geöffnet werden soll, ist immer optional. Standard ist *ForWriting* (vgl. Tabelle 5.5).

Symbolische Konstante	Numerische Konstante	Erläuterung
<code>TristateTrue</code>	-1	Unicode-Datei
<code>TristateFalse</code>	0	ASCII-Datei
<code>TristateUseDefault</code>	-2	Verwendung der Standardeinstellung des Systems

Tabelle 5.5:
Konstanten für
Tristate

Mit dem Parameter `Create` kann bestimmt werden, ob die Datei erzeugt werden soll, wenn sie nicht existiert. Der Standardwert ist *False*: Wenn die angegebene Datei nicht existiert, kommt es zu einer Fehlermeldung. Beim Wert *True* wird die Datei erzeugt – allerdings wirklich nur dann, wenn es sie noch nicht gibt. Die Frage, ob die Datei existiert, stellt sich bei `OpenAsTextStream()` nicht, da über das `File`-Objekt ja bereits ein Zugriff darauf besteht.

Parameter

Der `IOMode` *ForAppending* funktioniert also auch dann korrekt und lässt den bisherigen Inhalt bestehen, wenn `Create` auf *True* gesetzt ist. Mit dem Parameter `Tristate` kann zwischen ASCII- und Unicode-Dateien unterschieden werden.

Während der ASCII-Zeichensatz auf einer 8-Bit-Darstellung beruht, verwendet Unicode 16 Bit. Mit Unicode können daher wesentlich mehr Sonderzeichen eindeutig dargestellt werden. Unicode-Dateien sind nicht in allen Texteditoren darstellbar. Der Microsoft Notepad unterstützt nur ASCII. Microsoft Visual InterDev kann jedoch auch Unicode-Dateien verarbeiten.



Tabelle 5.6:
Konstanten für
IOMode

Symbolische Konstante	Numerische Konstante	Erläuterung
ForReading	1	Datei lesen
ForWriting	2	Datei lesen und schreiben
ForAppending	8	An Datei anfügen

Verarbeitung von Textdateien

Features Die `TextStream`-Klasse stellt Methoden bereit, um

- ein Zeichen aus Textdateien zu lesen
- ein Zeichen in Textdateien zu schreiben
- die aktuelle Position im Textfile zu ermitteln und zu verändern

Lesen Dem Auslesen dienen die nachfolgenden Funktionen, die alle einen String zurückliefern:

- Function `ReadAll()`
- Function `ReadLine()`
- Function `Read(Characters As Long)`

`ReadAll()` liest die komplette Textdatei in einen String ein, `ReadLine()` dagegen nur eine Zeile. Noch feiner granuliert werden kann das Einlesen mit `Read()`: Der Parameter `Characters` bestimmt die Anzahl der nächsten `n` Zeichen, die eingelesen werden sollen.

Schreiben Analog zu den Lese-Methoden gibt es Methoden für den Schreibzugriff.

- Sub `Write(Text As String)`
- Sub `WriteLine([Text As String])`
- Sub `WriteBlankLines(Lines As Long)`

Es ist möglich, eine Textdatei zeilenweise (`WriteLine()`) oder zeichenweise (`Write()`) zu schreiben. Es gibt jedoch keine explizite Methode, um eine komplette Textdatei mit Hilfe eines Methodenaufrufs zu speichern. `WriteBlankLines()` schreibt eine beliebige Anzahl von Leerzeilen. Das folgende Skript schreibt die Buchstaben des deutschen Alphabets (a bis z) in eine Zeile einer Textdatei. Würde `Write()` durch `WriteLine()` ersetzt, stände jeder Buchstabe in einer eigenen Zeile.

```
Dim FSO 'As Scripting.FileSystemObject
Dim oTX ' As Scripting.TextStream
Dim a ' As Byte
Set FSO = CreateObject("Scripting.FileSystemObject")
Set oTX = FSO.OpenTextFile("d:\buch\docs\alphabet.txt", _
    ForWriting, _ True, TristateUseDefault)
For a = 1 To 26
    oTX.Write Chr(96 + a)
Next
oTX.Close
```

Listing 5.12: Schreiben der Buchstaben von a bis z in eine Datei [*fsotx_writeletters.wsf*]

Arbeit mit Textstreams

```
Dim FSO 'As Scripting.FileSystemObject
Dim oTX ' As Scripting.TextStream
Set FSO = CreateObject("Scripting.FileSystemObject")
Set oTX = FSO.OpenTextFile("d:\buch\docs\append.txt", _
ForAppending, True, TristateTrue)
oTX.WriteLine "Letzter Eintrag am " & Now
oTX.Close
```

Listing 5.13: Anhängen einer Textzeile an eine bestehende Datei [fsotx_append.wsf]

Positionsbestimmung

Sehr wichtig für den Zugriff (insbesondere beim Lesen) ist die Bestimmung der aktuellen Position innerhalb der Datei. Alle vier nachfolgend genannten Eigenschaften sind read-only. Sie können also nicht benutzt werden, um die Position innerhalb der Datei zu verändern.

Column und **Line** geben die Spalte bzw. Zeile an, in der sich der virtuelle Cursor des Textstream-Objekts befindet. Es ist immer die Position, an der als nächstes gelesen oder geschrieben werden würde. Die Zählung beginnt bei 1. Eine neu geöffnete Datei steht also bei **Column** = 1 und **Line** = 1.

Column und Line

Mit **AtEndOfLine** und **AtEndOfStream** (beide liefern nur *True* oder *False*) kann bestimmt werden, ob das Ende der Zeile oder gar das Ende der Datei erreicht ist.

AtEndOf

```
Dim FSO 'As Scripting.FileSystemObject
Dim oTX ' As Scripting.TextStream
Set FSO = CreateObject("Scripting.FileSystemObject")
Set oTX = FSO.OpenTextFile("d:\buch\docs\test.txt", ForReading, _
True, TristateUseDefault)
Do While Not oTX.AtEndOfLine
    say oTX.Read(1)
Loop
oTX.Close
```

Listing 5.14: Das Skript zeigt, wie eine komplette Zeile zeichenweise ausgelesen werden kann. [fsotx_readletters.wsf]

AtEndOfStream muss immer dann verwendet werden, wenn man nicht vorher schon ganz genau weiß, wie lang die Textdatei ist. Um alle Zeilen einer Textdatei auszugeben, müssen Sie in Listing 5.15 nur die Schleife ersetzen:

```
Do While Not oTX.AtEndOfStream
    say oTX.ReadLine
Loop
```

Listing 5.15: Schleife zur Iteration über alle Zeilen einer Textdatei

Positionsveränderung

Positions- veränderung	Die Position des Cursors ändert sich mit jedem Schreib- und Lesezugriff. Sie kann auch durch zwei <code>Skip()</code> -Methoden beeinflusst werden, jedoch nur in eine Richtung: zum Dateiende hin. Eine Rückwärtsbewegung erlaubt Textstream leider nicht.
Skip() und SkipLine()	<code>Skip(lngCharacters)</code> setzt den Cursor <code>n</code> Zeichen nach vorne. Ein negativer Wert für <code>lngCharacters</code> (was einer Rückwärtsbewegung gleichkäme) ist nicht erlaubt. <code>SkipLine()</code> überspringt alle Zeichen bis zum Ende der Zeile. Leider kann man hier nicht angeben, dass mehrere Zeilen zugleich übersprungen werden sollen. Dies muss der Skriptprogrammierer über eine Schleife, die <code>SkipLine()</code> <code>n</code> -fach aufruft, nachbilden.

6 Microsoft Shell Objects

Die Microsoft Shell Objects-Komponente (in der MSDN Library auch *Scriptable Shell Objects* und *Shell Objects for Scripting and Visual Basic* genannt) ist eine COM-Komponente, die sich mit dem Internet Explorer (ab Version 4.0) auf einen Rechner schleicht. In Windows 98/ME und Windows2000/XP/.NET gehört die Komponente daher zum Standard.

Die Shell Objects stellen Funktionen des Windows Explorers und der Windows-Taskleiste bereit und sind aus diesem Grunde eng verzahnt mit den *Microsoft Internet Controls* (SHDOCVW.DLL), siehe Kapitel 7.

Shell Objects for Scripting

Name und Abkürzung	Microsoft Shell Objects
Name der Komponentendatei	SHELL32.DLL
Interner Name der Typbibliothek	Shell32
Helpstring der Typbibliothek	Microsoft Shell Controls And Automation
Abweichende ProgID	Shell
Hersteller	Microsoft
Lizenzierung	Teil des kostenlosen Internet Explorers
Besprochene Version	6.0.2462.0
NT 4 und Windows 95/98/ME	Installation Internet Explorer 6.0
Windows2000	Installation Internet Explorer 6.0
Whistler (Windows XP / Windows.NET)	Version 6.0 enthalten
Dokumentation	siehe Microsoft MSDN Library [MSL00]

Tabelle 6.1:
Schnellinfo Micro-
soft Shell Objects

Funktionsüberblick

Zentrale Funktionen der Shell Objects sind der Zugriff auf:

- ▶ aktive Datei- und Internet Explorer-Fenster, inkl. der dargestellten Ordner und markierten Einträge
- ▶ Start von Datei- und Internet Explorer-Fenstern sowie Start von Standarddialogen des Explorers
- ▶ Funktionen der Taskleiste (Fenster anordnen, minimieren, Eigenschaftsfenster der Taskleiste etc.)
- ▶ Standarddialoge des Startmenüs (Ordner suchen, Ausführen, Windows beenden, Suche nach Dateien, Eigenschaften von Datum/Uhrzeit, Favoriten hinzufügen etc.)

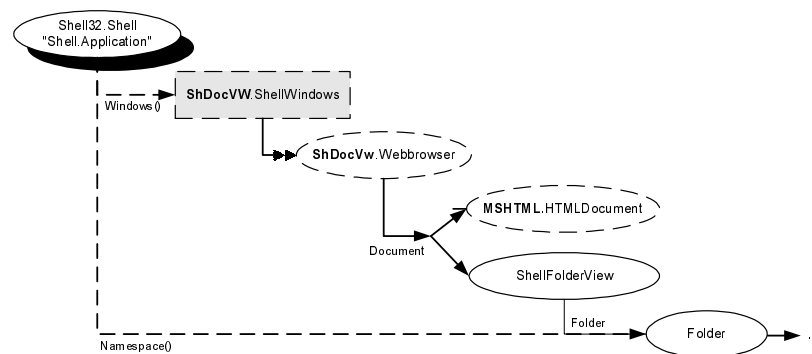
Features

- ▶ Start von Taskleistananwendungen (.CPL-Dateien)
- ▶ Ordner und Dateien
- ▶ Kontextmenüs von Ordnern und Dateien (inkl. der Möglichkeit, die Einträge zu aktivieren)
- ▶ erweiterte Dateiattribute
- ▶ Lesen und Bearbeiten von Dateiverknüpfungen (auch mit WSH-Objekten möglich). Im Gegensatz zu den WSH-Objekten besteht auch die Möglichkeit, nach dem Ziel einer gebrochenen Verknüpfung zu suchen.

Vergleich zu FSO

Viele dieser Funktionen werden auch durch die File System Objects (FSO) aus der Scripting Runtime-Komponente bereitgestellt. Die Implementierungen sind jedoch keineswegs identisch oder kompatibel. Die Shell Objects bieten nicht alle Funktionen der FSO, gehen jedoch in anderen Aspekten über die FSO hinaus.

Abbildung 6.1:
Ansicht der Klasse
Folder im VB6-
Objektkatalog



Laut MSDN Library [MSDN: Shellcc.chm::/shellcc/Shell/Objects/DiskQuotaControl/DiskQuotaControl.htm] sollen die Shell Objects in Version 5.0 auch zwei Klassen `DiskQuotaControl` und `DIDiskQuotaUser` zur Verwaltung des Quota-Managements besitzen. Diese Klassen sind jedoch in Wirklichkeit in der *Microsoft Disk Quota-Komponente* (DSKQUOTA.DLL) implementiert.

6.1 Objektmodell der Shell Objects

Objektmodell

Die nächsten beiden Abbildungen zeigen das Objektmodell. Stammklasse ist `Shell32.Shell`, die in der Registry als »shell.application« registriert ist. Das Objektmodell spiegelt die enge Verzahnung mit den *Internet Explorer-Objekten* wider: Die `ShellWindows`-Collection, die durch die Methode `Windows()` (wirklich eine Methode und kein Attribut) erreicht wird, wird nicht durch die Shell Objects, sondern durch die *Microsoft Internet Controls*



Abbildung 6.2:
Ansicht der Klasse
»Folder« im VB6-
Objektkatalog

(SHDOCVIEW.DLL) implementiert, ebenso wie die Klasse WebBrowser (ein Shell-
Windows-Objekt enthält n WebBrowser-Objekte). Ein WebBrowser-Objekt stellt
entweder eine HTML-Seite oder einen Verzeichnisbaum dar. Im ersten Fall
verweist das Attribut Document auf ein HTMLDocument (das wiederum aus einer
anderen Komponente, nämlich MSHTML kommt). Im zweiten Fall führt der
Weg zurück in die Shell Objects-Komponente und liefert ein ShellFolderView-
Objekt. Das Objekt enthält einen Verweis auf den aktuell geöffneten Ordner
und den aktuell markierten Eintrag.

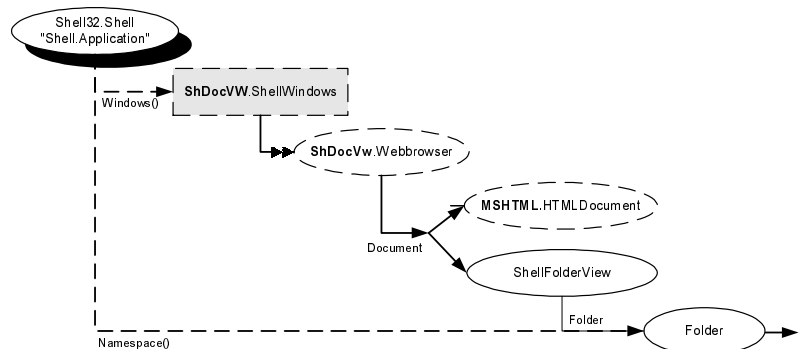
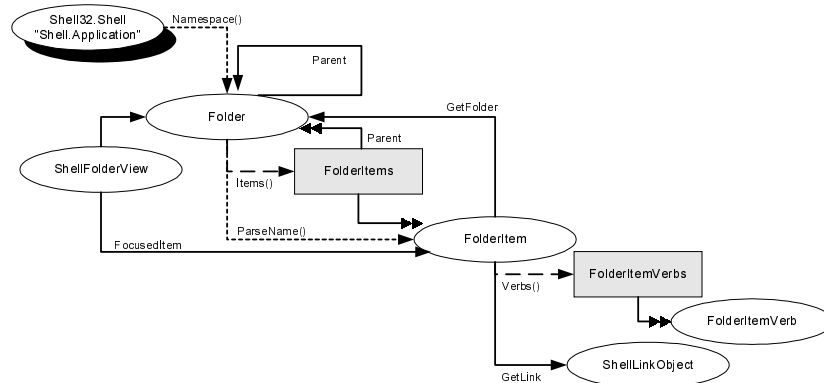


Abbildung 6.3:
Teil 1 des Objekt-
modells der Shell
Objects

Der zuvor geschilderte Weg führt über die geöffneten Explorer-Fenster. Wer
nur im Hintergrund unsichtbar auf Ordner zugreifen will, ist besser bedient,
wenn er sich über die Namespace()-Methode der Shell-Klasse direkt einen
Zeiger auf ein Folder-Objekt besorgt.

Abbildung 6.4:
Teil 2 des Objekt-
modells der Shell
Objects



Folder und FolderItem

Darunter ist das Objektmodell dann wieder etwas geradliniger, wenn auch nicht bis ins Letzte konsequent. Ein `Folder`-Objekt enthält `FolderItems`, wobei ein `FolderItem` ein Ordner oder eine Datei sein kann.

Die Shell Objects vollziehen also nicht die strikte Trennung in eine `File`- und eine `Folder`-Klasse wie FSO. Wenn ein `FolderItem` ein Ordner ist, führt das Attribut `GetFolder` zu einem `Folder`-Objekt; falls eine Datei eine Dateiverknüpfung ist, wird dafür ein `ShellLinkObject` bereitgestellt. Ein `FolderItem` enthält die Menge der verfügbaren Kontextmenüeinträge in Form einer `FolderItemVerbs`-Collection.



Die Benennung der Attribute und Methoden in den Shell Objects ist zum Teil unglücklich gewählt, so dass ein intuitives Verständnis des Objektmodells nicht immer einfach ist.

6.2 Arbeit mit Explorer-Fenstern

Liste aller Explorer-Fenster

Das erste Beispiel zeigt, wie Sie mit den *Shell Objects* in Zusammenarbeit mit den *Internet Explorer-Objekten* die geöffneten Explorer-Fenster (sowohl die Internet Explorer- als auch die Windows Explorer-Fenster, die ja im Zuge des Internet Explorers 4.0 miteinander verschmolzen sind) auflisten können. Die Unterscheidung ist anhand des Klassennamens des Objekts möglich, auf das das `Document`-Attribut des Webbrowsers verweist.

```
Dim SH ' As Shell32.shell
Dim w ' As SHDocVw.WebBrowser_V1
Dim ww ' As SHDocVw.ShellWindows
Set SH = CreateObject("shell.application")
say "Liste der geöffneten Explorer-Fenster:"
' -- Zugriff auf alle geöffneten Explorer-Fenster
Set ww = SH.Windows
' -- Iteration über alle Fenster
```

Arbeit mit Explorer-Fenstern

```
For Each w In ww
    ' -- Unterscheiden nach Browser-Typ
    If TypeName(w.Document) = "HTMLDocument" Then
        say "IE: " & w.LocationURL
    Else
        say "Dateiexplorer: " & w.LocationURL & " (" & _
            w.Document.ViewOptions & ")"
    End If
Next
```

*Listing 6.1: Liste aller geöffneten Web- und Windows Explorer-Fenster
[shell_openWindows.wsf]*

Die Ausgabe könnte so aussehen:

```
Liste der geöffneten Explorer-Fenster:
IE: http://sonne
Dateiexplorer: file:///D:/buch (131)
Dateiexplorer: file:///C:/ (131)
```

Die Zahlen in Klammern repräsentieren die ViewOptions, die sich aus den nachfolgenden Konstanten zusammensetzen. Die ViewOptions sind leider nicht beschreibbar.

ViewOptions

```
CONST SFVVO_SHOWALLOBJECTS = 1
CONST SFVVO_SHOWEXTENSIONS = 2
CONST SFVVO_SHOWCOMPColor = 8
CONST SFVVO_SHOWSYSFILES = 32
CONST SFVVO_WIN95CLASSIC = 64
CONST SFVVO_DOUBLECLICKINWEBVIEW = 128
CONST SFVVO_DESKTOPHTML = 512
```

*Listing 6.2: Konstanten für ShellFolderViewOptions
[CD: /konstantenlisten/komponenten/shell/viewoptions.txt]*

Browser- und Explorer-Fenster öffnen

Natürlich ist es möglich, ein neues Browserfenster zu öffnen. Der Unterschied zwischen Open() und Explore() liegt darin, dass bei Open() die Ordnerleiste nicht geöffnet wird. Natürlich können (auf Grund der Verschmelzung von Internet Explorer und Windows Explorer) auch beliebige andere gültige (Internet-)Protokolle angegeben werden.

FensterIn

```
Dim SH ' As Shell32.Shell
Set SH = CreateObject("shell.application")
' -- Explorer ohne Ordnerleiste
SH.Open "d:\buch"
' -- Explorer mit Ordnerleiste
SH.Explore "d:\buch"
' -- Explorer mit Webseite
SH.Open "http://sonne"
```

Microsoft Shell Objects

```
' -- Verzeichnisdienstbrowser
SH.Open "LDAP://sonne"
' - Netzwerkumgebung anzeigen
SH.Explore 18' Eine ShellSpecialFolder-Konstante
' -- Alle minimieren
SH.MinimizeAll
' -- Computer-Suchen Dialog
SH.FindComputer
```

Listing 6.3: Vielfältige Möglichkeiten für neue Explorer-Fenster [shell_newWindows.wsf]

Shell-Klasse Die Shell-Klasse ist auch der Anbieter des Zugriffs auf die Taskleiste. Die folgende Tabelle zeigt alle Methoden dieser Klasse.

Tabelle 6.2:
Methoden der
Shell-Klasse

Methode	Erläuterung
Namespace(Verzeichnis)	Liefert ein Folder-Objekt auf das gewünschte Verzeichnis
BrowseForFolder (FensterHandle, Titel, Options, StartOrdner)	Zeigt den Standarddialog ORDNER SUCHE. Dabei werden der angegebene Titel und der Startordner verwendet. Die anderen Parameter können auf 0 gesetzt werden.
Windows()	Liefert einen Zeiger auf eine ShellWindows-Collection
Open(Verzeichnis)	Öffnet ein Explorer-Fenster ohne Ordneransicht
Explore(Verzeichnis)	Öffnet ein Explorer-Fenster mit Ordneransicht
MinimizeAll()	Minimiert alle Fenster
UndoMinimizeALL()	Entspricht ALLE MINIMIEREN RÜCKGÄNGIG MACHEN
FileRun()	Standarddialog AUSFÜHREN aus dem Startmenü
CascadeWindows()	Überlappende Anordnung aller Fenster
TileVertically()	Anordnung UNTEREINANDER für alle Fenster
TileHorizontally()	Anordnung NEBENEINANDER für alle Fenster
ShutdownWindows()	Standarddialog WINDOWS BEENDEN aus dem Startmenü
Suspend()	Computer, die einen Suspend-Mode unterstützen, werden schlafen geschickt.
EjectPC()	Auswurf aus einer Docking-Station (sofern von der Hardware unterstützt)
SetTime()	Standarddialog EIGENSCHAFTEN VON DATUM/UHRZEIT aus dem Startmenü
TrayProperties()	Eigenschaftsdialog der Taskleiste zeigen
Help()	Windows-Hilfe

Methode	Erläuterung
FindFiles()	Standarddialog DATEI SUCHEN aus dem Startmenü
FindComputer()	Standarddialog COMPUTER SUCHEN aus dem Startmenü
RefreshMenu()	Neuaufbau des Startmenüs
ControlPanelItem (cpldatei)	Start einer Taskleistanwendung

Tabelle 6.2:
Methoden der
Shell-Klasse
(Forts.)

Sofern ein Verzeichnis als Parameter erwartet wird (z.B. bei Namespace(), Open(), Execute()) kann als Alternative zu einem Pfad auch eine Konstante für einen besonderen Ordner des Dateisystems angegeben werden. Sie finden diese umfangreiche Konstantenliste auf der CD in der Datei /KONSTANTENLISTEN/KOMPONENTEN/SHELL/SONDERORDNER.TXT.



Die Methoden Namespace() und BrowseForFolder() erzeugen keine Fehlermeldungen:

- ▶ Namespace() liefert *Nothing* zurück, wenn der Ordner nicht existiert.
- ▶ BrowseForFolder() startet beim Arbeitsplatz, wenn der Startpfad nicht existiert.
- ▶ Wenn der Benutzer den ORDNER ÖFFNEN-Dialog abbricht, liefert BrowseForFolder() *Nothing* zurück.

Das Beispiel zeigt den letzten Punkt:

```
Dim SH ' As Shell32.Shell
Dim FO ' As Shell32.folder
Const start = "d:\buch"
Set SH = CreateObject("shell.application")
' -- Dialog "Ordner wählen"
Set FO = SH.BrowseForFolder(0, "Bitte Ordner wählen", 0, start)
' -- Auswertung des Rückgabewerts
If FO Is Nothing Then
    MsgBox "Kein Ordner gewählt!"
Else
    MsgBox "Installation startet in..." & start & FO.Title
End If
```

Listing 6.4: Den Benutzer zur Ordnerauswahl auffordern mit BrowseForFolder()
[shell_BrowseForFolder.wsf]

6.3 Arbeit mit Ordnern und Dateien

Die nachfolgende Routine listet alle Dateien und Unterordner eines angegebenen Pfads auf. Dabei werden alle Dateiattribute und die Kontextmenüeinträge ebenfalls ausgegeben. Die Routine bedient sich zweier Unterpro-

Ordner und
Dateien

Microsoft Shell Objects

gramme, `list_attribs()` und `list_verbs()`, die im Anschluss erklärt werden. Die Trennung zwischen Ordnern, Dateien und Verknüpfungen erfolgt mit Hilfe der Attribute `IsFolder` und `IsLink`. Eine Dateiverknüpfung sind auch Dateien – selbst dann, wenn das Ziel der Verknüpfung ein Ordner ist.

```
Dim SH ' Shell32.Shell
Dim FO ' Shell32.Folder
Dim fi ' Shell32.FolderItem
Const strFO = "d:\buch "
Set SH = CreateObject("shell.application")
say "---- Inhalt des Ordners " & strFO
' -- Zugriff auf Ordner
Set FO = SH.NameSpace(strFO)
' Liefert im Fehlerfall Nothing!
If FO Is Nothing Then
    MsgBox "Pfad nicht gefunden!"
Else
    say "-- Ordner:"
    ' -- Iteration über alle Items (zuerst die Ordner!)
    For Each fi In FO.Items
        If fi.IsFolder Then
            say fi.Name
            list_attribs FO, fi
            list_verbs fi
        End If
    Next
    say "-- Dateien:"
    ' -- Iteration über alle Items
    For Each fi In FO.Items
        If Not fi.IsFolder Then
            ' -- Unterscheidung zwischen Dateien und Links
            If fi.IsLink Then
                say "Verknüpfung " & fi.Name
            Else
                say "Datei " & fi.Name
            End If
            list_attribs FO, fi
            list_verbs fi
        End If
    Next
End If
```

Listing 6.5: Dieses Skript gibt alle Dateien und Unterverzeichnisse getrennt aus und liefert zudem detaillierte Informationen zu jedem Eintrag. [shell_ordner.wsf]

Zugriff auf Dateieigenschaften

Die Methode `GetDetailsOf()` ist auf der Ebene der `Folder`-Klasse angesiedelt und übergibt ohne Angabe eines `FolderItem`-Objekts den Namen der Eigenschaft und unter Angabe eines entsprechenden Objektverweises den konkreten Wert der Eigenschaft für das angegebene Objekt. Leider können die Eigenschaften mit diesem Verfahren nicht über ihren Namen, sondern nur über die laufende Spaltennummer in der Standard-Ordneransicht angesprochen werden. Die übliche Belegung ist somit:

GetDetailsOf()

```
Const DetailName = 0
Const DetailSize = 1
Const DetailType = 2
Const DetailLastMod = 3
Const DetailAttrib = 4
Const DetailTip = -1
```

Listing 6.6: Konstanten für Spaltennamen

Die Schleife in `list_attribs()` durchläuft daher die Werte von -1 bis 4. -1 ist der Inhalt des Tooltips, der bei einigen Dateitypen (z.B. Microsoft Office-Dokumenten) angezeigt wird. Dadurch besteht z.B. Zugriff auf die Informationen Autor, Titel und Betreff.

```
Sub list_attribs(fo, fi)
Dim i, eigenschaft, wert
' --- Alle fünf Attribute ausgeben
For i = -1 To 4
    eigenschaft = fo.GetDetailsOf(, i)
    wert = fo.GetDetailsOf(fi, i)
    say i & ": " & eigenschaft & "=" & wert
Next
End Sub
```

Listing 6.7: list_attribs listet alle Attribute eines FolderItem-Objekts auf.

Kontextmenüeinträge

Zugriff auf die Kontextmenüeinträge gibt die Verbs-Collection. `list_verbs()` listet alle Kontextmenüeinträge eines übergebenen `FolderItem`-Objekts.

Verbs

```
Sub list_verbs(fi)
Dim v ' As Shell32.FolderItemVerb
' -- Iteration über alle Kontextmenüeinträge
For Each v In fi.Verbs
    say "- " & v.Name
Next
End Sub
```

Listing 6.8: list_verb() listet alle Kontextmenüeinträge eines FolderItem-Objekts.



Leider gibt es in der Version 4.71 der Komponente einen Fehler in der IDispatch-Schnittstelle der FolderItemVerb-Klasse, so dass spätes Binden und damit die Verwendung in Skriptsprachen nicht möglich ist. Möglich ist in Skripten aber ein Aufruf mit der Methode DoIt() in der Klasse FolderItemVerb oder durch InvokeVerb(verbname) in der Klasse FolderItem. InvokeVerb() würde das Standard-Verb ausführen.

Das letzte Beispiel zeigt einen direkten Zugriff auf eine Word-Datei in einem Verzeichnis und das Ausführen des Kontextmenüeintrags DRUCKEN.

```
Dim SH ' Shell32.Shell
Dim FO ' Shell32.folder
Dim FI ' Shell32.FolderItem
Const strFO = "d:\buch\docs"
Set SH = CreateObject("shell.application")
' -- Zugriff auf Ordner
Set FO = SH.Namespace(strFO) ' Liefert im Fehlerfall Nothing!
Set FI = FO.ParseName("test.doc")
say FI.Name & " wurde zuletzt geändert am " & FI.ModifyDate
' -- Standardeintrag ausführen
FI.InvokeVerb "&Drucken"
```

Listing 6.9: Drucken einer Word-Datei über ein Verb [SHELL_InvokeVerb.wsf]

Inkompatibilität mit FSO

»Scripting.Folder« versus »Shell32.Folder«

Die Klasse Folder aus den *File System Objects* und die namensgleiche Klasse Folder aus den *Shell Objects* besitzen keine gemeinsame Schnittstelle. Eine direkte Zuweisung von Zeigern ist also nicht möglich. Die Synchronisation muss über den Pfadnamen erfolgen. Leider kann anhand eines konkreten Objekts nur auf Basis von Try&Fail festgestellt werden, welche Klasse vorliegt, da TypeName() in beiden Fällen nur »Folder« liefert.

```
Set SH = CreateObject("Shell.Application")
Set FSO = CreateObject("Scripting.FileSystemObject")
' -- F01 ist ein Scripting.Folder
Set fo1 = FSO.GetFolder("c:\")
say TypeName(fo1) ' Ergibt "Folder"
' ...
' -- Abbildung auf einen Shell32.Folder
Set fo2 = SH.Namespace(fo1.Path)
say TypeName(fo2) ' Ergibt "Folder"
```

Listing 6.10: Vergleich der Folderobjekte [shell_ShellUndFSO.wsf]

7 Microsoft Internet Controls

Der Name *Microsoft Internet Controls* lässt vermuten, dass diese Komponente alle möglichen Funktionen für die Arbeit im Internet bereitstellt. Dem ist aber nicht so: Die Internet Controls beschränken sich auf Funktionen des Internet Explorer-Webrowsers; in der MSDN Library heißt die Komponente daher auch *Webbrowser Control*. Das »Controls« im Namen der Komponente erklärt sich damit, dass diese Komponente auch ein ActiveX-Steuer-element ist, nämlich genau jenes, mit dem man ein Internet Explorer-Fenster in eigene ActiveX-Container einbinden kann, also Anwendungen schreiben kann, die einen Webbrowser enthalten.

Die Controls sind aber auch als nichtvisuelle Automatisierungskomponente nutzbar, um den Internet Explorer fernzusteuern. Dabei enthalten die Internet Controls nur das äußere Grundgerüst des Explorers. Die Hauptfunktionalität des IE und die Darstellung der Dokumente übernimmt die Komponente *MSHTML*. Die *Internet Controls-Komponente* ist aber nicht nur mit *MSHTML*, sondern auch mit der Komponente *Shell Objects* eng verzahnt (vgl. Kapitel 6) – einmal mehr ein Beweis für die enge Verflechtung von Webbrowser und Windows-Benutzeroberfläche.

Internet Explorer-Komponente

Verzahnung

Name und Abkürzung	Microsoft Internet Controls / Webbrowser Control
Name der Komponentendatei	SHDocVw.dll
Interner Name der Typbibliothek	SHDocVw
Helpstring der Typbibliothek	Microsoft Internet Controls
Abweichende ProgID	InternetExplorer.Application
Hersteller	Microsoft
Lizenzierung	Kostenloses Add-on
Besprochene Version	6.0.2462.0
NT4 und Windows 95/98/ME	Installation Internet Explorer 6.0
Windows2000	Installation Internet Explorer 6.0
Whistler (Windows XP / Windows.NET)	Version 6.0 enthalten
Dokumentation	Microsoft MSDN Library [MSL00]

Tabelle 7.1:
Schnellinfo
Microsoft Internet
Controls

7.1 Objektmodell des Webbrowser Controls

Objektmodell Die Microsoft Internet Controls enthalten nur ein minimales Objektmodell, das aus der Collection `ShellWindows` mit Instanzen der Klasse `Webbrowser` besteht. Diese Beziehung ist grafisch bereits im Zusammenhang mit den Shell Objects (Kapitel 6) dargestellt worden. Die Klasse `Webbrowser` existiert innerhalb der Komponente in verschiedenen Versionen. Weiterhin wird die hier nicht näher besprochene Klasse `ShellUIHelper` implementiert.

Abbildung 7.1:
Ansicht der
SHDocVw-Typ-
bibliothek im
VB6-Objektkatalog



7.2 Anwendungsbeispiele

Navigation Das folgende Skript zeigt, wie ein Explorer-Fenster mit einem bestimmten Uniform Resource Locator (URL) angezeigt werden kann. Ein virtuelles Fenster wird bereits mit der Instanziierung erzeugt. Die Darstellung auf dem Bildschirm erfolgt aber erst durch das Setzen des Attributs `visible`.

Anwendungsbeispiele

```
Dim ie ' As SHDocVw.WebBrowser
Set ie = CreateObject("InternetExplorer.Application")
ie.navigate "http://sonne2000"
ie.MenuBar = False
ie.Visible = True
MsgBox "Gelesen?"
ie.Quit
```

Listing 7.1: IE starten und Webseite anzeigen [ie_message.wsf]

Das Skript schließt das Fenster wieder, nachdem der Anwender die Dialogbox weggeklickt hat.

Der Internet Explorer als Ausgabefenster

Der Internet Explorer eignet sich hervorragend dazu, als Ein- und Ausgabefenster für WSH-Skripte benutzt zu werden. Um ein leeres Dokument in den IE zu bekommen, gibt es einen Trick: Der Moniker ABOUT: stellt all das, was nach dem Doppelpunkt angegeben wird, auf einer leeren Seite im Browser dar. Da das komplette Leerlassen und Zeichenketten, die nur aus Leerzeichen bestehen, zu Problemen führen, sollten Sie nach dem Doppelpunkt irgendetwas angeben.

**Ausgaben via
IE-Fenster**

```
Dim ie ' As SHDocVw.WebBrowser
Dim doc ' As MSHTML.HTMLDocument
Set ie = CreateObject("InternetExplorer.Application")
ie.Visible = True
ie.navigate "about:<HTML></HTML>"
' -- warten bis dargestellt
Do While ie.Busy
Loop
' -- Zugriff auf DOM
Set doc = ie.Document
doc.Write "<h2>Hallo!</h2>"
```

Listing 7.2: Verwendung des IEs als Bildschirmmaske [ie_about.wsf]

Nachdem Sie Zugriff auf das Document-Objekt des dargestellten ABOUT-Dokuments genommen haben, können Sie das Internet Explorer-Fenster dynamisch mit beliebigen HTML-Inhalten füttern. Die Möglichkeiten zur dynamischen Generierung von HTML-Inhalten mit Hilfe des Document Object Models (DOM) für HTML ist Thema in Kapitel 7.1.

Bitte beachten Sie die notwendige Schleife, die darauf wartet, dass das Attribut `ie.busy` den Wert `False` erhält. Wenn Sie zu früh auf `ie.document` zugreifen, werden Sie einen Fehler erhalten. Diese Fehlerursache ist tückisch, weil der Internet Explorer lokal verfügbare Seiten so schnell darstellt, dass der Fehler auch ohne die Abfrage von `busy` normalerweise nicht auftritt.



Ereignisse

Webbrowser-Ereignisse Das Webbrowser-Objekt definiert eine Reihe von Ereignissen, die Sie abfangen können. Wenn Sie den Browser zur Darstellung von Informationen nutzen, können Sie mit dem `OnQuit`-Ereignis feststellen, ob der Benutzer das Fenster schließt. Verhindern können Sie das Schließen des Fensters leider nicht. Ihr Skript könnte aber den Benutzer darauf hinweisen und die Seite erneut darstellen.

```
' --- Ereignisbehandlung
```

```
ie_OnQuit() Sub IE_OnQuit()  
WScript.Echo "Internet Explorer wird geschlossen"  
ende = true  
End Sub
```

```
ie_document-complete() Sub ie_documentcomplete(o,url)  
WScript.Echo "Download beendet:" & url  
End Sub
```

```
ie_event() ' --- Hauptroutine  
Dim ie, ende  
Set ie = WScript.CreateObject("InternetExplorer.Application","IE_")  
ie.navigate "http://1.1.1.21"  
ie.MenuBar = False  
ie.Visible = True  
ende = False  
do while not ende  
    WScript.sleep 1  
loop  
WScript.Echo "Programmende"
```

Listing 7.3: Ein WSH-Skript, das auf Ereignisse aus dem IE-Fenster reagiert. Das Skript wird erst beendet, wenn das Browserfenster geschlossen wird. [`_ie_event.vbs`]

8 Document Object Model (DOM)

Das *Document Object Model (DOM)* ist eine Programmierschnittstelle für HTML- und XML-Dokumente. DOM ist die Repräsentation eines HTML- oder XML-Dokuments in Form eines Objektmodells, das die logische Struktur des Dokuments wiedergibt. Mit DOM lassen sich per Programmcode folgende Funktionen ausführen:

- ▶ Navigation durch die Struktur und den Inhalt eines Dokuments
- ▶ Veränderung der Struktur durch Einfügen und Löschen von Elementen
- ▶ Veränderungen der Eigenschaften von Strukturelementen
- ▶ Veränderung der Inhalte
- ▶ Erstellung kompletter Dokumente von Grund auf

Das Document Object Model liegt in der Verantwortung der DOM Working Group des World Wide Web Consortiums (W3C). Aktuell liegen DOM Level 1 als Proposed Recommendation und Level 2 als Working Draft vor.

Aufbau der Spezifikation

Die DOM-Spezifikation Level 1 besteht aus zwei Teilen,

- ▶ dem DOM-Kern und
- ▶ dem DOM für HTML

Der DOM-Kern ermöglicht allgemein die Beschreibung eines hierarchischen Dokuments, während das DOM für HTML spezifische Schnittstellen für die in HTML erlaubten Tags definiert.

Der Kern wiederum besteht aus Basisschnittstellen und erweiterten Schnittstellen für XML. Letztere werden für den Zugriff auf HTML-Dokumente nicht benötigt. Eine Implementierung des DOM muss die Basisschnittstellen des DOM-Kerns implementieren und zusätzlich entweder

- ▶ die erweiterten Schnittstellen für XML (für den Zugriff auf XML-Dokumente) oder
- ▶ das DOM für HTML (für den Zugriff auf HTML-Dokumente) oder
- ▶ das komplette DOM Level 1 (für den Zugriff auf HTML- und XML-Dokumente).

Die Microsoft COM-Komponente *MSHTML* ist eine Implementierung des zweiten Typs, also für den Zugriff auf HTML-Dokumente. *MSXML* ist dagegen eine Implementierung des ersten Typs für den Zugriff auf XML-Dokumente.

Das DOM Level 2 war zum Zeitpunkt der Erstellung dieses Buchs noch nicht endgültig verabschiedet. DOM Level 2 wird in Erweiterung zu Level 1

Document Object Model

W3C

DOM-Spezifikation Level 1

MSHTML und MSXML

DOM Level 2

Document Object Model (DOM)

weitere optionale Module (z.B. für Events, Iteration/Filterung, Cascading Style Sheets) enthalten.

DOM Level 0 Als DOM Level 0 werden die in Netscape Navigator 3.0 und Internet Explorer 3.0 implementierten Schnittstellen bezeichnet. Es handelt sich dabei jedoch nicht um eine formale Spezifikation des W3C, sondern um eine nachträgliche Benennung durch Hersteller implementierter proprietärer Funktionalitäten.

DOM und COM

DOM ist nicht COM und nicht DCOM: Das DOM wurde von Microsoft mit COM implementiert und kann auch mit DCOM genutzt werden, andere Hersteller haben aber andere Implementierungsarten gewählt.

Dynamic HTML Object Model hieß bei Microsoft die Vorstufe der kompletten Implementierung von DOM, die es im Internet Explorer 4.0 gab. Der Internet Explorer ab Version 5.0 implementiert das vollständige DOM des W3C in Form der COM-Komponenten *MSHTML* und *MSXML*.

8.1 HTML Document Object Model

MSHTML DOM MSHTML ist die COM-Komponente von Microsoft für den Zugriff auf HTML-Dokumente und die Steuerung des Browsers. MSHTML implementiert aus dem W3C-DOM die Basisschnittstellen des Kerns und das DOM für HTML. Die Komponente enthält darüber hinaus Klassen für den Zugriff auf den Browser, die nicht Teil des W3C-DOMs sind.

MSHTML ist eine sehr mächtige Komponente mit 110 Klassen, 301 Schnittstellen und 100 Konstantenlisten. An der wesentlich größeren Anzahl der Schnittstellen gegenüber der Anzahl der Klassen können Sie bereits erkennen, dass MSHTML intensiv mit Mehrfachschnittstellen arbeitet.

Tabelle 8.1:
Schnellinfo
Document Object
Model (DOM)

Name und Abkürzung	MSHTML
Name der Komponentendatei	MSHTML.DLL (Typbibliothek: MSHTML.TLB)
Interner Name der Typbibliothek	MSHTML
Helpstring der Typbibliothek	Microsoft HTML Object Library
Hersteller	Microsoft
Lizenzierung	Teil des kostenlosen Internet Explorers
Besprochene Version	6.0.2462.0
NT4 und Windows 95/98/ME	Installation Internet Explorer 6.0
Windows2000	Installation Internet Explorer 6.0
Whistler (Windows XP / Windows.NET)	Version 6.0 enthalten
Dokumentation	siehe Microsoft MSDN Library [MSL00]



Abbildung 8.1:
MSHTML umfasst
zahlreiche Klassen
(VB6-Objekt-
katalog)

8.1.1 MSHTML-Objektmodell

Die Abbildung 8.2 zeigt die browserbezogenen Klassen der MSHTML-Komponente, deren Stammklasse `Window` ist. Das Objektmodell enthält an zwei Stellen eine Verbindung zum HTML-DOM: Einerseits gibt es ein `Document`-Objekt direkt unterhalb eines `Window`-Objekts, andererseits enthält die `Collection Frames` eine beliebige Anzahl von `Document`-Objekten, wenn die dargestellte Seite ein Frameset ist.

Browser-Klassen

Abbildung 8.3 zeigt den Teil der MSHTML-Komponente, die das DOM implementiert. An der Spitze steht das `Document`-Objekt, das auf zahlreiche Collections verweist. Die wichtigste Collection ist die `All`-Collection, die alle Elemente des Dokuments enthält. Daneben gibt es elementtypspezifische Collections wie `Forms`, `Scripts` und `Images`.

DOM-Klassen

Eine HTML-Seite besteht aus Sicht des HTML-DOMs aus COM-Objekten verschiedener Klassen. Im MSHTML haben all diese Klassen gemein, dass sie die Schnittstelle `IHTMLDOMElement` implementieren. Zur Vereinfachung bildet das HTML-DOM diese Objekte in Listen ab, wobei die Listen Instanzen der Klasse `HTMLDOMElementCollection` sind. Für einige Elementtypen gibt es spezielle Collection-Klassen (`HTMLFramesCollection` und `HTMLStyleSheetCollection`). Das Document-Objekt enthält in Form eines Attributs den Verweis auf neun »Element«-Collections (`all`, `anchors`, `applets`, `embeds`, `forms`, `images`, `links`, `plugins`, `scripts`), die selbsterklärende Namen haben.

Collections

Document Object Model (DOM)

Abbildung 8.2:
Browserbezogene
Klasse aus
MSHTML

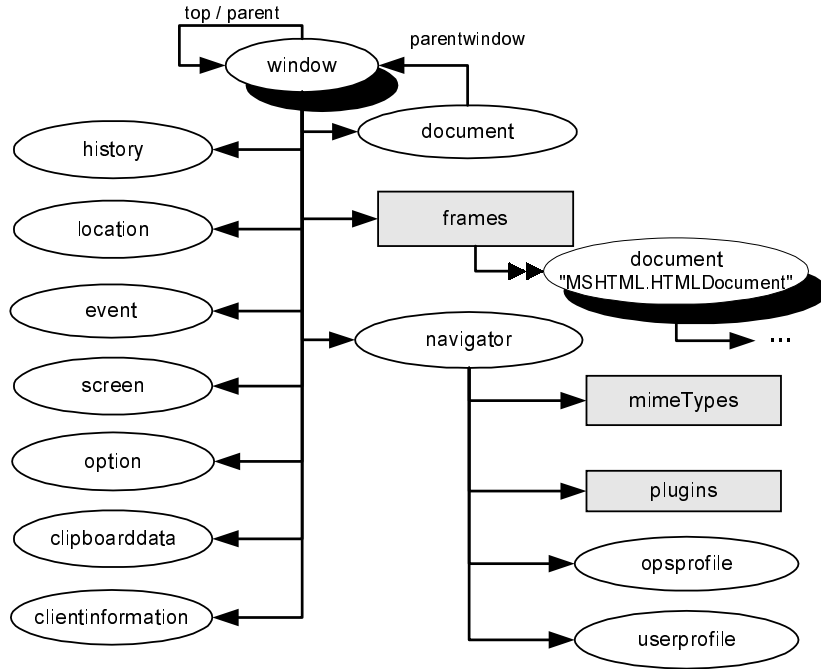
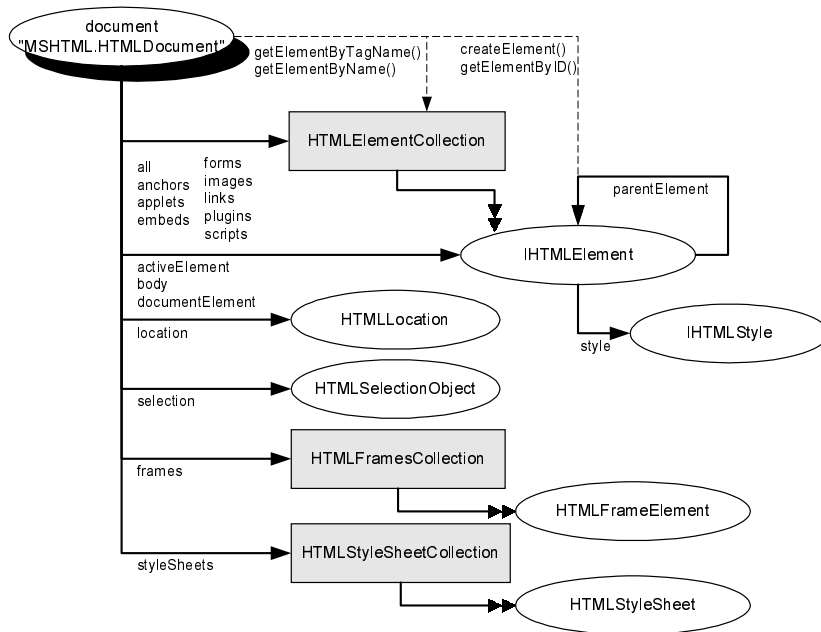


*Abbildung 8.3:
Das DOM in
MSHTML*



8.1.2 Zugriff auf ein einzelnes Element

Instrumente zur Identifizierung eines Elements in einem HTML-Dokument sind die `id`-Attribute, die jedes Tag in HTML besitzen kann, z.B.

Elementzugriff

```
<h2 id="ueberschrift">Anmelde-Seite</h2>
<form id="login" action="next.htm">
<P id=feld1>Ihr Name
<INPUT id=benutzername name=benutzer value=HS>
</P></form>
<DIV id="ausgabebereich"> </DIV>
```

Listing 8.1: Eine Anmeldemaske als Beispiel zur Verwendung des ID-Attributs (Ausschnitt aus der Datei `elementzugriff.htm` auf der CD)

Für den Zugriff auf ein einzelnes Element gibt es auf Basis des `ID`-Attributs verschiedene Möglichkeiten:

Verwendung des ID-Attributs

- über die `all`-Collection mit dem `ID`-Attribut

```
document.all.item("IDName")
```

Dies kann verkürzt werden, weil die Collections für alle untergeordneten Objekte dynamisch ein Attribut anlegen:

```
document.all.IDName
```

- über die Methode `getElementById()` der `Document`-Klasse

```
document.getElementById("id")
```

- über eine spezielle Collection, z.B. die `Forms`-Collection

```
document.forms.item("id")
```

Auch hier ist eine Verkürzung möglich:

```
document.forms.id
```

Innerhalb eines Browser-Skripts stehen alle mit einer `ID` versehenen Elemente auch als `Intrinsic Objects` zur Verfügung, so dass die `ID` wie eine Objektvariable behandelt werden kann. Auf den Inhalt des Absatzes mit der `ID` »*Ueberschrift*« kann dann einfach über `ueberschrift.innertext` zugegriffen werden.

Eine Ausnahme gibt es aber leider für Eingabefelder innerhalb eines Formulars: Der Zugriff ist nur über das Formular-Objekt möglich, also im obigen Beispiel `login.benutzername.value`.

Ein `id`-Attribut soll pro Dokument einmalig sein. Wenn Sie ein `id`-Attribut mehrfach vergeben, dann funktionieren die obigen Zugriffsmethoden nicht mehr, weil das Ergebnis kein Einzelobjekt mit `HTMLElement`-Schnittstelle ist, sondern eine Collection vom Typ `HTMLCollection`.



Document Object Model (DOM)

IHTMLElement Es gibt im HTML-DOM für sehr viele Elementtypen eigene Klassen, die alle die Schnittstelle `IHTMLElement` unterstützen. Fast jedes Objekt, das Sie ansprechen, verfügt daher über die Mitglieder der Schnittstelle `IHTMLElement`, was die Arbeit sehr vereinfacht. Seit Version 5.0 implementieren die Elementklassen die zusätzliche Schnittstelle `IHTMLElement2`.

*Tabelle 8.2:
Ausgewählte
Attribute der
Schnittstelle
IHTMLElement*

Attributname	Erläuterung
<code>Id</code>	Das <code>id</code> -Attribut des Elements
<code>TagName</code>	Tagname des Elements
<code>className</code>	Das <code>class</code> -Attribut eines Elements
<code>parentElement</code>	Ein Zeiger auf das übergeordnete Element im DOM-Baum
<code>Style</code>	Ein Zeiger auf ein <code>Style</code> -Objekt, das die Information aus dem zu diesem Objekt angegebenen <code>Style</code> -Attribut enthält
<code>Children</code>	Ein Zeiger auf eine Collection der direkten Unterknoten im Baum
<code>All</code>	Ein Zeiger auf eine Collection aller Unterknoten im Baum (also auch die Unterknoten der Unterknoten etc.)
<code>Document</code>	Ein Zeiger auf das <code>document</code> -Objekt
<code>innerHTML</code>	Enthält den kompletten Inhalt des Elements einschließlich der untergeordneten Elemente. Bei einer Zuordnung an dieses Attribut wird übergebener Text geparkt und entsprechend formatiert.
<code>InnerText</code>	Enthält den Inhalt des Elements, wobei alle untergeordneten HTML-Tags herausgefiltert werden. Bei einer Zuordnung an dieses Attribut wird übergebener Text NICHT geparkt, Tags werden nach Umsetzung in die entsprechenden Sonderzeichen in Textform ausgegeben.
<code>outerHTML</code>	Enthält den kompletten Inhalt des HTML-Elements einschließlich der untergeordneten Elemente sowie das Tag des Elements selbst
<code>Title</code>	Enthält den Wert des <code>title</code> -Attributs (sofern das Element über dieses Attribut verfügt)
<code>Language</code>	Enthält den Wert des <code>language</code> -Attributs (sofern das Element über dieses Attribut verfügt)
<code>onclick</code> , <code>ondblclick</code> , <code>onmouseover</code> , etc.	Inhalt der jeweiligen Ereignisattribute eines Elements

Methode	Erläuterung
setAttribute (attributname, name, flag)	Setzt ein Attribut auf einen Wert. Wenn das Attribut nicht existiert, wird es neu angelegt. Flag gibt an, ob die Groß-/Kleinschreibung des Attributnamens berücksichtigt werden soll (0 = Nein, 1 = Ja). Standard ist Ja.
getAttribute (attributname, flag)	Ermittelt den Wert eines Attributs. Wenn es mehrere Attribute gleichen Namens gibt, wird nur der letzte Wert zurückgegeben (Bedeutung von flag wie bei setAttribute()).
removeAttribute (attributname, flag)	Entfernt ein Attribut (Bedeutung von flag wie bei setAttribute()).
insertAdjacentHTML (strOrt, strText)	Fügt dem Inhalt des Elements HTML-Inhalt hinzu, dabei wird der übergebene Text geparkt. Das Attribut strOrt gibt an, an welcher Stelle der Text eingefügt werden soll: Erwartet wird keine symbolische Konstante, sondern einer der vier folgenden Strings: »beforeBegin«, »afterBegin«, »beforeEnd«, »afterEnd«.
insertAdjacentText (strOrt, strText)	Fügt dem Inhalt des Elements Text hinzu, dabei wird der übergebene Text nicht geparkt. Das Attribut strOrt gibt an, wo der Text eingefügt werden soll (s. o.).
click	Simuliert einen Klick des Benutzers auf dieses Element, so dass das onclick-Ereignis ausgelöst wird.

Tabelle 8.3:
Ausgewählte
Methoden der
Schnittstelle
HTMLElement

Die Besprechung aller »Element«-Klassen und ihrer Schnittstellen würde den Rahmen dieses Buchs sprengen – diese Informationen füllen üblicherweise mehrere hundert Seiten starke Referenzhandbücher (z.B. [GOO98]). So besitzt allein die Klasse HTMLInputElement achtzehn Schnittstellen.

Hinweis auf
Referenz

Sie können alle Daten eines Elements auf Grundlage der beschriebenen Attribute und Methoden (insbesondere innerHTML, getAttribute() und setAttribute()) lesen und verändern. Die Arbeit mit den anderen Schnittstellen bringt Ihnen nur noch den Vorteil, dass diese im direkten Zugriff auf die Punktnotation stehen. Beispielsweise verfügen die Klassen für Formularelemente über das Value-Attribut, das aber auch problemlos über getAttribute()/setAttribute() der HTMLElement-Schnittstelle gelesen werden könnte. HTMLElement spielt in MSHTML die gleiche Rolle wie IADs bei ADSI: Man kann alles mit dieser einen Schnittstelle machen, mit anderen Schnittstellen wird es nur noch etwas eleganter.



Document Object Model (DOM)

Beispiel

```
' --- Zugriff auf ein einzelnes Element
say "Überschrift: " & ueberschrift.innertext
say "Benutzer: " & loginform.benutzername.value
say "Benutzer: " & document.all("Benutzername").value
say "Benutzer: " & document.getElementById("benutzername").value
say "Der 14. Tag der Seite: " & document.all.item(13).value
say "Das 1. Eingabefeld des Formulars: " & _
document.forms("loginform").item(0).value
say "Das 1. Eingabefeld des Formulars: " & _
document.forms.loginform.item(0).value
say "Action-Attribut des Forms: " & _
document.forms.loginform.getAttribute("action")
```

Listing 8.2: Beispiele für Lesezugriff auf Elemente. Für den Zugriff auf das Eingabefeld Benutzername werden verschiedene Möglichkeiten gezeigt.

```
ueberschrift.innerText = "Neue Überschrift"
ueberschrift.setAttribute "align","center"
loginform.benutzername.value = "HolgerS"
```

Listing 8.3: Beispiele für Veränderungen an Elementen

8.1.3 Arbeit mit Collections

HTMLElementCollection

Innerhalb einer `HTMLElementCollection` wird ein Element auf Basis seines ID-Attributs oder auf Grund seiner Position innerhalb der Collection identifiziert. Es ist leider nicht möglich, über eine `HTMLElementCollection` per `For...Each` zu iterieren. Die Iteration ist nur über eine `For...Next`-Schleife mit Hilfe der Attribute `length` und `item` möglich.

Im folgenden Skript wird die Collection Links ausgegeben.

```
set col = document.links
For i = 0 To col.length - 1
  Set oEle = col(i)
  say oEle.innerHTML & " -> " & oEle.href
Next
```

Listing 8.4: Ausgabe aller Links auf einer Seite

Es bietet sich an, zu Testzwecken eine universelle Ausgaberoutine für `HTMLElementCollection`-Collections einzusetzen:

```
Sub saycol(col)
For i = 0 To col.length - 1
  Set oEle = col(i)
  say "<b>" & i & ": " & oEle.tagName & " (" & oEle.id & ")</b>"
Next
End Sub
```

Listing 8.5: Universelle Ausgabe einer beliebigen HTMLElementCollection

HTML Document Object Model

```
' --- Ausgabe verschiedener Listen
say "<hr>Alle Tags:"
saycol(document.all)
say "<hr>Alle Input Tags:"
saycol(document.getElementsByTagName("INPUT"))
say "<hr>Alle Tags, die Benutzer heißen:"
saycol(document.getElementsByName("benutzer"))
```

Listing 8.6: Test der Routine saycol()

Die Methoden `getElementsByTagName()` und `getElementsByName()` der Document-Klasse liefern immer Collections, auch dann, wenn es nur ein Element dieses Namens gibt. `getElementById()` liefert normalerweise ein Element. Wenn Sie eine ID doppelt vergeben, liefert auch diese Methode bei Anfrage nach dieser ID eine Collection.



Abbildung 8.4:
Ausschnitt aus der
Seite elementzu-
griff.html nach der
Ausführung der ver-
schieden in diesem
Kapitel besproche-
nen Skripte

8.2 XML Document Object Model (MSXML) Version 4.0

MSXML	Den Zugriff auf das Document Object Model (DOM) eines XML-Dokuments ermöglicht die Komponente MSXML. MSXML erweitert jedoch das standardisierte XML-DOM um zusätzliche Funktionalitäten zum Parsen des Dokuments und zum Laden eines Dokuments in den XML-Parser. Erst durch diese Erweiterungen wird es möglich, MSXML unabhängig vom Internet Explorer in eigene Anwendungen zu integrieren.
XMLHttpRequest	Die Komponente MSXML enthält auch eine Klasse XMLHttpRequest (ProgID Microsoft.XMLHTTP), die zur Ausführung von HTTP-Anfragen nach XML-Daten dient. Diese Klasse wird hier nicht besprochen. Nähere Informationen dazu finden Sie in der MSDN Library.
Neuerungen in Version 4.0	Neu in MSXML 4.0 gegenüber 3.0 ist, dass zur Strukturdefinition nicht nur XML-Data Reduced (XDR) und Document Type Definition (DTD) sondern auch die neue XML Schema Definition (XSD)-Sprache verwendet werden kann.

Tabelle 8.4:
Schnellinfo
MSXML

Name und Abkürzung	Microsoft XML
Name der Komponentendatei	msxml.dll
Interner Name der Typbibliothek	MSXML2 (kein Druckfehler im Buch, sondern wohl ein Fehler von Microsoft, dass es nicht MSXML3 heißt!)
Helpstring der Typbibliothek	Microsoft XML
Abweichende ProgID	Microsoft.XMLDOM
Hersteller	Microsoft
Lizenzierung	Teil des kostenlosen Internet Explorers Kostenloses Add-on
Besprochene Version	8.0.7820.0
NT4 und Windows 95/98/ME	Installation Internet Explorer 6.0 oder kostenloses XML-Parser 4.0-Add-On [CD:/install/komponenten/XML]
Windows2000	Installation Internet Explorer 6.0 oder kostenloses XML-Add-on [CD:/install/komponenten/XML]
Whistler (Windows XP / Windows.NET)	Version 8.0 enthalten
Dokumentation	siehe Microsoft MSDN Library [MSL00]

XML Document Object Model (MSXML) Version 4.0

Typbibliothek

Leider enthält die Typbibliothek – mit Ausnahme der Stammklasse `DOMDocument` – nicht die Klassendefinitionen, sondern nur die Definitionen der Standardschnittstellen. Wenigstens entsprechen diese den Konventionen: großes I + Klassenname. Sie müssen beim frühen Binden also `IXMLDOMNode` statt `XMLDOMNode` deklarieren.

Typbibliothek

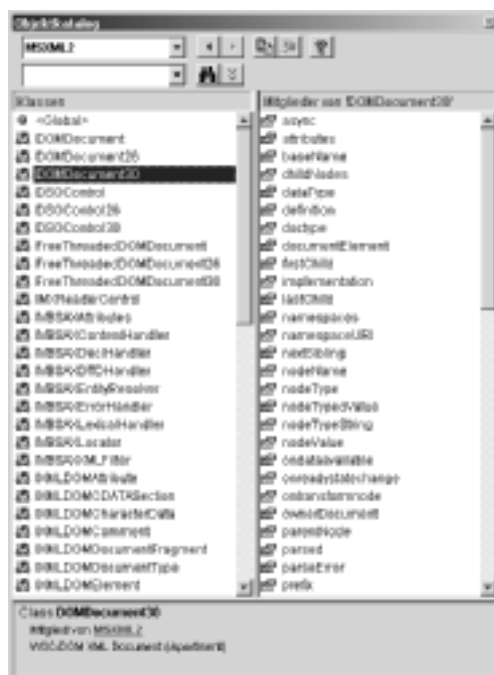


Abbildung 8.5:
Die Typbibliothek
von MSXML
Version 3.0 heißt
MSXML2

Versionen

Es sind zahlreiche Versionen von MSXML im Umlauf. Die folgende Tabelle gibt einen Überblick.

Produktversion	Komponentendatei	Version der Komponentendatei
1.0	msxml.dll	4.71.1712.5
1.0a	msxml.dll	4.72.2106.4
1.0 Service Pack 1 (SP1)	msxml.dll	4.72.3110.0
2.0	msxml.dll	5.0.2014.0206
2.0a	msxml.dll	5.0.2314.1000
2.0b	msxml.dll	5.0.2614.3500
2.5 Beta2	msxml.dll	5.0.2919.38

Tabelle 8.5:
Von Microsoft ver-
breitete Versionen
von MSXML

Tabelle 8.5:
Von Microsoft ver-
breitete Versionen
von MSXML
(Forts.)

Produktversion	Komponentendatei	Version der Komponentendatei
2.5a	msxml.dll	5.0.2919.6303
2.5	msxml.dll	5.0.2920.0
2.5 Service Pack 1 (SP1)	msxml.dll	8.0.5226
2.6 Beta1	msxml2.dll	7.50.4920.0
2.6 Beta2	msxml2.dll	8.0.5207.3
2.6	msxml2.dll	8.0.6518.1
3.0 Vorabversion März 2000	msxml3.dll	7.50.5108.0
3.0 Vorabversion Mai 2000	msxml3.dll	8.0.7309.3
3.0 Vorabversion Juli 2000	msxml3.dll	8.0.7520.1
3.0 Vorabversion September 2000	msxml3.dll	8.0.7722.0
3.0 Endfassung	msxml3.dll	8.0.7820.0
4.0 Technology Preview	msxml4.dll	4.0.8411.1

8.2.1 MSXML-Objektmodell

Zwölf Knotentypen

XML kennt zwölf verschiedene Knotentypen (vgl. Tabelle 8.6) und entsprechend zwölf verschiedene Klassen. Alle Knotenklassen erben von der Basisklasse `XMLDOMNode`, dazu gehört auch die Stammklasse `DOMDocument`. Die Unterklassen unterscheiden sich dadurch, dass sie neben der `XMLDOMNode`-Schnittstelle noch jeweils genau eine zusätzliche Schnittstelle implementieren. Die Schnittstelle `IXMLDOMNode` enthält den kleinsten gemeinsamen Nenner dieser zwölf Unterklassen. Auf die Attribute und Methoden, die bei allen Knotentypen vorhanden sind, kann somit auch über ein `XMLDOMNode`-Objekt polymorph zugegriffen werden. Für die zusätzlichen knotentypspezifischen Funktionen wird ein Zeiger auf eine der elf spezifischen Schnittstellen benötigt. Der Knotentyp wird über das Attribut `nodeType` angezeigt, wobei die in der Tabelle 8.6 gezeigten Werte und 0 für ein ungültiges Element möglich sind.

Automation nicht beeinträchtigt

Ein Automation-Client erhält eine `IDispatch`-Schnittstelle, die alle Mitglieder beider benutzerdefinierter Schnittstellen umfasst. Daher stehen auch einem Skript, das bekanntlich keine Schnittstellen wechseln kann, alle Funktionen zur Verfügung.

Hinweise für früh bindende Clients

Ein COM-Client, der frühes Binden (VTBL Binding) benutzt, kann mit einem Zeiger des Typs `IXMLDOMNode` zwar den kompletten Baum durchlaufen. Für den Zugriff auf knotentypspezifische Funktionen muss er jedoch anhand des Knotentyps (Attribut `nodeType`) den Schnittstellenzeiger wechseln.

Es gibt auch innerhalb der obigen knotentypspezifischen Klasse noch Vererbungsbeziehungen. So erbt etwa `XMLDOMCDATASection` von `XMLDOMText`. Außer-

XML Document Object Model (MSXML) Version 4.0

dem gibt es eine weitere Schnittstelle, `XMLDOMCharacterData`, die Basisfunktionen für die Klassen `XMLDOMComment`, `XMLDOMCDATASection` und `XMLDOMText` bereitstellt. Ein VTBL Binding Client kann eine Instanz von `XMLDOMCharacterData` nutzen, um polymorph auf die erbenenden Klassen zuzugreifen.

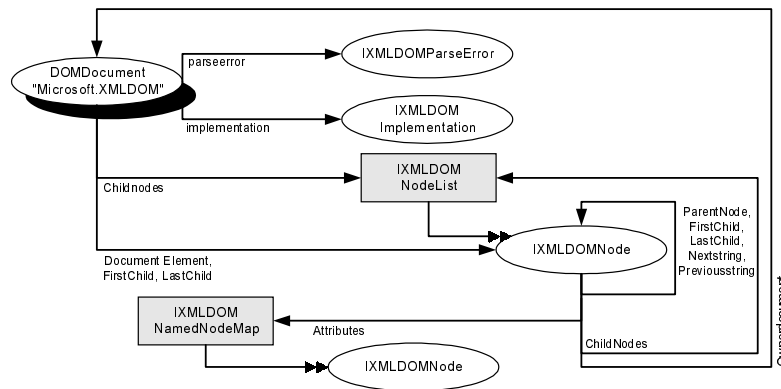


Abbildung 8.6:
MSXML-DOM-
Objektmodell

Klasse	Erläuterung
DOMDocument	Ein <code>DOMDocument</code> repräsentiert den im XML-Quelltext unsichtbaren Startknoten eines XML-Baums, der stets den Namen »document« und den gleichnamigen Knotentyp besitzt. Eine Umbenennung ist nicht möglich. <code>DOMDocument</code> stellt zahlreiche Basisfunktionen wie das Erzeugen neuer Knoten und das Laden und Speichern des Dokuments bereit. <code>DOMDocument</code> ist das Stammobjekt des Objektmodells und als einziges instanziiierbar. Die Klasse besitzt die ProgID »Microsoft.XMLDOM«. Jedes <code>DOMDocument</code> besitzt genau ein so genanntes <code>DocumentElement</code> , das nicht mit dem unsichtbaren Startknoten verwechselt werden darf. Das <code>DocumentElement</code> ist der sichtbare oberste Knoten eines XML-Dokuments.
XMLDOMParseError	Ein Objekt dieser Klasse ist über das Attribut <code>parseError</code> des <code>DOMDocument</code> erreichbar. <code>XMLDOMParseError</code> enthält detaillierte Informationen über den letzten Fehler, der beim Parsen aufgetreten ist.
XMLDOMImplementation	Diese Klasse stellt lediglich eine Methode bereit: Mit <code>HasFeature(feature, version)</code> kann überprüft werden, ob der Parser bestimmte Features unterstützt. In Version 1.0 unterstützt MSHTML die Features »XML«, »DOM« und »MS-DOM«. <code>Version</code> erwartet die Versionsnummer als String oder <code>NULL</code> für eine versionsnummernunabhängige Prüfung. In der MSDN Library wird fälschlicherweise dargestellt, dass diese Klasse nicht per Skript verwendet werden kann.

Tabelle 8.6:
Die wichtigsten
MSXML-Klassen

Document Object Model (DOM)

*Tabelle 8.6:
Die wichtigsten
MSXML-Klassen
(Forts.)*

Klasse	Erläuterung
XMLDOMNodeList	Ist eine Collection von XMLDOMNode-Objekten. Da diese Collection sich an dem DOM-Standard des W3C orientiert, entspricht sie nicht einer Standard-COM-/VB-Collection. ▶ Die Methode <code>Item()</code> gestattet nur den Zugriff über einen numerischen Index. ▶ Das Attribut zur Rückgabe der Anzahl der Elemente heißt <code>length</code> und nicht <code>count</code>. ▶ Zur Iteration werden die Methoden <code>nextNode()</code> und <code>reset()</code> unterstützt (nur vorwärts, nicht rückwärts; <code>reset()</code> setzt den Zeiger auf das erste Element zurück). Die Iteration über <code>For...Each</code> ist jedoch möglich.
XMLDOMNamedNodeMap	Ist eine spezielle Collection für Knoten des Typs <code>XMLDOMAttribute</code>, also für die Attribute von XML-Elementen. Unterstützt im Gegensatz zu <code>XMLDOMNodeList</code> den Zugriff auf enthaltene Attribute über ihren Namen (u. a. durch <code>getNamedItem(name)</code> und <code>setNamedItem(name)</code>).
XMLDOMNode	Repräsentiert einen Knoten in einem XML-Baum. Dabei sind viel mehr Dinge ein Knoten als nur die Elemente (Tags). Auch der Inhalt der Tags, die Attribute und das Dokument selbst sind Knoten (vgl. Liste der Knotentypen in Tabelle 8.7). Jedes Node-Objekt enthält zwei Collections: <code>childNodes</code> ist eine <code>XMLDOMNodeList</code>-Collection der untergeordneten Knoten. <code>Attributes</code> ist eine <code>XMLDOMNamedNodeMap</code> der Attribute des Knotens.

*Tabelle 8.7:
XML-Knotentypen
für die XML-
Elementtypen*

Klasse	Symbolische Konstante	Wert
DOMDocument	NODE_DOCUMENT	8
XMLDOMAttribute	NODE_ATTRIBUTE	2
XMLDOMCDATASection	NODE_CDATA_SECTION	4
XMLDOMComment	NODE_COMMENT	9
XMLDOMDocumentFragment	NODE_DOCUMENT_FRAGMENT	11
XMLDOMDocumentType	NODE_DOCUMENT_TYPE	10
XMLDOMElement	NODE_ELEMENT	1
XMLDOMEntity	NODE_ENTITY	6
XMLDOMEntityReference	NODE_ENTITY_REFERENCE	5
XMLDOMNotation	NODE_NOTATION	12
XMLDOMProcessingInstruction	NODE_PROCESSING_INSTRUCTION	7
XMLDOMText	NODE_TEXT	3

8.2.2 XML-Dateien für die Skripte

Die Skripte in den nachfolgenden Kapiteln arbeiten mit zwei XML-Dateien, die auf der Buch-CD mitgeliefert werden [CD:/umgebung/XML]:

- INTERNETDOMAIN.XML
- NTDOMAIN.XML

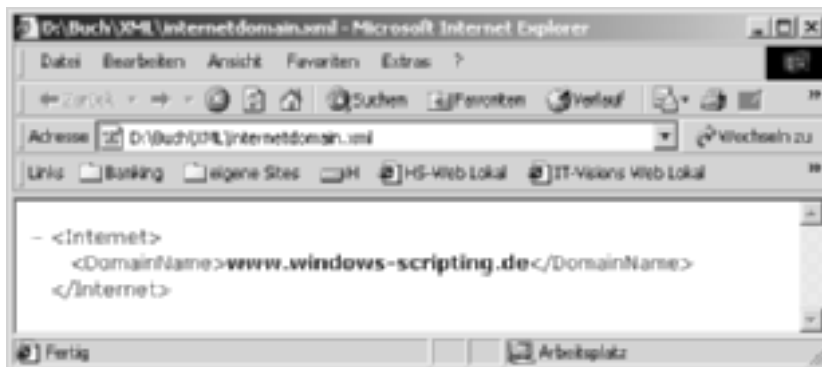


Abbildung 8.7:
internetdomain.xml

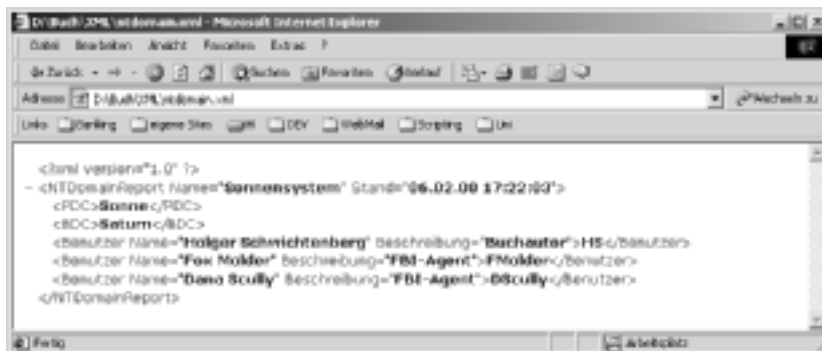


Abbildung 8.8:
ntdomain.xml

8.2.3 Lesezugriff auf XML-Dokumente

Das erste Beispiel testet die Features, die der XML-Parser unterstützt. Bitte beachten Sie, dass MSXML 3.0 die XML-Version 1.0 implementiert. Die Versionsnummer der Komponente hat nichts mit der Versionsnummer der XML-Sprache zu tun.

XML Feature-Test

```
Dim xdoc ' As MSXML.DOMDocument
Dim ximp 'As MSXML.IXMLDOMImplementation
Const VERSION = "1.0"
Set xdoc = CreateObject("Microsoft.XMLDOM")
Set ximp = xdoc.implementation
say "Unterstützte Eigenschaften:"
```

Document Object Model (DOM)

```
say "XML      : " & ximp.hasFeature("XML", VERSION)
say "DOM      : " & ximp.hasFeature("DOM", VERSION)
say "MS-DOM   : " & ximp.hasFeature("MS-DOM", VERSION)
```

Listing 8.7: Test auf von XML unterstützte Features [xml_test.wsf]

Die Ausgabe wird so aussehen:

Unterstützte Eigenschaften:

```
XML      : Wahr
DOM      : Wahr
MS-DOM   : Wahr
```

Beim Test auf Version 2.0 und 3.0 wird in allen Fällen »Falsch« ausgegeben.

Laden und Parsen

Load() und LoadXML()

Ein XML-Dokument kann auf drei Weisen in den Speicher geladen werden:

- ▶ aus einer externen Datei mit Hilfe der Methode `Load()`
- ▶ durch Übergabe von XML-Quelltextstrings an die Methode `LoadXML()`
- ▶ durch schrittweise Erzeugung des XML-Baums

Synchrones und asynchrones Laden

Standard im XML-DOM ist das asynchrone Laden. Nach der Ausführung von `Load()` wird die Kontrolle direkt an den Aufrufer zurückgegeben. Mit Hilfe des Attributs `readyState` kann der Zustand des Ladevorgangs überwacht werden. `ReadyState` durchläuft die Werte von 0 bis 4. Der Wert 4 bedeutet, dass das Dokument komplett geladen wurde.

```
Do While xdoc.readyState < 4
    say "Warte..." & xdoc.readyState
Loop
```

Async

Sie können dies vermeiden, indem Sie den Ladevorgang auf »synchron« setzen. Dazu müssen Sie dem Attribut `async` den Wert `False` zuweisen.

```
xdoc.async = False
```

Beispiel

Beispiel

Das Beispiel zeigt `Load()` und `LoadXML()`. Der Parser wird automatisch gestartet, so dass eine Fehlerauswertung über das `XMLDOMParseError`-Objekt möglich ist. Diese Aufgabe übernimmt die Funktion `xsayparseerror()`. MSXML bricht das Parsen nach dem ersten Fehler ab, daher kann auch immer nur ein Fehler ermittelt werden.



Bitte beachten Sie, dass im Fehlerfall das `DOMDocument`-Objekt keine Unter-elemente enthält, auch dann nicht, wenn das `documentElement` erfolgreich geparkt werden konnte. Die Fehlerausgabe kann daher im Fehlerfall nicht ermitteln, wie das Root-Element des Dokuments heißt.

Wenn eine Datei nicht gefunden werden kann, kommt es nicht zu einem Laufzeitfehler, sondern ebenfalls nur zu einem *ParseError*!

XML Document Object Model (MSXML) Version 4.0

```
' ### Gibt Fehler beim Parsen eines XML-Dokuments aus
Function xsayerror(xdoc) 'As MSXML.DOMDocument
Dim xerr ' MSXML.IXMLDOMParseError
Set xerr = xdoc.parseError
If xerr.reason = "" Then ' Kein Fehler
    say "Parsing '" & xdoc.documentElement.nodeName & "' OK!"
    xsayerror = 0
Else ' Fehler
    say "Parsing ERROR: " & xerr.errorCode & " (" & xerr.reason & ")
    in " & xerr.Line
    xsayerror = xerr.errorCode
End If
End Function

' === Laden und Parsen eines XML-Dokuments
Dim xdoc ' As MSXML.DOMDocument
Set xdoc = CreateObject("Microsoft.XMLDOM")
xdoc.async = False ' synchrones Laden!
' -- Laden eines XML-Dokuments aus einer Datei
xdoc.Load ("file:///d:/buch/xml/ntdomain.xml")
xsayerror xdoc
' -- Laden von Quelltext in den Parser
xdoc.loadXML (_
"<Internet><DomainName>www.windows-scripting.de</DomainName></
Internet>")
xsayerror xdoc
' -- Speichern des Quelltexts
xdoc.Save "file:///d:/buch/xml/internetdomain.xml"
' -- Laden von fehlerhaftem Quelltext in den Parser
xdoc.loadXML ("<Internet><Domainname>www.windows-scripting.de</
DomainName></Internet>")
xsayerror xdoc
```

Listing 8.8: Laden von XML-Quellcode in den XML-Parser [xml_load.wsf]

Die Ausgabe sollte folgendermaßen aussehen, denn XML ist sehr kleinlich.

```
Parsing 'dokument' OK!
Parsing 'NTDomain' OK!
Parsing ERROR: -1072896659 (Das Ende-Tag 'DomainName' stimmt nicht mit
dem Start-Tag 'Domainname' überein.) in 1
```

Listing 8.9: Ausgabe von xml_load.wsf

Speichern

Mit der Methode `Save()` kann das aktuell im `DOMDocument` geladene XML-Dokument in Form von XML-Quelltext in eine Datei gespeichert werden.

```
xdoc.Save Dateiname
```

Document Object Model (DOM)

DOM-Ereignisse

DOM-Events Das XML-DOM kennt zwei Ereignisse in der DOMDocument-Klasse:

- ▶ Ondataavailable() wird ausgelöst, wenn das XML-Dokument komplett zur Verfügung steht.
- ▶ Onreadystatechange() wird ausgelöst, wenn sich die Eigenschaft Ready State des DOMDocument-Objekts ändert.

Das folgende auf den WSH ausgelegte Beispiel zeigt die Verwendung:

```
' ### Eventhandler 1
Sub xdoc_onreadystatechange
msgbox "XML Readystate: " & xdoc.readystate
If xdoc.readystate = 4 Then WScript.Echo xdoc.documentElement.nodeName
& " ist fertig geladen!"
End Sub
' ### Eventhandler 2
Sub xdoc_ondataavailable
WScript.Echo "Data Available!"
End Sub
' === HAUPTPROGRAMM
' Laden und Parsen eines XML-Dokuments, Verbinden der Events
dim xdoc ' As MSXML.DOMDocument
Set xdoc = WScript.CreateObject("Microsoft.XMLDOM","XDOC_")
' -- Laden eines XML-Dokuments aus einer Datei
xdoc.async = False ' Synchrones Laden!
xdoc.Load ("file://d:\buch\xml\ntdomain.xml")
WScript.disconnectobject xdoc
Set xdoc = Nothing
msgbox "Ende"
```

Listing 8.10: XML-DOM Events im WSH abfangen [_xml-events.vbs]

Die XMLDOMNode-Klasse

XMLDOMNode Die XMLDOMNode-Klasse ist die zentrale Basisklasse. Auch die Klasse DOMDocument ist davon abgeleitet, so dass die in den folgenden Tabellen genannten Attribute und Methoden auch auf das DOMDocument anwendbar sind.

*Tabelle 8.8:
Attribute der
Node-Klasse*

Attribut	Rückgabebetyp	Erläuterung
nodeName	STRING	Name des Knotens
nodeValue	VARIANT	Daten des Knotens
nodeType	INTEGER	Knotentyp
nodeTypeString	STRING	Knotentyp in Stringform
parentNode	IXMLDOMNode	Zeiger auf Vaterknoten
childNodes	IXMLDOMNodeList	Collection der Kinderknoten

XML Document Object Model (MSXML) Version 4.0

Attribut	Rückgabotyp	Erläuterung
firstChild	IXMLDOMNode	Erster Kinderknoten
lastChild	IXMLDOMNode	Letzter Kinderknoten
previousSibling	IXMLDOMNode	Vorhergehender Knoten
nextSibling	IXMLDOMNode	Nachfolgender Knoten
attributes	IXMLDOMNamed NodeMap	Collection der Elementattribute
ownerDocument	IXMLDOMDocument	Zeiger auf das Dokument
text	STRING	Textliche Darstellung von Knoten und Inhalt
definition	IXMLDOMNode	Zeiger auf die Definition des Knotens in einer Document Type Definition (DTD) oder einem Schema
nodeTypedValue	VARIANT	Knotentyp
dataType	VARIANT	Datentyp des Knotens
xml	STRING	XML-Darstellung des Knotens und seiner Unterknoten
parsed	BOOLEAN	Zeigt an, ob der Teilbaum bereits geparkt wurde
namespaceURI	STRING	Uniform Resource Identifier (URI) des Namespaces, aus dem der Knoten stammt
prefix	STRING	Präfix des Namespaces
baseName	STRING	Knotenname ohne das Namespace-Präfix

*Tabelle 8.8:
Attribute der
Node-Klasse
(Forts.)*

Methode	Erläuterung
insertBefore (newChild As IXMLDOMNode, refChild As VARIANT) As IXMLDOMNode	Fügt einen Knoten als Kinderknoten vor einem bestimmten Knoten ein
replaceChild (newChild As IXMLDOMNode, oldChild As IXMLDOMNode) As IXMLDOMNode	Ersetzt einen Kinderknoten
removeChild (childNode As IXMLDOMNode) As IXMLDOMNode	Entfernt einen Kinderknoten
appendChild (newChild As IXMLDOMNode) As IXMLDOMNode	Fügt einen Knoten als Kinderknoten am Ende an
hasChildNodes () As BOOLEAN	Informiert, ob der Knoten Kinderknoten hat

*Tabelle 8.9:
Methoden der
Node-Klasse*

Document Object Model (DOM)

Tabelle 8.9:
Methoden der
Node-Klasse
(Forts.)

Methoden	Erläuterung
cloneNode (deep As BOOLEAN) As IXMLDOMNode	Erstellt eine Kopie des Knotens. Mit Deep=False wird das Kopieren der Kinderknoten verhindert.
transformNode (stylesheet As IXMLDOMNode) As STRING transformNodeToObject (stylesheet As IXMLDOMNode, output Object As VARIANT) As VT_VOID	Anwendung eines Stylesheets auf den Knoten
selectNodes (queryString As STRING) As IXMLDOMNodeList selectSingleNode (queryString As STRING) As IXMLDOMNode	Führt unter den Kinderknoten eine Suche auf Basis eines XSL-Patterns aus

Rekursiver Durchlauf

Für die Ausgabe eines XML-Baums bietet es sich wieder an, eine rekursive Prozedur zu schreiben:

```
Sub x_Reklist(xVater, ebene)
'xVater As MSXML.IXMLDOMNode, ebene As Integer
Dim xSohn 'As MSXML.IXMLDOMNode
Dim s ' As String
s = Space(ebene) & xVater.nodeName & " (" & xVater.nodeTypeString & _
")"
If xVater.nodeValue <> "" Then s = s & " = " & xVater.nodeValue
say s
' -- Enthält der Knoten Attribute?
If Not xVater.Attributes Is Nothing Then
' -- Iteration über alle Attribute
For Each xSohn In xVater.Attributes
x_Reklist xSohn, ebene + 1
Next
End If

' -- Enthält der Knoten Unterknoten?
If xVater.childNodes.length > 0 Then
' -- Iteration über alle Söhne
For Each xSohn In xVater.childNodes
x_Reklist xSohn, ebene + 1
Next
End If
End Sub
```

Listing 8.11: Hilfsroutine zur rekursiven Ausgabe eines XML-Dokuments

x_RekList() x_RekList() erwartet als Parameter einen Zeiger auf ein XMLNode-Objekt und einen Integer, der nur der Einrückung in den verschiedenen Rekursionsstu-

fen dient. Das Attribut `nodeTypeString` enthält den Knotentyp in Stringform, so dass dieses Objektmodell dem Entwickler erspart, eine Konstante selbst rückwandeln zu müssen. Die Prozedur wird innerhalb der Iteration über die `Attributes` und `childNodes` rekursiv aufgerufen. Leider gibt es hier einige Inkonsistenzen (möglicherweise Bugs) zu dem in COM üblichen Vorgehen:

- ▶ Knotentypen, die keine Attribute besitzen, haben keine `Attributes-Collection`. Der Zeiger ist *Nothing*. Das muss vor der Iteration geprüft werden. **Inkonsistenzen**
- ▶ Der Einstieg in eine leere `childNodes-Collection` mit `For...Each` führt zu einem Fehler.

```
' === Laden und Ausgeben eines XML-Dokuments
Dim xdoc ' As MSXML.DOMDocument
Dim ximp ' As MSXML.IXMLDOMImplementation
Set xdoc = CreateObject("Microsoft.XMLDOM")
' -- Laden eines XML-Dokuments aus einer Datei
xdoc.Load ("file://d:\buch\xml\ntdomain.xml")
' -- Rekursive Ausgabe wenn OK
If xsayerror(xdoc) = 0 Then
    say "Original-Quelltext:"
    say xdoc.xml
    say "Rekursiver Durchlauf:"
    x_Reklist xdoc, 0
End If
```

Listing 8.12: Laden und Ausgeben eines XML-Dokuments [xml_show.wsf]

Dabei wird im Vergleich zu der Ausgabe von `x_Reklist()` auch der über das Attribut `xml` zugreifbare XML-Quelltext ausgegeben.

```
-- Loading d:\buch\xml\ntdomain.xml
Parsing 'NTDomainReport' OK!
-- Original-Quelltext:
<?xml version="1.0"?>
<NTDomainReport Name="Sonnensystem" Stand="06.02.00 17:22:03">
    <PDC>Sonne</PDC>
    <BDC>Saturn</BDC>
    <Benutzer Name="Holger Schwichtenberg"
Beschreibung="Buchautor">HS</Benutzer>
</NTDomainReport>
-- Rekursiver Durchlauf:
#document (document)
xml (processinginstruction) = version="1.0"
version (attribute) = 1.0
#text (text) = 1.0
NTDomainReport (element)
Name (attribute) = Sonnensystem
#text (text) = Sonnensystem
```

Document Object Model (DOM)

```
Stand (attribute) = 06.02.00 17:22:03
#text (text) = 06.02.00 17:22:03
PDC (element)
#text (text) = Sonne
BDC (element)
#text (text) = Saturn
Benutzer (element)
Name (attribute) = Holger Schwichtenberg
#text (text) = Holger Schwichtenberg
Beschreibung (attribute) = Buchautor
#text (text) = Buchautor
#text (text) = HS
```

Listing 8.13: Ausgabe von xml_show.wsf auf Basis des auf der CD mitgelieferten XML-Dokuments ntdomain.xml

Selektion einzelner Elemente

Elementselektion

Natürlich wollen Sie nicht immer den kompletten Baum traversieren, um ein Element zu finden. Das DOMDocument-Objekt bietet daher mit `SelectNodes()` und `SelectSingleNode()` zwei Suchmethoden an. `SelectNodes()` findet alle Knoten, die einer bestimmten Pfadangabe entsprechen. `SelectSingleNode()` findet nur einen einzelnen Knoten. Gibt es mehrere Knoten, die der Pfadangabe entsprechen, so wird der erste genommen. Die Angabe des Pfads erfolgt in XSL Pattern Syntax (siehe [PAR00], S. 194ff.).

```
' === Knotenselektion
Sub xml_nodesel()
Dim xnode ' As MSXML.IXMLDOMNode
Dim xnodes ' As MSXML.IXMLDOMNodeList
' --- Auswahl eines Knoten
Set xdoc = CreateObject("Microsoft.XMLDOM")
xdoc.async = False ' synchrones Laden!
' --- Laden eines XML-Dokuments aus einer Datei
xdoc.Load ("file://d:\buch\xml\ntdomain.xml")
' --- Auswahl eines einzelnen Knotens
Set xnode = xdoc.selectSingleNode("NTDomainReport/PDC")
say xnode.nodeName & " = " & xnode.childNodes(0).nodeValue
' --- Auswahl einer Knotenmenge
Set xnodes = xdoc.selectNodes("*/Benutzer")
For Each xnode In xnodes
    say xnode.nodeName & " = " & xnode.childNodes(0).nodeValue
Next
End Sub
```

Listing 8.14: Knotenselektion [xml_nodesel.wsf]

```
PDC = Sonne
Benutzer = HS
```

Benutzer = FMolder

Benutzer = DScully

Listing 8.15: Ausgabe von *xml_nodels.wsf*

8.2.4 Erzeugen eines XML-Dokuments

Das Erzeugen eines Knotens innerhalb eines XML-Dokuments vollzieht sich in zwei Schritten, da die `XMLDOMNodeList`-Collection keine `Add()`-Methode bereitstellt.

- ▶ Zunächst muss eine Instanz der entsprechenden Knotentyp-Klasse erzeugt werden. Da die Knotentyp-Klassen nicht von außen instanzierbar sind, kann dies nur über eine Methode der Stammklasse erfolgen. `DOMDocument` bietet Methoden der Form `create{KnotentypName}()` an, also z.B. `CreateElement()` und `createProcessingInstruction()`. **CreateElement()**
- ▶ Danach muss das Element an die gewünschte Stelle in den Baum eingehängt werden. Dafür stehen die Methoden `AppendChild()` und `InsertBefore()` in der `Nodes`-Klasse zur Verfügung. `AppendChild()` fügt den neuen Knoten am Ende der Liste der Kinderknoten an. Bei `InsertBefore()` kann ein Kinderknoten angegeben werden, von dem aus gesehen links der neue Knoten angefügt werden soll. **AppendChild(),
InsertBefore()**

`x_add()` dient der Erzeugung neuer Elemente in einem bestimmten Dokument am Ende der Kinder eines übergebenen Vaters. Bitte beachten Sie die Fallunterscheidung für das Root-Element: Dies kann nicht durch `InsertBefore()` oder `AppendChild()` angefügt werden, sondern nur durch direkte Zuweisung an das Attribut `documentElement`. **x_add()**

```
Function x_add(xdoc, xparent, name, value)
' xdoc As MSXML.DOMDocument
' xparent As MSXML.IXMLDOMElement
' name As String, value As Variant
Dim xele ' As MSXML.IXMLDOMElement
' -- Unterelement erzeugen
Set xele = xdoc.createElement(name)
' -- Wert setzen
xele.text = value
' -- Element anfügen
If xdoc.documentElement Is Nothing Then ' root-Element?
    Set xdoc.documentElement = xele ' Ja
Else
    xparent.appendChild xele ' Nein
End If
Set x_add = xele
End Function
```

Listing 8.16: Hilfsroutine zur Erzeugung neuer XML-Elemente

Document Object Model (DOM)

Auf Basis von `x_add()` erzeugt das folgende Listing das XML-Dokument »NTDomainReport«, das Sie schon aus den vorherigen Beispielen kennen. Beachtung verdient nur noch die Erzeugung der Processing Instruction. Hier wird mit `InsertBefore()` sichergestellt, dass diese am Anfang des Dokuments steht.

```
Dim xdoc ' As MSXML.DOMDocument
Dim xparent ' As MSXML.IXMLDOMElement
Dim xchild ' As MSXML.IXMLDOMElement
Dim xpi ' As MSXML.IXMLDOMProcessingInstruction
Dim xele ' As MSXML.IXMLDOMElement
' -- Dokument erzeugen
Set xdoc = CreateObject("Microsoft.XMLDOM")
' -- Erzeuge ProcessingInstruction
Set pi = xdoc.createProcessingInstruction("xml", " version=""1.0""")
xdoc.InsertBefore pi, xdoc.childNodes.Item(0)
' -- Erzeuge Root-Element
Set xele = x_add(xdoc, xdoc, "NTDomainReport", "")
xele.setAttribute "Name", "Sonnensystem"
xele.setAttribute "Stand", Now
' -- Erzeuge Unterelemente
Set xparent = xele
Set xele = x_add(xdoc, xparent, "PDC", "Sonne")
Set xele = x_add(xdoc, xparent, "BDC", "Saturn")
Set xele = x_add(xdoc, xparent, "Benutzer", "HS")
xele.setAttribute "Name", "Holger Schwichtenberg"
xele.setAttribute "Beschreibung", "Buchautor"
say xdoc.xml
```

Listing 8.17: Dynamische Erzeugung des XML-Dokuments NTDomainReport
[xml_dynCreate.wsf]



Bitte beachten Sie die Möglichkeit, mit Hilfe der *ActiveX Data Objects* beliebige Datenquellen in XML-Form umzuwandeln, zu speichern bzw. entsprechend strukturierte XML-Dateien in Form eines Recordsets zu laden (siehe Kapitel 9.5).

8.2.5 Aufruf des XSLT-Processors in MSXML

XSLT-Transformationen ausführen

Auch der XSLT-Processor zur Verarbeitung von XSLT-Dokumenten, der sich innerhalb der MSXML-Komponente befindet, kann per Programmcode aufgerufen werden. Dazu sind zwei `DOMDocument`-Objekte notwendig. Ein Objekt enthält das XML-Dokument, ein anderes das XSLT-Dokument. Die Transformation wird gestartet, wenn auf einem mit einem XML-Dokument gefüllten `DOMDocument`-Objekt die Methode `transformNode()` unter Angabe des Zeigers auf das XSLT-Dokument aufgerufen wird. Dabei werden auch eventuell in der XSLT-Datei enthaltene Active Scripts ausgeführt.

XML Document Object Model (MSXML) Version 4.0

```
' === Ausführung einer XSL-Transformation
Sub xml_transform()
Dim xdoc ' As MSXML.DOMDocument
Dim xstyle ' As MSXML.DOMDocument
Dim ergebnis ' As String
' --- Eingabedateien
Const XMLFILE = "D:\buch\xml\test.xml"
Const XSLFILE = "D:\buch\xml\test.xsl"
' --- Laden des XML-Dokuments
Set xdoc = CreateObject("Microsoft.XMLDOM")
xdoc.async = False ' synchron laden!
xdoc.Load ("file://" & XMLFILE)
' --- Laden des XSL-Dokuments
Set xstyle = CreateObject("Microsoft.XMLDOM")
xstyle.async = False ' synchron laden!
xstyle.Load ("file://" & XSLFILE)
' --- Transformation ausführen
ergebnis = xdoc.transformNode(xstyle)
say "Transformation beendet!"
' --- Ergebnis ausgeben
say ergebnis
End Sub
```

Listing 8.18: Ausführung einer XSL-Transformation [xml_transform.wsf]

Das obige Listing 8.18 verwendet die nachfolgende XML- und XSL-Datei.

```
<?xml version="1.0" ?>
<?xml:stylesheet type="text/xsl" href="test.xsl" ?>
<test>
</test>
```

Eingabedateien

Listing 8.19: Eingabedatei test.xml

```
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">

<!-- Globales Skript -->
<xsl:script xmlns:xsl="http://www.w3.org/TR/WD-xsl"
language="VBScript">
<![CDATA[
start = now
function add(x,y)
add = x+y
end function
]]>
</xsl:script>

<!-- Transformation -->
```

XSL-Datei

Document Object Model (DOM)

```
<xsl:template match="/">
<HTML>
Dieses Dokument wurde transformiert am/um:
<xsl:eval language="vbscript">start</xsl:eval>
<hr/>
Eine einfache Rechnung:
Add(1,2) =
<xsl:eval language="vbscript">add(1,2)</xsl:eval>
</HTML>
</xsl:template>

</xsl:stylesheet>
```

Listing 8.20: Eingabedatei test.xsl

Ergebnis Das Ergebnis der Transformation sollte der folgende wohlgeformte HTML-Code sein:

```
<HTML>
Dieses Dokument wurde transformiert am/um:
07.05.2001 13:12:20
<hr />
Eine einfache Rechnung:
Add(1,2) =
3
</HTML>
```

Listing 8.21: Ausgabe von xml_transform.wsf

9 ActiveX Data Objects (ADO)

Der Zugriff auf Datenbanken und andere Datenquellen ist eine wichtige Funktion im Rahmen der skriptbasierten Administration. Zum einen sind es teilweise die Datenbestände in Datenbanken selbst, die per Skript verändert werden sollen, zum anderen sind Datenbanken oft die Speicher für von Skripten benötigte Informationen.

Datenbankzugriff

Microsofts aktuelle COM-Komponente für den Zugriff auf Datenbanken und andere Datenquellen sind die *ActiveX Data Objects (ADO)*. Dabei ist ADO Teil des Redmonder Konzepts *Universal Data Access (UDA)*. Zum besseren Verständnis von ADO ist zunächst eine Betrachtung der ADO zu Grunde liegenden OLE DB-Architektur notwendig.

9.1 Von ODBC zu OLE DB

Datenbankmanagementsysteme (DBMS) unterscheiden sich im Allgemeinen sehr stark hinsichtlich ihrer Programmierschnittstellen. Daraus resultiert das Streben nach einem Treiberkonzept für DBMS. Analog zu Treibern für Hardware wie Grafikkarten, Soundkarten, Drucker etc. soll jede Anwendung in der Lage sein, mit jeder Datenbank zusammenzuarbeiten. Einige von Ihnen mögen sich noch an die Zeit erinnern, als Treiber für Hardware in den Anwendungsprogrammen integriert waren. Jede Anwendung hatte ihre eigene Treibersammlung. Heute wird durch das Windows-Betriebssystem eine Abstraktionsschicht bereitgestellt. Im Idealfall wird für jeden Hardwarebaustein nur ein Treiber installiert. Die Anwendungen rufen Funktionen des Betriebssystems auf, die mit Hilfe des installierten Treibers auf die spezifischen Schnittstellen der einzelnen Geräte umgesetzt werden.

DBMS

Ein Treiberkonzept mit einer derartigen Abstraktionsschicht gibt es in der Windows-Welt bereits seit langem unter dem Namen *Open Database Connectivity (ODBC)*. Anwendungen nutzen für den Zugriff auf Datenbanken die ODBC-Programmierschnittstelle, ohne sich um die Charakteristika der einzelnen Datenbankmanagementsysteme sorgen zu müssen.

ODBC

OLE DB

ODBC hat sich aber über die Jahre als zu langsam und zu unflexibel erwiesen. Die Ablösung für ODBC heißt OLE DB. DB steht für »Database« und OLE für »Object Linking and Embedding«, was zunächst einmal in diesem Zusammenhang verwunderlich ist. Der Name weist darauf hin, dass die Technik schon Mitte der 90er Jahre entwickelt wurde. Damals war OLE noch der aktuelle Marketingbegriff für Microsofts Komponententechnologie. Etwas später geboren hätte das Kind dann wohl den Namen »COM DB« bekommen. Aktuell in Mode wäre »ActiveX DB«.

OLE DB

Mögliche Datenquellen OLE DB ist flexibler als ODBC und ermöglicht auch den Zugriff auf Datenquellen, die in relationaler Form angeordnet sind. Als OLE DB-Datenquellen kommen beispielsweise in Frage:

- ▶ serverbasierte Datenbankmanagementsysteme (z.B. Oracle, Informix, Microsoft SQL Server)
- ▶ Datenbankdateien (z.B. Access, dBase)
- ▶ Textdateien (z.B. CSV, HTML-Tabellen)
- ▶ Tabellenkalkulationen (z.B. Excel)
- ▶ Verzeichnisdienste (z.B. LDAP, WinNT, Novell Directory Service)
- ▶ Mail-Systeme (z.B. Microsoft Exchange, Lotus Notes)

Strukturierte versus semi-strukturierte Datenquellen

Strukturierte versus semi-strukturierte Datenquellen OLE DB ist in der Lage, sowohl auf strukturierte als auch auf semi-strukturierte Datenquellen zuzugreifen. Während bei strukturierten Datenquellen ein einzelner Datensatz nicht von einer vorgegebenen Feldstruktur abweichen kann, verfügen bei semi-strukturierten Datenquellen nicht alle Datensätze über die gleichen Felder. Ein Beispiel für eine strukturierte Datenquelle ist eine relationale Datenbank, in der alle Datensätze einer Tabelle (Relation) die gleichen Felder besitzen. Beispiele für semi-strukturierte Datenquellen sind Message Stores und Verzeichnisdienste. Semi-strukturierte Datenquellen sind oft in Form von Objektmengen modelliert.

OLE DB selbst verarbeitet Daten immer in Form einer strukturierten Relation. Beim Zugriff auf semi-strukturierte Datenquellen bleiben also Felder in der Ergebnistabelle leer.

OLE DB-Architektur

ODBC und OLE DB haben gemein, dass ihre Programmierschnittstellen komplex und systemnah sind. Es besteht daher Bedarf nach einer einfachen, objektbasierten Programmierschnittstelle auf Basis der komplexen OLE DB-Schnittstellen. Dies ist die Rolle der ActiveX Data Objects (ADO).

Die Gesamtarchitektur zeigt die folgende Abbildung. Die Rolle der ODBC-Treiber haben OLE DB-Provider eingenommen.

OLE DB-Bausteine Die OLE DB-Architektur kennt folgende Bausteine:

- ▶ **OLE DB Data Provider (Datenanbieter)** implementieren den Zugriff auf Datenquellen.
- ▶ **OLE DB Service Provider (Dienstanbieter)** erweitern die Funktionalität von OLE DB.
- ▶ **OLE DB Consumer (Konsumenten)** sind Anwendungen, die OLE DB nutzen. Dabei ist aus der Sicht der OLE DB-Schicht auch ADO ein Konsument.

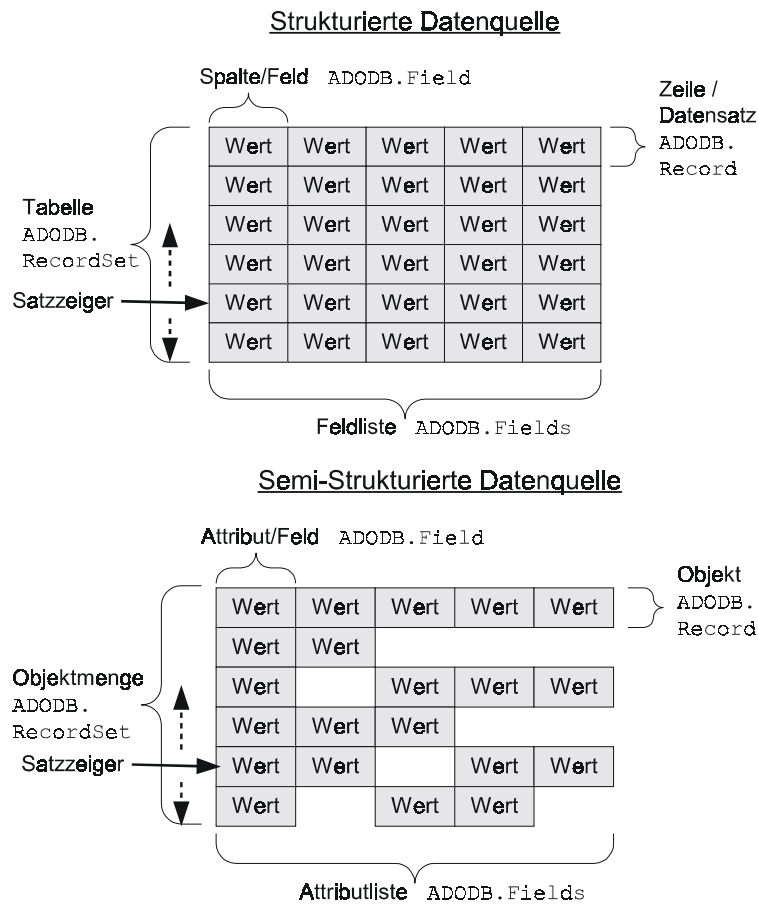


Abbildung 9.1:
Strukturierte
versus
semi-strukturierte
Datenquellen

Die in Abbildung 9.1 genannten Klassennamen (ADODB.xy) beziehen sich auf die in diesem Kapitel erläuterten ADO-Klassen.

Verfügbare OLE DB Data Provider von Microsoft

Microsoft liefert folgende Data Provider:

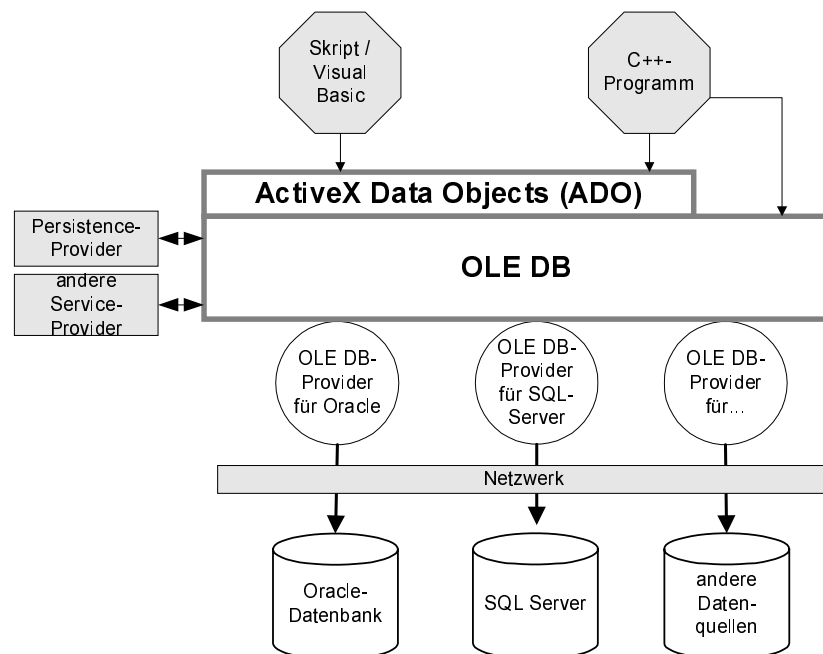
- ▶ *Microsoft OLE DB Provider for ODBC*: Zugriff auf ODBC-Treiber. Dieser Treiber wird auch »OLE DB-ODBC-Bridge« genannt.
- ▶ *OLE DB Provider for Microsoft Jet*: Zugriff auf JET-Datenbanken und ISAM-Treiber
- ▶ *Microsoft OLE DB Provider for Oracle*: Zugriff auf Oracle-Datenbanken
- ▶ *Microsoft OLE DB Provider for SQL Server*: Zugriff auf Microsoft SQL Server-Datenbanken
- ▶ *Microsoft OLE DB Provider for Internet Publishing (MDAIPP)*: Zugriff auf Webordner und den Exchange Server 2000-Webstore

**Data Provider von
Microsoft**

ActiveX Data Objects (ADO)

- *Microsoft OLE DB Provider for Microsoft Indexing Service*: Zugriff auf das Dateisystem und den Index Server (nur Lesezugriff)
- *Microsoft OLE DB Provider for Microsoft Active Directory Service*: Zugriff auf Verzeichnisdienste. Beispiele dazu finden Sie im Kapitel zu ADSI (Kapitel 3).
- *SQL Server DTS Flat File OLE DB Provider*: Zugriff auf Textdateien
- *Microsoft OLE DB Provider for DTS Packages*: Zugriff auf Pakete des Data Transformation Service
- *Exchange OLE DB Provider*: Zugriff auf den Exchange Server 2000-Webstore

Abbildung 9.2:
Architektur von
OLE DB und ADO



OLE DB-Provider für ODBC

Da noch nicht für alle Datenbankmanagementsysteme OLE DB-Provider zur Verfügung stehen, gibt es einen OLE DB-Provider für ODBC. Damit können auch Datenquellen verwendet werden, für die nur ein ODBC-Treiber vorhanden ist. Da die Daten auf diesem Wege viele Schichten durchqueren müssen, ist die Performance nicht gut. Suchen Sie also immer erst einen »native« OLE DB-Provider für Ihre Datenquelle, bevor Sie auf den Umweg über ODBC zurückgreifen.

Andere OLE DB Data Provider

Da Microsoft die OLE DB-Architektur und die dafür definierten COM-Schnittstellen veröffentlicht hat, gibt es auch von anderen Anbietern OLE DB Data- und Service-Provider:

- ▶ ACES [ACE00] bietet einen OLE DB-Provider für NT-Ereignisprotokolle (siehe Kapitel 12.2).
- ▶ Im Rahmen des Produkts FTPX liefert die Firma Mabry [MAR00] einen OLE DB-Provider für den Zugriff auf FTP-Server (siehe Kapitel 21.4).
- ▶ ISG Navigator liefert OLE DB-Provider für Oracle, Sybase, Informix, SQL Server, DB2, Rdb, Ingres, Red Brick, Non-Stop SQL/MP, Non-Stop SQL/MX [ISG00].
- ▶ IBM bietet einen Provider für DB2/400 AS/400 an [IBM00].
- ▶ Für die objektorientierte Datenbank ObjectStore kommt der Provider vom Hersteller, der Firma Object Design [ODI00].
- ▶ Die Firma Merant (aus der Fusion InterSolv und MicroFocus entstanden) bietet im Rahmen des Produkts Connect OLE DB bzw. Connect ADO Provider für Informix, Oracle, Sybase Adaptive Server, Sybase System 10/11, SAP R/3, XML und Lotus Notes an [MER00].

Diese Liste erhebt keinen Anspruch auf Vollständigkeit, da es ständig neue Produkte gibt. Es gibt es eine »unabhängige« (von der Firma Merant gesponserte) OLE DB-Site [ODB00b].

Das Schreiben eigener OLE DB-Provider ist nicht schwer und auch mit Visual Basic 6 möglich, siehe MSDN Library.



**Provider von
anderen
Anbietern**



9.2 Einführung in die ActiveX Data Objects (ADO)

Die Idee, auf Basis der eigentlichen Datenbankschnittstelle eine einfachere Programmierschnittstelle zur Verfügung zu stellen, ist nicht neu. Mit den Remote Data Objects (RDO) und den Data Access Objects (DAO) gab es einen solchen Ansatz auch schon für ODBC. ADO vereinheitlicht diese bestehenden Ansätze und ermöglicht die Nutzung spezifischer OLE DB-Funktionen. Aber auch der Zugriff auf ODBC ist weiterhin möglich.

ADO ist eine automationsfähige COM-Komponente und daher insofern programmiersprachenunabhängig, als alle COM-fähigen Sprachen ADO nutzen können. Die COM-Schnittstellen von OLE DB sind dagegen nicht COM-automationsfähig und können auf Grund verschiedener komplexer Abläufe nur von C++ aus komplett genutzt werden. ADO besitzt aber Zugriff auf alle OLE DB-Funktionen. Es besteht also kein Grund, OLE DB lieber direkt programmieren zu wollen.

**ADO versus
OLE DB**

ActiveX Data Objects (ADO)

Tabelle 9.1:
Schnellinfo ActiveX
Data Objects
(ADO)

Name und Abkürzung	ActiveX Data Objects (ADO)
Name der Komponentendatei	msado15.dll
Interner Name der Typbibliothek	ADODB
Helpstring der Typbibliothek	Microsoft ActiveX Data Objects 2.5 Library
Hersteller	Microsoft
Lizenzierung	Bestandteil diverser Produkte; auch als einzelnes kostenloses Add-on verfügbar
Besprochene Version	2.70.7012.8
NT4, Windows 95/98/ME Windows ME und Windows2000	Installation von MDAC 2.7 [CD:/Install/komponenten/ADO] oder [ODB00a]
Whistler (Windows XP / Windows.NET)	Version 2.7 enthalten
Dokumentation	siehe Microsoft MSDN Library [MSL00]

Installation und Versionen

MDAC *Microsoft Data Access Components* (MDAC) ist der Name des Installationspakets, mit dem OLE DB und ADO installiert werden. ADO wurde erstmalig mit den Active Server Pages im Internet Information Server 3.0 ausgeliefert mit dem Einsatzgebiet der datenbankbasierten Webanwendungen. Eine neuere Version von ADO wurde mit dem SQL Server 7.0 ausgeliefert. Microsoft hatte zahlreiche Probleme mit früheren Versionen. MDAC 1.5 war nur eingeschränkt Jahr-2000-kompatibel. Das »Jahr-2000-compliant-Release« MDAC 2.0 enthielt viele Fehler und wurde schnell durch zwei Service-Packs ergänzt und schließlich durch MDAC 2.1 abgelöst, die mit Office 2000 und dem Internet Explorer ausgeliefert wurden. Doch auch mit MDAC 2.1 gab es Probleme: So ist ADO 2.1 nicht kompatibel mit dem Active Directory Service Interface (ADSI) in der Version 2.0. ADO 2.1 benötigt ein Update auf ADSI 2.5.



MDAC ist in der Version 2.5 Bestandteil von Windows2000 und für die anderen 32-Bit-Windows-Versionen als optionales Setup erhältlich. Sie finden das Installationspaket MDAC 2.5 auf der Buch-CD [CD:/install/komponenten/ado/mdac/].



Benutzen Sie für den Zugriff auf Access97-Datenbanken nicht den Provider *Microsoft.Jet.oledb* in Version 4.0, sondern die Vorgängerversion 3.51. Die Version 4.0 ist auf Access2000 ausgelegt und in Verbindung mit früheren Access-Versionen zu langsam. Da *Microsoft.Jet.OLEDB.3.51* seit MDAC 2.1 nicht mehr Bestandteil von MDAC ist, müssen Sie erst MDAC 2.0 und danach eine neuere Version installieren.

Ermittlung der MDAC-Versionen

Sie erkennen die installierte MDAC-Version an der Versionsnummer der MSDADC.DLL. Die folgende Tabelle zeigt die Versionsnummern zu den am weitesten verbreiteten MDAC-Versionen.

msdadc.dll

MDAC-Version	Versionsnummer der msdadc.dll
MDAC 1.5c	1.50.3506.00
MDAC 2.0	2.00.3002.4
MDAC 2.0 SP1 / SP2	2.00.3002.23
MDAC 2.1.0 (SQL Server 7)	2.10.3513.0
MDAC 2.1.1 (Internet Explorer 5)	2.10.3711.2
MDAC 2.1.2 (MDAC 2.1 SP2)	2.10.4202.0
MDAC 2.5 Beta (Windows2000 RC1)	2.50.3719.9
MDAC 2.5 (Windows2000)	2.50.4403.0
MDAC 2.6 (Add-on)	2.60.6526.0

Tabelle 9.2:
Tabelle zur Ermittlung der MDAC-Version anhand der Version der msdadc.dll

Der MDAC Component Checker ist ein kostenloses Zusatzwerkzeug, das detaillierte Informationen über die installierte MDAC-Version zusammenstellt. Der Component Checker überprüft alle installierten MDAC-DLLs und liefert auf mehreren Auswertungsseiten Informationen über deren Versionsstand. Sie finden den MDAC Component Checker auf der Buch-CD im Verzeichnis /INSTALL/ADO/COMPONENTCHECKER.

MDAC Component Checker

Mit Hilfe der in der eigenständigen COM-Komponente *MDACVer* implementierten Klasse *Version* können Sie die installierte MDAC-Version auch per Programmcode ermitteln.

MDACVer-Komponente

```
' Komponente MDACVer,odbconf.dll
' "Microsoft Data Access Components Installed Version"
Dim x 'As MDACVer.Version
Set x = CreateObject("MDACVer.Version")
say x.Major & "." & x.Minor & "." & x.Build
```

Listing 9.1: Ermittlung der MDAC-Version per Programmcode

ADO-Erweiterungen

Für ADO sind inzwischen verschiedene Erweiterungen verfügbar:

► Remote Data Service (RDS)

RDS ist ein Dienst zum Zugriff auf Datenbanken über das Hypertext Transfer Protocol (HTTP) und wurde geschaffen, um einen Datenaustausch zwischen Webserver und Webbrowser zu ermöglichen. Durch den Einsatz von RDS müssen Daten dem Browser nicht mehr in Form von auf dem Server generierten HTML-Seiten präsentiert werden, sondern die Daten können zum Browser übertragen und dort verarbeitet werden.

RDS

- ▶ Microsoft ADO Extensions for Data Definition and Security (ADOX)
ADOX ADOX ist eine Erweiterung von ADO, um ADO wieder auf die volle Funktionalität von DAO zu bringen. ADO wurde zunächst bewusst gegenüber DAO in der Funktionalität reduziert, da das Lesen und Schreiben von Daten im Vordergrund stand. Die Definition von Tabellen und deren Sicherheitseinstellungen kann bei Datenbankservern über die Ausführung von SQL Data Definition-Befehlen bzw. Stored Procedures ausgeführt werden. ADO bot zwar die Möglichkeit, diese Befehle auszuführen; Programmierer waren es jedoch aus DAO gewöhnt, dass auch das Objektmodell eigene Funktionalitäten dafür bereitstellte. Daten-Definitionen und Sicherheitseinstellungen sind in ADO durch die Benutzung von ADOX optional.
- ▶ ADO Multi Dimensional (ADOMD)
ADOMD ADOMD ist eine Erweiterung zum Zugriff auf multidimensionale Daten. Diese Funktion wird für OLAP (Online Analytical Processing) benötigt. Die Komponente ist implementiert in MSADOMD.DLL.

9.3 Beschreibung von Datenverbindungen

Informationen über Datenquellen

Um eine Verbindung zu einer Datenquelle aufzubauen, werden zahlreiche Informationen benötigt. Dies sind insbesondere:

- ▶ Name des Treibers bzw. Providers
- ▶ Name des (Datenbank-)Servers oder Pfads zu der Datei, die die Daten enthält
- ▶ nähere Beschreibung des Aufbewahrungsortes der gewünschten Informationen (z.B. Name der Datenbank auf einem Datenbank-Server)
- ▶ Benutzername und Kennwort für die Herstellung der Verbindung
- ▶ Einstellungen für den Ablauf der Verbindung (Zugriffsmodus, Puffergrößen, Transaktionssteuerung, Timeout etc)

Anzahl und Art der benötigten Informationen sind stark vom Treiber bzw. Provider abhängig. Es liegt nahe, diese Verbindungsinformationen außerhalb der Anwendungen zu speichern, um sie zentral verwalten zu können.

ODBC und OLE DB gehen dabei ähnliche, aber dennoch verschiedene Wege.

- ▶ ODBC kennt drei Formen so genannter *Data Source Names (DSNs)*, die in der Registry oder dem Dateisystem gespeichert werden.
- ▶ OLE DB verwendet so genannte *Microsoft Datenlink-Dateien*, die im Dateisystem gespeichert werden.

9.3.1 ODBC-Data Source Names (DSN)

ODBC-DSNs werden über die Systemsteuerung unter dem Icon ODBC-DATENQUELLE (32-BIT) konfiguriert. Unter Windows2000 befindet sich dieses Werkzeug unter SYSTEMSTEUERUNG/VERWALTUNG/DATENQUELLEN (ODBC).

DSNs

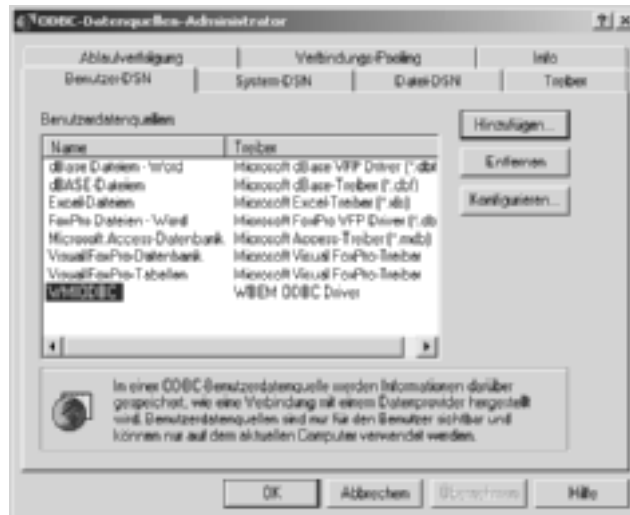


Abbildung 9.3:
ODBC-
Konfiguration

Es gibt drei Arten von DSNs:

- ▶ Eine **Benutzer-DSN** steht nur einem bestimmten Benutzerkonto zur Verfügung. Die Verbindungsinformationen werden daher in der Registry benutzerbezogen gespeichert. Die Benutzer-DSNs des aktuell angemeldeten Benutzers befinden sich unter HKEY_CURRENT_USER\SOFTWARE\ODBC\ODBC.INI.
- ▶ Datenquellen, die allen Benutzern eines Computers zur Verfügung stehen sollen, müssen als **System-DSN** abgespeichert werden. Diese Informationen liegen in der Registry unter HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBC.INI.
- ▶ Eine **Datei-DSN** wird nicht in der Registry, sondern in Form von Dateien gespeichert. Diese Dateien haben die Extension .DSN und werden in einem zentralen Verzeichnis gespeichert. DSN-Dateien sind einfache ASCII-Textdateien mit der Struktur einer INI-Datei. Den Namen dieses Verzeichnisses finden Sie in der Registry unter HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBC.INI\ODBC FILE DSN\DEFAULTDSNDIR.

Benutzer-DSN
(User-DSN)

System-DSN

Datei-DSN
(File-DSN)

Unter Windows2000 können Sie im ODBC-DATENQUELLE-ADMINISTRATOR (Systemsteuerung) mit Hilfe eines Dateidialogs den Pfad auswählen.

9.3.2 OLE DB Data Links

OLE DB Data Links

Während ODBC-Verbindungen in der Windows-Systemsteuerung definiert wurden, können Verbindungen zu OLE DB-Datenquellen überall im Dateisystem in Form von so genannten Microsoft Datenlink-Dateien definiert werden. Datenlink-Dateien haben die Dateierweiterung .UDL. Sie können mit Hilfe ihrer Dateieigenschaften verändert werden. Während es in manchen neueren Konfigurationen unter NT4 und Windows 95/98/ME einen Standardeintrag »NEUE DATA LINK DATEI« im Kontextmenü des Filebrowsers gab, fehlt unter Windows2000 dieser Eintrag. Legen Sie hier einfach zunächst eine reine Textdatei an und ändern Sie dann für diese die Extension in .UDL.

Abbildung 9.4:
Auswahl des OLE
DB-Providers in
einer UDL-Datei
und Spezifikation
einer Verbindung zu
einer Access- bzw.
SQL Server-
Datenbank



Die Speicherung erfolgt in Unicode-Textdateien, die auch direkt mit einem Unicode-fähigen Texteditor bearbeitet werden können. Eine UDL-Datei enthält die Spezifikation der Datenquelle in Form eines so genannten *Connection-Strings*.

Beschreibung von Datenverbindungen

```
[oledb]
; Everything after this line is an OLE DB initstring
Provider=Microsoft.Jet.OLEDB.3.51;Persist Security Info=False;User
ID=admin;Data Source=D:\buch\data\meier.mdb
```

Sie sollten eine UDL-Datei nur dann direkt mit einem Texteditor manipulieren, wenn Sie mit der Bedeutung der einzelnen Einstellungen gut vertraut sind. Sie müssen bei diesem Vorgehen auf die Konsistenz der Einstellungen selbst achten, was normalerweise das UDL-GUI für Sie erledigt.

In UDL- und DSN-Dateien gespeicherte Kennwörter werden nicht verschlüsselt!



9.3.3 Verbindungszeichenfolgen (Connection Strings)

Eine Anwendung hat grundsätzlich drei Möglichkeiten, die Verbindungsinformationen zu speichern:

1. Eine Anwendung kann auf die in Form von ODBC-DSNs oder OLE DB-Datenlink-Dateien gespeicherten Verbindungsinformationen zurückgreifen.
2. Eine Anwendung kann die kompletten Verbindungsinformationen im Quellcode enthalten.
3. Eine Anwendung kann die Verbindungsinformationen in einer anderen Quelle speichern (z.B. in der Registry oder einer INI-Datei).

Aus Sicht von ADO gibt es keinen Unterschied zwischen den Möglichkeiten 2 und 3. Der dritte Fall wird nicht durch besondere Funktionen unterstützt (warum auch, denn Möglichkeit 1 ist in der Regel ausreichend). Wenn der Programmierer von Möglichkeit 1 keinen Gebrauch macht, dann ist es ADO egal, woher er seine Verbindungsinformationen holt. Die erwartete Form ist die gleiche.

ADO erwartet einen so genannten *Connection String* (auch *Init-String* genannt). Ein Connection-String ist eine Reihe von Attribut-Wert-Zuweisungen.

- Attribut und Wert werden durch ein Gleichheitszeichen getrennt.
- Ein Attribut-Wert-Paar muss mit einem Semikolon beendet werden.

Die allgemeine Darstellung eines Connection-Strings ist also:

```
"Attribut1=Wert1;Attribut2=Wert2;...;AttributN=WertN;"
```

**Speicherung von
Verbindungs-
informationen
in Anwendungen**

ActiveX Data Objects (ADO)

ADO unterstützt insgesamt vier Formen von Connection Strings:

Tabelle 9.3:
Formen von
Connection-Strings

	zu Möglichkeit 1	zu Möglichkeiten 2+3
OLE DB	Der Connection-String enthält lediglich einen Verweis auf eine UDL-Datei, in der die nötigen Informationen abgelegt sind.	OLE DB-Connection-String, der alle nötigen Informationen enthält
ODBC	ODBC-Connection-String, der auf einen Data Source Name (DSN)-Eintrag verweist	ODBC-Connection-String, der alle nötigen Informationen enthält

Verweise

UDL Der einfachste Fall ist der Verweis auf eine UDL-Datei:

```
"File Name=\\pfad\\datenlinkname.udl;"
```

DSN Ähnlich ist der Verweis auf eine DSN bei ODBC, wobei hier seit ADO 2.1 die Angaben für UserID und Passwort Pflicht sind:

```
"Provider=MSDASQL;  
DSN=dsnname;UID=username;PWD=kennwort;"
```

Datei-DSN Beim Verweis auf eine Datei-DSN muss das erste Attribut heißen:

```
"Provider=MSDASQL;  
FILEDSN=\\pfad\\dsndatei.dsn;UID=username;PWD=kennwort;"
```

Dabei ist die Angabe `Provider=MSDASQL` optional, denn immer dann, wenn das Provider-Attribut nicht oder auf einen Leerstring gesetzt wird, wird der OLE DB-Provider für ODBC verwendet.

Attribute

Connection-Strings ohne ODBC-DSN oder OLE DB-Datalink

Die Connection Strings ohne Verweis auf eine ODBC-DSN oder eine OLE DB-Datalink-Datei (Möglichkeiten 2+3) sind natürlich wesentlich komplexer. In den Connection-Strings werden insbesondere folgende Attribute verwendet:

- ▶ **PROVIDER** enthält den Namen des OLE DB-Providers. Fehlt diese Angabe, wird der OLE DB-Provider für ODBC verwendet.
- ▶ **DRIVER** spezifiziert bei Verwendung des OLE DB-Providers für ODBC den ODBC-Treiber.
- ▶ **DATA SOURCE** enthält bei OLE DB den Namen des anzusprechenden Datenbankservers oder bei Datenbankdateien den Dateinamen. Bei ODBC steht an dieser Stelle **SERVER** (bei Datenbankservern) bzw. **DBQ** (bei Datenbankdateien).
- ▶ **INITIAL CATALOG** steht bei Datenbankservern für die gewünschte Datenbank. Bei ODBC heißt das Attribut **DATABASE**.

Beschreibung von Datenverbindungen

- USER ID (in ODBC: UID) und PASSWORD (in ODBC: PWD) enthalten Benutzername und Kennwort.

Die nachfolgende Tabelle zeigt einige Beispiele für Connection Strings.

Provider	Beispiel für einen Connection-String
Microsoft Access	Provider=Microsoft.Jet.OLEDB.4.0; User ID=hs; Password=egal; Persist Security Info=True; Data Source=D:\buch\data\katalog.mdb;
Microsoft Access mit dezidiertem Workgroup-Datei	Provider=Microsoft.Jet.OLEDB.4.0; Persist Security Info=False; User ID=admin; Data Source=D:\buch\data\katalog.mdb; Jet OLEDB:System database=d:\buch\data\katalog.mda
Excel-Datei	Provider=Microsoft.Jet.OLEDB.4.0; Data Source=D:\buch\data\preise.xls; Extended Properties="Excel 8.0;HDR=Yes«;
Text-Datei über ODBC	Driver={Microsoft Text Driver (*.txt; *.csv)}; Dbq=D:\buch\data\; Extensions=asc,csv,tab,txt;
SQL Server über SQL Server-Sicherheit	Provider=SQLOLEDB.1; User ID=sa; Password=egal; Initial Catalog=IT-Visions; Data Source=SONNE2000;
SQL Server über NT-Sicherheit	Provider=SQLOLEDB.1; Integrated Security=SSPI; Initial Catalog=IT-Visions; Data Source=SONNE2000;
Oracle	Provider=MSDAORA; Data Source=serverName; User ID=username; Password=userPassword;
Konfigurierte ODBC-DSN	Provider=MSDASQL; DSN=dsnname;UID=username;PWD=kennwort;
Über einen ODBC-Treiber	Provider=MSDASQL;DRIVER=driver; SERVER=server; DATABASE=database; UID=user; PWD=password«;
OLE DB-Provider für Active Directory	Provider=ADsDSOObject; Password=ds9; User ID=administrator; Encrypt Password=False;

*Tabelle 9.4:
Beispiele für Connection-Strings für unterschiedliche Datenquellen und OLE DB-Provider*



An dieser Stelle sollen nur einige Beispiele gegeben werden, denn es ist nicht notwendig, die komplette Syntax zu kennen: Sie können die Systemsteuerung bzw. die grafische Benutzeroberfläche (GUI) zu den Datenlink-Dateien benutzen, um sich die Connection-Strings generieren zu lassen.

- ▶ Erstellen Sie eine UDL-Datei mit dem Windows Explorer, öffnen Sie die Datei anschließend in einem Unicode-fähigen Texteditor und kopieren Sie die dritte Zeile über die Zwischenablage in Ihren Quellcode.
- ▶ Bei ODBC ist es nur ein klein wenig mehr Arbeit: Erstellen Sie eine Datei-DSN und öffnen Sie diese in einem Texteditor. Verbinden Sie alle Attribut-Wert-Paare innerhalb der Sektion [ODBC] zu einer Zeile, indem Sie einfach den Zeilenumbruch durch ein Semikolon ersetzen. Wenn in einem Wert Leerzeichen vorkommen, dann setzen Sie den Wert in geschweifte Klammern.

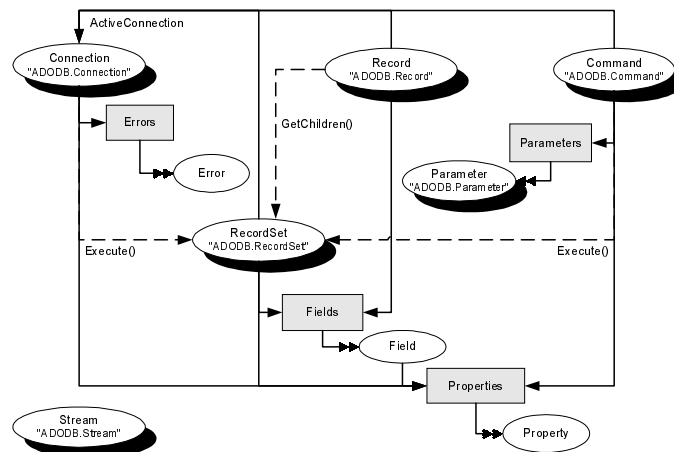
9.4 Das ADO-Objektmodell

Das ADO-Objektmodell besitzt kein eindeutiges Stammobjekt, sondern sechs eigenständig instanziiierbare Klassen mit einigen abhängigen Klassen.

Flexibilität und Erweiterbarkeit

Der Vertragscharakter einer Schnittstelle wird bei ADO insofern verletzt, als nicht alle in den ADO-Schnittstellen definierten Methoden und Attribute von allen OLE DB-Providern auch wirklich unterstützt werden. Die Benutzung nicht im OLE DB-Provider vorhandener Attribute oder Methoden löst einen Fehler aus. Einige Provider unterstützen zusätzliche Attribute. ADO ist daher ein Meta-Objektmodell auf Attributebene.

Abbildung 9.5:
ADO-Objektmodell



Das ADO-Objektmodell

Connection	Connection repräsentiert eine Verbindung zu einer Datenquelle.
Errors Error	Eine Connection-Klasse enthält eine Collection der möglicherweise aufgetretenen Fehler bei einer Datenverbindung.
Recordset	Recordset repräsentiert eine Menge von Datensätzen, also eine Tabelle. Recordset ist ein cursororientierter Zugriff: Dabei ist jeweils nur eine Zeile im direkten Zugriff.
Record	Die Record-Klasse ist ein einzelnes Datenelement. Allerdings ist die Record-Klasse nicht – wie viele glauben – über eine Assoziation mit der Recordset-Klasse verbunden. Record wird weniger für den Zugriff auf relationale Datenbanken (z.B. SQL Server) benutzt als für den Zugriff auf Objekte in semi-strukturierten Datenquellen, die einzeln identifizierbar sind (z.B. Exchange 2000-Webstore).
Stream	Ein Stream-Objekt repräsentiert einen binären Datenstrom. Einzelne Datenelemente können in einen Datenstrom (engl. <i>Stream</i>) serialisiert werden, um sie persistent zu machen (z.B. im Dateisystem).
Fields Field	Diese Klasse enthält die einzelnen Felder (Spalten) einer Tabelle (in Verwendung mit Recordset) bzw. die Felder des Datenelements (in Verwendung mit Record).
Command	Command ist ein Befehl an die Datenquelle.
Parameters Parameter	Eine Command-Klasse enthält eine Parameters-Collection mit Parameter-Objekten. Jedes Parameter-Objekt repräsentiert Argumente für eine parametrisierte Abfrage oder eine Stored Procedure. Dabei können Werte in beide Richtungen übergeben werden. Das Parameter-Objekt kann zwar eigenständig erzeugt, aber nur in Verbindung mit dem Command-Objekt eingesetzt werden.
Properties Property	Diese Klassen ermöglichen den Zugriff auf providerspezifische Attribute und bilden ein Meta-Objektmodell auf Attributebene.

Es ist unter Programmierern üblich, die Elemente der Realwelt genauso zu bezeichnen wie die o. g. Klassen, also von Recordsets, Records, Commands, Streams und Connections zu sprechen.



Recordset, Connection und Command

Die wichtigsten Klassen, die von Anfang an zu ADO gehörten, sind Recordset, Connection und Command. Alle drei Klassen können eigenständig instanziiert werden. Es ist möglich, einen Recordset oder ein Command zu nutzen, ohne vorher explizit ein Connection-Objekt erzeugt zu haben. Recordset- und Command-Objekte sind selbst in der Lage, eine Verbindung zu einer Datenquelle zu erzeugen.

Meta-Objektmodell auf Attributebene

Properties-Collection	Viele der Klassen enthalten eine Collection <code>Properties</code> . ADO unterscheidet zwischen so genannten eingebauten Attributen und providerspezifischen Attributen.
Eingebaute Attribute	Eingebaute Attribute (<i>Built-In Properties</i>) sind die in der Klasse definierten Attribute, die in der Regel von jedem Provider implementiert werden. Diese Attribute sind im direkten Zugriff.
Providerspezifische Attribute	Darüber hinaus kann jeder Provider für die ADO-Hauptobjekte providerspezifische Attribute implementieren. Diese Attribute können nicht direkt angesprochen werden. Jedes providerspezifische Attribut wird durch ein Property-Objekt in der <code>Properties</code> -Collection des zugehörigen Hauptobjekts repräsentiert. Das Property-Objekt stellt die Attribute <code>Name</code> , <code>Type</code> , <code>Value</code> und <code>Attributes</code> bereit, um auf ein dynamisches Attribut zuzugreifen (mit <code>Attributes</code> sind die Attribute eines Attributs gemeint). Das <code>Properties</code> -Attribut, das den Zugriff auf die <code>Properties</code> -Collection liefert, ist in der allgemeinen Schnittstelle <code>_ADO</code> definiert, von der die meisten ADO-Schnittstellen abgeleitet sind.
Versionen 2.6 und 2.7	ADO hat seit ADO 2.5 nur kleine Änderungen erfahren. Die größte Änderung in ADO 2.6 war die verbesserte Unterstützung für Streams: Das <code>Command</code> -Objekt kann in ADO 2.6 Streams sowohl als Eingabe verarbeiten als auch einen Stream statt eines Recordsets (z.B. XML) als Ausgabe liefern. Außerdem kann mit der <code>Open()</code> -Methode des <code>Record</code> -Objekts ein Befehl gestartet werden, der aus einer Datenquelle einen einzigen Datensatz holt. Wenn der Befehl mehrere Datensätze zurückliefern würde, wird nur der erste in dem <code>Record</code> -Objekt gespeichert. Dies ist performanter als das Öffnen eines Recordsets.

Neu in ADO 2.7 ist lediglich die Unterstützung für 64-Bit-Prozessoren.

9.5 Tabellenzugriff mit Recordsets

Recordset	Die <code>Recordset</code>-Klasse ermöglicht den lesenden und schreibenden Zugriff auf Daten in Form einer relationalen Tabelle mit <i>n</i> Zeilen und <i>m</i> Spalten. Dabei spielt es keine Rolle, wie die Daten wirklich gespeichert sind. Mit einem <code>Recordset</code> kann beispielsweise auch auf hierarchisch angeordnete Informationen eines Verzeichnisdienstes zugegriffen werden. Es ist die Aufgabe des OLE DB-Providers, die Daten in Tabellenform zu bringen und – sofern der OLE DB-Provider Änderungen zulässt – die von OLE DB-Konsumenten vorgenommenen Änderungen an der Tabelle wieder in die Speicherform der Datenquelle zu überführen. Ein <code>Recordset</code>-Objekt bezieht sich zwar auf eine Tabelle, dennoch ist nach dem Cursor-Prinzip zu einem Zeitpunkt jeweils nur ein Datensatz zugreifbar. Die <code>Recordset</code>-Klasse stellt Methoden zur Bewegung des Cursors durch die Tabelle bereit.



Abbildung 9.6:
Ansicht der Klasse
»Recordset« im
VB6-Objektkatalog

9.5.1 Lesezugriff auf Daten

Bei der Arbeit mit einem Recordset-Objekt sind folgende Schritte üblich:

1. Instanziierung der Recordset-Klasse

```
Set rs = CreateObject("ADODB.Recordset")
```

Für Objektvariablen für die Klasse Recordset verwendet man nach der ungarischen Notation das Präfix rs. Wenn in einer Routine nur ein Recordset-Objekt verwendet wird, ist es durchaus üblich, die Objektvariable einfach rs zu nennen.

2. Aufruf der Methode Open() auf dem Recordset-Objekt. Dabei müssen ein Connection-String und ein SQL-Select-Befehl angegeben werden, die die gewünschten Daten liefern. Außerdem werden zwei Parameter benötigt, die die Art des Zugriffs spezifizieren. Statt eines kompletten Select-Befehls kann hier auch der Name einer Tabelle oder einer gespeicherten Abfrage angegeben werden, wenn eine Veränderung der Ergebnismenge nicht gewünscht ist.

**Arbeit mit einem
Recordset-Objekt**

```
rs.Open SQL, ConnectionString, CursorTyp, LockTyp, [Options]
```

3. Danach steht der erste Datensatz der Tabelle über das Objekt `rs` zur Verfügung. Der Zugriff auf die Spalten erfolgt über die `Fields`-Collection.
4. Nach Beendigung der Arbeit an der Tabelle muss das `Recordset`-Objekt durch Aufruf der Methode `Close()` aufgefordert werden, die geöffnete Verbindung zu schließen.

```
rs.Close
```

Parameter beim Öffnen von Tabellen

Die beim Öffnen des `Recordsets` angegebenen Parameter `CursorTyp` und `LockTyp` haben entscheidenden Einfluss auf die Arbeit mit den Daten.

Cursor-Typen

Der verwendete `Cursor`-Typ bestimmt die Möglichkeiten zur Navigation und zum Schreibzugriff auf eine Tabelle (vgl. Tabelle 9.5). Ein dynamischer `Cursor` ist der mächtigste und langsamste Typ. Sie sollten daher immer prüfen, ob einer der anderen Typen in Ihrem Anwendungsfall reicht. Wenn dieser Parameter nicht angegeben wird, ist der Standard `adOpenForwardOnly`.



Die zur Verfügung stehenden `Cursor`-Typen hängen vom OLE DB-Provider ab. Nicht alle OLE DB-Provider unterstützen alle `Cursor`-Typen!

Tabelle 9.5:
Cursor-Typen des
`Recordset`-Objekts

Cursor-Typ	Wert	Beschreibung
<code>adOpenStatic</code>	3	Ein statischer <code>Cursor</code> bedeutet, dass ADO eine statische Kopie aller Datensätze der Quelle anlegt. Es ist eine uneingeschränkte Bewegung durch die Datensätze möglich, jedoch können die Datensätze nicht geändert werden. Ein statischer <code>Recordset</code> gibt nur den Stand der Daten zum Zeitpunkt des Öffnens wieder.
<code>adOpenForwardOnly</code>	0	Ein <code>Forward-Only-Cursor</code> erzeugt ebenfalls einen statischen <code>Recordset</code> , aber mit einer weiteren Einschränkung gegenüber dem statischen <code>Cursor</code> : eine Bewegung durch die Datensätze ist nur in eine Richtung, vom Anfang zum Ende, möglich. Ein Rücksprung zum Anfang der Tabelle führt zu einer erneuten Ausführung der Abfrage. Dieser <code>Cursortyp</code> ist der effizienteste Zugriff, wenn Daten nur gelesen und nur sequenziell ausgegeben werden sollen.

Tabellenzugriff mit Recordsets

Cursor-Typ	Wert	Beschreibung
adOpenDynamic	2	Ein dynamischer Cursor ist der mächtigste, aber auch langsamste Cursortyp. Datensätze können gelesen und geändert werden. Das Anlegen neuer Datensätze und das Löschen von Datensätzen ist möglich. Auch Änderungen, die von anderen Benutzern der Datenquelle nach dem Öffnen vorgenommen wurden, werden berücksichtigt.
adOpenKeyset	1	Ein Keyset-Cursor ist fast so mächtig wie ein dynamischer Cursor. Er unterliegt jedoch einer Einschränkung: Datensätze, die von anderen Benutzern nach dem Öffnen angelegt wurden, sind nicht sichtbar. Änderungen an bestehenden Datensätzen werden jedoch weiterhin berücksichtigt.

Tabelle 9.5:
Cursor-Typen
des Recordset-
Objekts
(Forts.)

Der Fehler 3251 »Die von der Anwendung angeforderte Operation wird von dem Provider nicht unterstützt« bedeutet nicht zwangsläufig, dass die Operation von dem OLE DB-Provider grundsätzlich nicht unterstützt wird. Dieser Fehler tritt leider auch dann auf, wenn Sie einen Recordset eines Typs verwenden, der die gewünschte Operation nicht unterstützt.



Als zweiter Parameter kann die Art der Sperrung angegeben werden, die ausgeführt werden soll, um gleichzeitige Änderungen durch mehrere Clients zu verhindern. Wenn der Parameter fehlt, ist der Standard *adLockReadOnly*.

Sperrung

Konstante	Wert	Erläuterung
adLockReadOnly	1	Daten können nur gelesen werden.
adLockPessimistic	2	Schreib- und Lesezugriff. Ein Lock auf einen Datensatz wird gesetzt, sobald ein Feld beschrieben wird.
adLockOptimistic	3	Schreib- und Lesezugriff. Ein Lock auf einen Datensatz wird erst gesetzt, wenn der Abschluss der Änderungen mit der Methode <code>Update()</code> angezeigt wird.
adLockBatch Optimistic	4	Schreib- und Lesezugriff. Alle Änderungen werden gesammelt und zu einem definierten Zeitpunkt mit der Methode <code>UpdateBatch()</code> übergeben. Daher wird die Datenquelle erst dann gesperrt.

Tabelle 9.6:
Sperrungsarten

Spaltenzugriff

Der Zugriff auf die einzelnen Spalten (Felder) der Tabelle erfolgt über die `Fields-Collection`. Wie in Visual Basic üblich, also

Fields-Collection

ActiveX Data Objects (ADO)

```
rs.Fields.Item("feldname")
```

oder verkürzt in einer der folgenden synonymen Schreibweisen:

```
rs.Fields("feldname")  
rs("feldname")
```

Nur in VB6/A und VB.NET, nicht aber in VBS ist auch folgende Schreibweise möglich: `rs!feldname`.

Bewegung durch die Tabelle

Move-Methoden

Für die Bewegung zwischen den Zeilen (Datensätzen) der Tabelle stellt ADO eine Reihe von Methoden bereit, die jeweils mit dem Wort *Move* beginnen.

Tabelle 9.7:
Die Move-Methoden

Methode	Beschreibung
MoveFirst	Springt zum ersten Datensatz der Tabelle
MoveLast	Springt zum letzten Datensatz der Tabelle
MoveNext	Springt zum nächsten Datensatz. Falls der Cursor zum Zeitpunkt des Aufrufs auf dem letzten Datensatz der Tabelle stand, wird kein Fehler erzeugt, sondern das Attribut EOF gesetzt. Wenn bei gesetztem EOF die Methode MoveNext aufgerufen wird, kommt es zu einem Fehler.
MovePrevious	Springt einen Datensatz zurück. Falls der Cursor zum Zeitpunkt des Aufrufs auf dem ersten Datensatz der Tabelle stand, wird kein Fehler erzeugt, sondern das Attribut BOF gesetzt. Wenn bei gesetztem BOF MovePrevious aufgerufen wird, kommt es zu einem Fehler.
Move Num Records, [Start]	Führt einen Sprung zu einem bestimmten Datensatz aus. Dabei kann dies sowohl ein relativer Sprung vom aktuellen Datensatz aus als auch ein absoluter Sprung vom Anfang oder Ende der Tabelle aus gesehen sein. NumRecords legt die Anzahl der zu überspringenden Datensätze fest. Eine negative Zahl ist eine Rückwärtsbewegung. Die Angabe der Startposition ist optional. Wenn hier nichts angegeben wird, wird ein relativer Sprung ausgeführt. Die Optionen für Start sind: <i>adBookmarkCurrent</i> (0): beginnt beim aktuellen Datensatz <i>adBookmarkFirst</i> (1): beginnt beim ersten Datensatz <i>adBookmarkLast</i> (2): beginnt beim letzten Datensatz Hier kann auch ein Bookmark angegeben werden, das als Basis für den Sprung verwendet werden soll.

Feststellung der Recordset-Grenzen

BOF und EOF

Mit den Attributen BOF und EOF kann festgestellt werden, ob die Enden der Tabelle erreicht sind.

Tabellenzugriff mit Recordsets

- ▶ EOF ist wahr, wenn der Cursor den letzten Datensatz überschritten hat. Dies ist erst gegeben, nachdem ein `MoveNext()` auf dem letzten Datensatz ausgeführt wurde.
- ▶ BOF ist wahr, wenn der Cursor vor dem ersten Datensatz steht. Dies ist erst gegeben, nachdem ein `MovePrevious()` auf dem ersten Datensatz ausgeführt wurde.

Wenn ein `Recordset`-Objekt geöffnet wurde und Datensätze existieren, ist der erste Datensatz aktuell, und die Eigenschaften `BOF` und `EOF` sind *False*. Sie sind *True*, wenn kein Datensatz in dem `Recordset` enthalten ist. Mit Hilfe von `EOF` und `MoveNext()` ist ein sequenzieller Durchlauf durch alle Datensätze möglich.

```
Do While Not rs.EOF
    ... Aktionen ...
    rs.MoveNext
Loop
```

Machen Sie nicht den Fehler, den schon viele erfahrene Programmierer an dieser Stelle gemacht haben: Vergessen Sie nicht das `MoveNext()`! Ohne `MoveNext()` steckt das Skript in einer Endlosschleife!



Beispiel

Das folgende Beispiel zeigt den zuvor beschriebenen Ablauf. Dabei wird ein Standard-`Recordset`-Objekt (forward-only und read-only) erzeugt, das alle Datensätze aus der Tabelle `Produkte` aufnimmt. `Produkte` ist eine Tabelle aus der Access-Datenbank `KATALOG.MDB`, die Sie auf der CD finden. Um das Beispiel nutzen zu können, müssen Sie die ebenfalls enthaltene `KATALOG.udl` anpassen und in diesem Skript den Pfad zu dieser UDL-Datei. Die Definition von Konstanten zu Beginn dient nur der Übersichtlichkeit.

**Produkte aus einer
Produkttable**

```
Dim rs ' As ADODB.Recordset
' 0. --- Vorbereitungen
' - SQL-String: Welche Daten?
Const SQL = "SELECT * FROM Produkte where kategorie=23"
' - Connection-String bilden
Const CONNSTRING = "File Name=d:\buch\data\katalog.udl;"
'Const connString = "Provider=Microsoft.Jet.OLEDB.4.0;User
ID=hs;Password=egal;Persist Security Info=True;Data
Source=D:\buch\data\katalog.mdb;Jet OLEDB:System
database=d:\buch\data\katalog.mda"
' 1. --- Datenbankzugriffsobjekt (ADO) erzeugen
Set rs = CreateObject("ADODB.Recordset")
' 2. --- Verbindung herstellen, SQL ausführen
rs.Open SQL, CONNSTRING
' 3. --- Ausgabe aller Datensätze
```

ActiveX Data Objects (ADO)

```
Do While Not rs.EOF
    say rs.fields.Item("Name") & " kostet " & _
    FormatCurrency(rs("Preis"))
    rs.MoveNext
Loop
' 4.--- Schließen
rs.Close
```

Listing 9.2: Datenbank öffnen und alle Datensätze einer Tabelle ausgeben [ado_rs_lesen.wsf]



Vorsicht vor einem häufigen Fehler: Ohne den Befehl `rs.movenext()` kommt es zu einer Endlosschleife!

Ausgabe aller Felder

Feldzugriff

Im obigen Beispiel wurden zwei bestimmte Felder aus einer Tabelle ausgegeben. In vielen Fällen möchte man jedoch entweder alle Felder ausgeben (z.B. wenn man testen möchte, ob die SQL-Abfrage überhaupt korrekt arbeitet) oder aber man kennt die Feldnamen gar nicht. In beiden Fällen hilft eine universelle Unterroutine `list_table()`. `list_table()` erwartet als einzigen Parameter ein geöffnetes Recordset-Objekt. Innerhalb der üblichen Iteration über alle Datensätze erfolgt eine Iteration durch die `Fields`-Collection. Die Namen der einzelnen Felder werden dann verwendet, um die Werte aus dem Recordset-Objekt auszulesen.

```
Dim a ' Ausgabe
' --- Iteration über alle Datensätze
Do While Not rs.EOF
    a = ""
    ' --- Iteration über alle Felder
    For Each f In rs.Fields
        ' --- Zusammensetzen der Ausgabe
        If a <> "" Then a = a & ";"
        If Not IsNull(rs(f.Name)) Then
            a = a & rs(f.Name)
        End If
    Next
    ' --- Ausgabe
    say a
    ' --- nächster Datensatz
    rs.MoveNext
Loop
```

Listing 9.3: Eine hilfreiche Routine zur Ausgabe eines beliebigen Recordsets [list_table.wsf]

Nachträgliche Veränderungen der Ergebnismenge

Die Datensätze im Recordset können bei einigen Cursor-Typen auch nach Öffnen des Recordsets noch gefiltert oder neu sortiert werden. Die Filterung erfolgt über die Zuweisung einer Bedingung an das Attribut `Filter`. **Filter**

```
rs.Filter = "Menge > 5 and KundenID >7"
```

Erlaubt sind alle SQL-Bedingungen. Ebenso in SQL-Form ist die nachträgliche Sortierung möglich. **Sort**

```
rs.Sort = "Menge ASC, KundenID ASC"
```

Auf- und Wiederfinden von Datensätzen

Neben der sequenziellen Bewegung in einem Recordset sind folgende Bewegungen möglich: **Bookmark**

- Suche nach einem bestimmten Datensatz mit Hilfe von `Find()`

```
rs.Find "KundenID = 9"
```

- Die Verwendung von Lesezeichen. Damit kann auf einfache Weise zu einer bestimmten Position zurückgekehrt werden.

```
' Lesezeichen merken
lesezeichen = rs.Bookmark
....
' Rückkehr an diese Stelle
rs.Bookmark = lesezeichen
```

In typisierten Umgebungen muss `lesezeichen` den Typ *String* haben. Bookmarks können als Ausgangspunkte für `Find()` und *Move*-Methoden dienen.

Nicht alle Data Provider unterstützen Bookmarks.

9.5.2 Schreibzugriff auf Daten

Datensätze, die über ein `Recordset`-Objekt im Zugriff sind, können geändert und gelöscht werden. Außerdem können Datensätze angefügt werden. Voraussetzung dafür ist jedoch, dass

- der Provider die Operation grundsätzlich unterstützt, **Voraussetzungen**
- der `Recordset`-Typ (vgl. Ausführungen im vorherigen Kapitel), zu dem das `Recordset`-Objekt gehört, die Operation erlaubt.

Datensätze ändern

Die Änderung eines Datensatzes erfolgt auf Basis eines geöffneten `Recordsets` in drei Schritten: **Update()**

1. Navigation zum gewünschten Datensatz
2. Beschreiben der gewünschten Felder
3. Aufrufen der Methode Update()

```
rs("Name")= "Lienekogel"  
rs("Vorname")= "Rolf"  
rs.Update
```

Listing 9.4: Änderung eines Datensatzes

Datensätze anfügen

AddNew() und Update()

AddNew() erzeugt einen neuen Datensatz in einem Recordset-Objekt. Nach dem Aufruf der AddNew()-Methode wird der neu einzufügende Datensatz zum aktuellen Datensatz, solange bis die Update()-Methode aufgerufen wird.

```
rsDaten.AddNew  
rs("Name")= "Kleinschmidt"  
rs("Vorname")= "Carsten"  
rs.Update
```

Listing 9.5: Anfügen eines Datensatzes mit AddNew()

Um sicherzustellen, dass ein Recordset-Objekt die AddNew()-Methode erlaubt, können Sie die Supports()-Methode mit dem Parameter *adAddNew* benutzen.

Wenn eine der Move-Methoden benutzt wurde und das Recordset-Objekt keine Lesezeichen unterstützt, kann nicht mehr zu dem neu erzeugten Datensatz zurückgekehrt werden.

Es kann notwendig sein, die Requery()-Methode aufzurufen, um den neuen Datensatz im Recordset erscheinen zu lassen.

Datensätze löschen

Delete()

Zum Löschen von Datensätzen wird die Delete()-Methode benutzt.

```
Recordset.Delete [AffectRecords]
```

Der Parameter *AffectRecords* bestimmt, welche Datensätze vom Löschen betroffen sind.

- ▶ *adAffectCurrent* (1): Nur der aktuelle Datensatz wird gelöscht (Standardeinstellung).
- ▶ *adAffectGroup* (2): Alle Datensätze, die der aktuellen Filter-Einstellung entsprechen, werden gelöscht.
- ▶ *adAffectAll* (3): Alle Datensätze im Recordset werden gelöscht.

Tabellenzugriff mit Recordsets

```
rs.MoveLast  
rs.Delete
```

Listing 9.6: Löschen des letzten Datensatzes

Beispiel

In diesem Beispiel wird wieder die Produkttabelle aus dem vorherigen Unterkapitel verwendet. Dieses Mal werden alle Datensätze der Tabelle Produkte in den Recordset eingelesen. In einer Schleife werden alle Produkte der Produktkategorie 11 gelöscht und für alle anderen Produkte der Preis um 5% erhöht. Zum Schluss wird ein neues Produkt in die Tabelle eingefügt.

**Veränderungen
an der Produkt-
tabelle**

```
Dim rs ' As ADODB.Recordset  
Dim AlterPreis, NeuerPreis  
' -- ADO Konstanten  
Const adOpenDynamic = 2  
Const adLockOptimistic = 3  
' -- Eigene Konstanten  
Const SQL = "SELECT * FROM Produkte" ' Alle Datensätze!  
Const CONNSTRING = "File Name=d:\buch\data\katalog.udl;"  
' --- Datenbankzugriffsobjekt (ADO) erzeugen  
Set rs = CreateObject("ADODB.Recordset")  
' --- Verbindung herstellen, SQL ausführen  
rs.Open SQL, CONNSTRING, adOpenDynamic, adLockOptimistic  
' --- Ausführung von Änderungen  
Do While Not rs.EOF  
    If rs("Kategorie") = 11 Then ' Produkt löschen!  
        say "Produkt " & rs("Name") & " wird gelöscht!"  
        rs.Delete  
    Else ' Preis erhöhen  
        AlterPreis = rs("Preis")  
        NeuerPreis = AlterPreis * 1.05  
        rs("Preis") = NeuerPreis  
        say rs("Name") & ": Preiserhöhung von " & _  
        AlterPreis & " auf " & NeuerPreis  
        rs.Update  
    End If  
    rs.MoveNext  
Loop  
' --- Anfügen eines neuen Datensatzes  
rs.AddNew  
rs("Name") = "Amiga500"  
rs("Kategorie") = 23  
rs("Preis") = 1000  
rs.Update  
say "Neues Produkt wurde eingetragen!"  
' --- Schließen  
rs.Close
```

Listing 9.7: Löschen, Ändern und Anfügen mit einem Recordset [ado_rs_schreiben.wsf]

9.5.3 Dynamische Recordsets im Speicher

**Dynamische
Recordsets haben
keine Datenquelle**

Selbst vielen gestandenen ADO-Programmierern ist ein interessantes Feature von ADO nicht bekannt: ADO-Recordsets müssen keineswegs immer das Resultat einer Abfrage einer persistenten Datenquelle sein, sondern können auch eine beliebige dynamische Tabelle im Hauptspeicher sein. Sie können eine Instanz von `ADODB.Recordset` erzeugen und – ohne an eine Datenquelle zu binden – eine eigene Feldstruktur erzeugen, diese füllen und damit arbeiten wie mit einem »richtigen« Recordset. Die Konstanten für Datentypen, die beim Aufruf von `rs.fields.append` anzugeben sind, entsprechen im Wesentlichen den Datentypkonstanten von Visual Basic.

```
Dim rs ' As ADODB.Recordset
Dim felder() ' As Variant
' -- Neues Recordset-Objekt
Set rs = CreateObject("ADODB.Recordset")
' -- Aufbau des dynamischen Recordsets
felder = Array("ComputerID", "Computername", "Aktiv")
rs.fields.Append "ComputerID", 3 ' 3 = adInteger
rs.fields.Append "ComputerName", 8, 50 ' 8 = adBSTR
rs.fields.Append "Aktiv", 11 ' 11 = adBoolean

' -- Füllen des Recordsets
rs.Open
rs.AddNew felder, Array(1, "Sonne2000", True)
rs.Update
rs.AddNew felder, Array(2, "Mars", True)
rs.Update
rs.AddNew felder, Array(3, "Saturn", False)
rs.Update
' -- Ausgabe zur Kontrolle
rs.MoveFirst
rs.Sort = "ComputerID DESC"
rs.Filter = "aktiv = true"
list_table rs
```

Listing 9.8: In diesem Beispiel wird eine aus drei Feldern bestehende Tabelle im Hauptspeicher erzeugt und probeweise mit einigen Werten gefüllt, bevor die Routine `list_table()` zur Ausgabe aufgerufen wird. [ado_dynrs.wsf]



Dynamische Recordsets eröffnen im Rahmen von Automatisierungsprojekten die Möglichkeit, Daten auf einfache Weise im Hauptspeicher zu verwalten.

9.5.4 Persistenz von Recordsets

Save() Ein Recordset kann in zwei Formen persistent gespeichert werden:

Tabellenzugriff mit Recordsets

- in einer Datei im Dateisystem

```
rs.Save Datei, PersistentTyp
```

Die durch einen gültigen Pfad spezifizierte Datei darf noch nicht existieren, sonst kommt es zu einem Fehler.

- in einem Stream (verfügbar ab ADO 2.5), wobei `stm` ein Objekt vom Typ `ADODB.Stream` ist.

```
rs.Save stm, adPersistXML
```

Dabei verwendet OLE DB den Service-Provider »Microsoft OLE DB Persistence Provider«.

Formate

Bei der Abspeicherung in einer Datei stehen folgende Formate zur Verfügung:

ADTG und XML

- *Advanced Data Tablegram* (ADTG) (Const `adPersistADTG = 0`)
- *Extensible Markup Language* (XML) (Const `adPersistXML = 1`)

XML hat dabei den Vorteil, dass zahlreiche Werkzeuge zur Verfügung stehen, um XML darzustellen bzw. weiterzuverarbeiten. ADTG-Files sind jedoch kleiner. Natürlich können auch binäre Informationen (z.B. Bitmaps) gespeichert werden.

Eine persistente Recordset-Datei kann auf unterschiedliche Art und Weise verwendet werden:

Laden eines persistenten Recordsets

- Ein Recordset kann unter Angabe des Dateinamens geöffnet werden.

```
' --- Wiedereinlesen über rs.open
Set rs = CreateObject("ADODB.Recordset")
rs.Open "d:\buch\data\users.xml", "Provider=MSPersist"
```

- Der Dateiname kann auch der `Execute()`-Methode eines Connection-Objekts übergeben werden.

```
' --- Wiedereinlesen über conn.Execute
Set conn = CreateObject("ADODB.Connection")
conn.Open "Provider=MSPersist"
Set rs = conn.Execute("d:\buch\data\users.xml")
```

Bei der Speicherung als XML-Datei werden nicht nur die Daten selbst, sondern auch ein XML-Schema zur Beschreibung der Tabellenstruktur abgespeichert.



Streams

Neu seit ADO Version 2.5 ist die Klasse `ADODB.Stream`, die es ermöglicht, Recordsets im ADTG- oder XML-Format auch im Speicher zu halten. Ein Recordset kann mit `Save()` in ein Stream-Objekt gespeichert werden:

Streams

ActiveX Data Objects (ADO)

```
Set st = CreateObject("ADODB.Stream")
rs.Save st, adPersistXML
say "Der Stream enthält: " & st.Type
```

Listing 9.9: Speichern eines Recordsets in ein Stream-Objekt

Arbeit mit dem Stream-Objekt

Ein Stream kann zwei Typen von Daten enthalten: Binäre Daten (Type = 1) oder Textdaten (Type = 2). Ein binärer Stream entsteht durch die Speicherung eines Recordsets im ADTG-Form, ein Stream in Textform durch die Speicherung als XML. Ein Stream-Objekt bietet zahlreiche Möglichkeiten:

- ▶ Auslesen der Daten mit `Read()` bzw. `ReadText()`
 - ▶ Prüfen auf Erreichen des Stream-Endes mit dem Attribut `EOS` (`EOS` steht für End of Stream)
 - ▶ Beschreiben eines Streams mit `Write(array_of_bytes)` bzw. `WriteText(string)`
 - ▶ Speichern eines Streams in eine Datei
- ```
st.SaveToFile "d:\buch\data\users0.xml", 2
' 2 = adSaveCreateOverWrite
```
- ▶ Einlesen eines Streams aus einer Datei
- ```
st.LoadFromFile "d:\buch\data\users0.xml"
```

Aus einem Stream-Objekt kann sofort wieder ein Recordset erzeugt werden, indem beim Öffnen eines Recordsets statt eines SQL-Befehls das Stream-Objekt übergeben wird.

```
rs.Open st
```

Andere Objekte mit IStream-Schnittstelle

IStream-Schnittstelle

Die Stream-Klasse implementiert eine IStream-Schnittstelle. Es gibt aber auch andere Klassen, die diese Schnittstelle implementieren, z.B. das `Intrinsic Object Response` in ASP und die `MSXML.DOMDocument`-Klasse. Ein ADTG- oder XML-Stream kann direkt an Instanzen von Klassen, die IStream implementieren, übergeben werden. Der XML-Parser wird aber nicht glücklich sein, wenn Sie ihm einen ADTG-Stream übergeben.

```
Dim dom ' As MSXML.DOMDocument
Set dom = CreateObject("Microsoft.XMLDOM")
rs.Save dom, 1 'adPersistXML
```

Listing 9.10: Direktes Speichern eines Streams in den XML-Parser

9.5.5 Weitere Features der Recordset-Klasse

Die Recordset-Klasse stellt eine Reihe von weiteren Features zur Verfügung, die in diesem Buch jedoch nicht näher besprochen werden sollen, weil alle essenziellen Ausgaben auch ohne diese Funktionen erfüllt werden können:

Befehlsausführung mit der Command-Klasse

- ▶ Recordsets können mit der `Clone()`-Methode dupliziert werden, so dass mit zwei Zeigern parallel auf die gleiche Ergebnismenge zugegriffen werden kann. **Clone()**
- ▶ Große Datensatzmengen, die nicht in einem Rutsch, sondern in bestimmten, gleichgroßen Teilen bearbeitet werden sollen (z.B. weil der Benutzer sie seitenweise betrachtet), können mit Hilfe der Eigenschaft `PageSize` in Blöcke zerlegt werden. Mit Hilfe von `AbsolutePage` können einzelne Blöcke direkt angesprochen werden. **PageSize**
- ▶ Mit einer Ausführung der `Open()`-Methode können mehrere SQL-Anfragen übergeben werden (*Multiple Recordsets*). ADO erhält dann mehrere Ergebnismengen, die mit `NextRecordset()` durchblättert werden können. Diese Recordsets sind immer forward-only. **Multiple Recordsets**
- ▶ Die `Requery()`-Methode aktualisiert die Daten des Recordsets, indem die zu Grunde liegende Abfrage erneut ausgeführt wird. **Requery()**
- ▶ So genannte *Disconnected Recordsets* ermöglichen es, eine Datenmenge auf den Client zu laden, sie dort ohne weiteren Kontakt zum Server zu bearbeiten und zu einem gegebenen Zeitpunkt alle Änderungen zum Server zu übertragen. **Disconnected Recordsets**
- ▶ Der Cursor eines Recordsets wird normalerweise auf dem Server verwaltet. Für Provider, die dies (für manche Features) nicht unterstützen, sowie für Disconnected Recordsets ist es möglich, die Cursorverwaltung auf den Client zu verlagern. Dies sollte jedoch bei großen Recordsets nicht erfolgen, da die Performance erheblich sinken wird (`CursorLocation = 3 'adUseClient`). **Client-Side-Cursor**

9.6 Befehlsausführung mit der Command-Klasse

`Command` ist die flexibelste ADO-Klasse, denn sie ermöglicht die Ausführung beliebiger Befehle auf einer Datenquelle. Befehle können sein: **ADODB.Command**

- ▶ SQL-Befehle, die keine Ergebnismenge liefern (z.B. `DELETE`, `INSERT`, `UPDATE`)
- ▶ SQL-Abfragen mit `SELECT`, die eine Ergebnismenge liefern
- ▶ Stored Procedures, wobei Parameter übergeben und Rückgabewerte empfangen werden können
- ▶ der Name einer Tabelle, die komplett zurückgegeben werden soll
- ▶ der Name einer persistenten Recordset-Datei, die geöffnet werden soll
- ▶ HTTP-URLs, die eine Datenquelle spezifizieren (Dabei wird der Microsoft OLE DB Provider for Internet Publishing verwendet.)
- ▶ ADO-Streams (z.B. XML-Queries). Dieses Feature ist neu in ADO 2.6.

Der Ablauf bei Verwendung der Command-Klasse ist wie folgt:

1. Instanziierung der Command-Klasse

```
Set rs = CreateObject("ADODB.Command")
```

2. Setzen des Connection-Strings. Hier gibt es in Verbindung mit einem Connection-Objekt auch die Möglichkeit, eine bestehende Verbindung zu nutzen (siehe Kapitel 9.7).

```
objCmd.ActiveConnection = conString
```

3. Festlegen des Befehlstyps: Es gibt fünf Befehlstypen (vgl. Tabelle 9.8). Dabei ist adCmdText der gebräuchlichste Typ.

```
objCmd.CommandType = 1 'adCmdText
```

4. Festlegen des Befehls

```
objCmd.CommandText = sql
```

5. Befehl ausführen mit der Execute()-Methode

```
objCmd.Execute()
```

6. Objekt vernichten. Ein Command-Objekt muss nicht geschlossen werden. Die Verbindung wird von selbst sofort nach der Ausführung beendet.

```
Set objCmd = Nothing
```

Tabelle 9.8:
Befehlstypen für
Execute()

Konstante für Befehlstyp	Wert	Erläuterung
adCmdText	1	Befehl ist ein SQL-Befehl oder eine Stored Procedure
adCmdTable	2	Befehl ist ein Tabellenname (wird in einen SQL-Befehl umgesetzt)
adCmdStoredProc	4	Befehl ist eine Stored Procedure
adCmdFile	256	Befehl ist der Dateiname eines persistenten Recordsets
adCmdTableDirect	512	Befehl ist ein Tabellenname (wird direkt vom Provider geliefert)



In Zusammenarbeit mit einem Connection-Objekt kann ein Befehl mehrfach verwendet werden. Dabei erhält das Command-Objekt einen Namen, über den es im Folgenden von dem Connection-Objekt aus angesprochen werden kann.

Ausführung von SQL-Befehlen ohne Ergebnismenge

Beispiel: Delete

In diesem Beispiel wird der SQL-Befehl DELETE ausgeführt, der keine Ergebnistabelle liefert.

Befehlsausführung mit der Command-Klasse

```
' Deklarationen
Dim objCmd ' As ADODB.Command
' 0. --- Vorbereitungen
' - Befehl
Const sql = "Delete from Bestellungen"
' - Connection-String
Const conString = "File Name=d:\buch\data\katalog.udl;"
' 1. --- ADO-Command instanziiieren
Set objCmd = CreateObject("ADODB.Command")
' 2. --- Verbindungsinformationen setzen
objCmd.ActiveConnection = conString
' 3. --- Festlegen des Befehlstyps
objCmd.CommandType = 1 'adCmdText
' 4. --- Befehl festlegen
objCmd.CommandText = sql
' 5. --- Befehl ausführen
objCmd.Execute
' 6.--- Objekt vernichten
Set objCmd = Nothing
```

*Listing 9.11: Ausführung eines SQL-Befehls über ADODB.Command ohne Rückgabemenge
[ado_cmd_1.wsf]*

Ausführung von SQL-Befehlen mit Ergebnismenge

Auch SELECT-Befehle und SQL-Stored-Procedures, die eine Tabelle als Ergebnis liefern, können mit der Command-Klasse ausgeführt werden. In diesem Fall liefert Execute() einen Zeiger auf ein Recordset-Objekt zurück.

Beispiel: Select

```
' Deklarationen
Dim objCmd ' As ADODB.Command
' 0. --- Vorbereitungen
' - Befehl
Const sql = "Select * from Bestellungen"
' - Connection-String
Const conString = "File Name=d:\buch\data\katalog.udl;"
' 1. --- ADO-Command instanziiieren
Set objCmd = CreateObject("ADODB.Command")
' 2. --- Verbindungsinformationen setzen
objCmd.ActiveConnection = conString
' 3. --- Festlegen des Befehlstyps
objCmd.CommandType = 1 'adCmdText
' 4. --- Befehl festlegen
objCmd.CommandText = sql
' 5. --- Befehl ausführen
Set rs = objCmd.Execute
list_table rs
```

ActiveX Data Objects (ADO)

```
' 6.--- Objekt vernichten  
Set objCmd = Nothing
```

Listing 9.12: Ausführung eines SQL-Befehls über ADODB.Command mit Rückgabemenge [ado_cmd_2.wsf]



Ein Recordset, das von `Execute()` zurückgeliefert wird, hat stets den Cursor-Typ `forward-only` und den Sperrmodus `read-only`. Falls ein anderer Cursor-Typ gewünscht wird, muss ein Recordset-Objekt mit `CreateObject()` und den entsprechenden Eigenschaften erzeugt werden.

Ausführen von gespeicherten Prozeduren

adCmdStoredProc

Gespeicherte Prozeduren, die keine Parameter benötigen, werden ausgeführt wie SQL-Befehle. Sofern Parameter benötigt werden, muss der Befehlstyp `adCmdStoredProc` (=4) ausgeführt werden. Dabei werden die Parameter in Form von Parameter-Objekten übergeben. Der Erzeugung von Parametern dient die Methode `CreateParameter()`:

```
Set objPara = objCMD.CreateParameter (Name, Type, Direction,  
[Size], [Value])
```

Das Parameter-Objekt muss anschließend noch mit der Methode `Append()` der `Parameters`-Collection des jeweiligen `Command`-Objekts hinzugefügt werden.

```
Const CONSTRING = "Provider=SQLOLEDB.1;Data Source=sonne2000;Initial  
Catalog=Northwind;User ID=sa;Password=egal;"  
Const STOREDPROC = "custOrderHist"  
Const CUSTOMERID = "ALFKI"  
Const adBSTR = 8  
Const adParamInput = 1  
Dim objcmd ' As adodb.command  
' --- ADO-Command instanziiieren  
Set objcmd = CreateObject("ADODB.Command")  
' --- Verbindungsinformationen setzen  
objcmd.ActiveConnection = CONSTRING  
' --- Kommandotyp und Kommandoname setzen  
objcmd.CommandType = 4 ' = adCmdStoredProc  
objcmd.CommandText = STOREDPROC  
' -- Parameter anlegen  
Set objPara = objcmd.CreateParameter("CustomerID", adBSTR, _  
adParamInput, 5, CUSTOMERID)  
objcmd.Parameters.Append objPara  
' --- Befehl synchron ausführen
```


Dauerhafte Verbindungen mit der Connection-Klasse

```
Set rs = objcmd.Execute  
list_table rs
```

Listing 9.13: Ausführung einer Stored Procedure auf der Northwind-Datenbank eines SQL Servers 7.0 [ado_cmdstoredproc.wsf]

Achten Sie genau auf den in der Stored Procedure verwendeten Datentyp bzw. dessen Länge. Das obige Beispiel funktioniert nicht, wenn bei CreateParameter() die Länge 5 nicht genannt wird.



Asynchrone Ausführung

Befehle können auch asynchron ausgeführt werden. Dabei wird bei der Ausführung ein spezieller Parameter angegeben. Mit Hilfe des Attributs state kann der aktuelle Zustand überprüft werden. Sofern eine Ergebnismenge erwartet wird, kann diese anschließend mit einem weiteren Execute(, , adAsyncFetch) abgeholt werden.

Execute()

```
...  
' --- Befehl asynchron ausführen  
Const adAsyncExecute = 16  
Const adStateExecuting = 4  
Const adAsyncFetch = 32  
objcmd.Execute , , adAsyncExecute  
' --- Warten auf Ende...  
Do While (objcmd.state = adStateExecuting)  
    say "Aufruf läuft..."  
Loop  
' --- Daten abholen und ausgeben  
Set rs = objcmd.Execute(, , adAsyncFetch)  
list_table rs
```

Listing 9.14: Asynchrone Ausführung einer Stored Procedure [ado_cmdasync.wsf]

9.7 Dauerhafte Verbindungen mit der Connection-Klasse

Mit einem Connection-Objekt wird eine dauerhafte Verbindung (Sitzung) zu einer Datenquelle hergestellt. Im Rahmen dieser Sitzung können Recordsets geöffnet und Befehle ausgeführt werden. Dies hat den Vorteil, dass mehrere Anweisungen eine bestehende Verbindung nutzen können und nicht auf kostspielige Weise jeweils eine eigene Verbindung auf- und wieder abgebaut werden muss. Die Connection-Klasse wird wie folgt benutzt:

ADODB.
Connection

1. Instanziierung der Connection-Klasse

Vorgehensweise

```
Set con = CreateObject("ADODB.Connection")
```

ActiveX Data Objects (ADO)

2. Aufruf der Methode `Open()` aus dem `Connection`-Objekt. Dabei muss ein `Connection-String` angegeben werden. Die Angabe von Benutzername und Passwort ist nur notwendig, wenn diese Angaben nicht im `Connection-String` spezifiziert sind.

```
objCon.Open conString [,benutzer, kennwort]
```

3. Danach können mit der Methode `Execute()` beliebig viele SQL-Befehle auf der stehenden Verbindung ausgeführt werden. Sofern der mit `Execute()` übermittelte SQL-Befehl eine Ergebnismenge liefert, kann diese als `Recordset`-Objekt empfangen und verarbeitet werden.

```
objCon.Execute(sql)
```

oder

```
Set rs = objCon.Execute(sql)
```

4. Nach Beendigung der Arbeit an der Tabelle muss das `Connection`-Objekt durch Aufruf der Methode `Close()` aufgefordert werden, die geöffnete Verbindung zu schließen. Sofern `Recordsets` verwendet wurden, sollten diese vorher geschlossen werden.

```
rs.Close
```

```
objConn.Close
```



Es ist empfehlenswert, ein `Connection`-Objekt zu erzeugen, wenn mehr als ein `Recordset` von einer Datenquelle geöffnet bzw. mehr als ein Befehl auf der Datenquelle ausgeführt werden soll. Dieses Vorgehen spart Ressourcen gegenüber einer impliziten Öffnung einer Verbindung bei jedem `Recordset` bzw. `Command`.

Beispiel

Beispiel Das Beispiel zeigt, wie nacheinander vier Aktionen auf einer Verbindung ausgeführt werden:

- Zuerst werden alle Bestellungen des Kunden mit der KundenID 100 gelöscht.
- Danach werden alle enthaltenen Datensätze angezeigt. Dazu wird ein `Recordset`-Objekt verwendet.
- Es wird für den Kunden 100 eine neue Bestellung für zehn Einheiten des Produkts 1234 eingefügt.
- Alle Datensätze werden erneut ausgegeben. Dabei muss die Methode `Requery()` auf dem noch geöffneten `Recordset`-Objekt angewendet werden, um die Inhalte zu aktualisieren. Alternativ wäre möglich, das `Recordset` neu zu öffnen. Diese Aktion wäre aber zeitaufwendiger.

```
Dim objCmd, objrs, objcon ' Objekte
```

```
Dim i ' Zählvariable
```

```
' ----- Connection-String definieren: Wo sind die Daten
```

Dauerhafte Verbindungen mit der Connection-Klasse

```

gespeichert?
conString = "File Name=d:\buch\data\katalog.udl;"
' --- Connection instanzieren und öffnen
Set objcon = CreateObject("ADODB.Connection")
objcon.Open conString, "hs", "egal"
' --- Command instanzieren und verbinden
Set objCmd = CreateObject("ADODB.Command")
objCmd.ActiveConnection = objcon
' --- Recordset instanzieren, verbinden und öffnen
Set objrs = CreateObject("ADODB.Recordset")
objrs.ActiveConnection = objcon
objrs.Open "Select * from Bestellungen", , adOpenDynamic, _
adLockReadOnly
' -- 1. Aktion
objCmd.CommandText = "Delete from Bestellungen where KundenID = 100"
objCmd.Execute
' -- 2. Aktion
list_table objrs
' -- 3. Aktion
sql1 = "Insert into Bestellungen(KundenID,PID,Menge)
values(100,1234,10)"
objCmd.CommandText = sql1
objCmd.Execute
' --- 4. Aktion
objrs.Requery
list_table objrs
' --- Aufräumen
objrs.Close
objcon.Close

```

Listing 9.15: Mehrfachverwendung einer Connection [ads_con_2.wsf]

Auch Connections können asynchron aufgebaut werden.



Mehrere Wege führen zum Ziel

Die folgende Tabelle zeigt die verschiedenen Möglichkeiten, SQL-Befehle auszuführen, noch einmal im Überblick.

Recordset versus Command/Connection

ADO-Klasse	Befehle mit Ergebnistabelle	Befehle ohne Ergebnistabelle
Recordset-Klasse	lesend und schreibend	nein
Command-Klasse und Connection-Klasse	Execute() kann read-only-Recordset zurückliefern	mit Execute()

Tabelle 9.9: Unterschiede zwischen den ADO-Klassen

9.8 Zugriff auf einzelne Elemente mit der Record-Klasse

Record Die Klasse `ADODB.Record` dient bei ADO dem Zugriff auf nicht-relationale Quellen, in denen einzelne Elemente direkt ansprechbar sind. Dabei wird eine Instanz dieser Klasse über einen Moniker an einen Eintrag in der Datenquelle gebunden. Es ist aber nicht so, dass ein `Recordset`-Objekt eine Menge von `Record`-Objekten darstellt. Die Klasse `Record` wurde mit ADO Version 2.5 neu eingeführt.

Die Klasse `Record` wird eingesetzt:

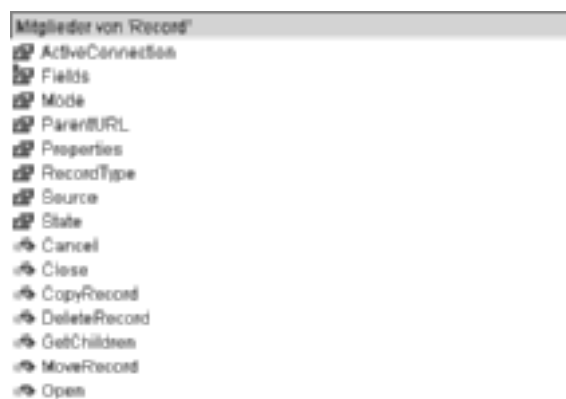
- ▶ in Verbindung mit dem *OLE DB Provider for Internet Publishing (MDAIPP)* für den Zugriff auf Webfolder und den Exchange Server 2000-Webstore
- ▶ in Verbindung mit dem *EXOLEDB-Provider* für den Zugriff auf den Exchange Server 2000-Webstore.

Mitglieder

Felder Die `Record`-Klasse besitzt wie die `Recordset`-Klasse eine `Fields`-Collection, die die Eigenschaften des angesprochenen Elements enthält. Je nach den Fähigkeiten der Datenquelle können Eigenschaften gelesen, geändert, angefügt und gelöscht werden. Die Klasse `Record` stellt im Gegensatz zu `Recordset` keine `Update()`-Methode bereit. Dazu muss die `Update()`-Methode der `Fields`-Collection aufgerufen werden.

Bewegungen Da die Klasse keine Objektmenge darstellt, gibt es keine »Move«-Methoden zwischen verschiedenen Elementen, sondern nur eine `MoveRecord()`-Methode, um das ganze Element zu bewegen. Ebenso gibt es Methoden zum Löschen (`DeleteRecord()`) und Kopieren des Elements (`CopyRecord()`). `GetChildren()` liefert einen Zeiger auf ein `Recordset`-Objekt, wenn `Record` auf ein Element zeigt, das Unterelemente hat (z.B. ein Ordner im Dateisystem oder ein Ordner im Exchange 2000-Webstore).

Abbildung 9.7:
Liste der Mitglieder
der Klasse
`ADODB.Record` im
VB6-Objektkatalog



Zugriff auf einzelne Elemente mit der Record-Klasse

Open()

Zentral in dieser Klasse ist die Methode `Open()`, die über zahlreiche Attribute verfügt.

Open Quelle, Verbindung, Verbindungsmodus, Erstellungsoptionen, Öffnungsoptionen

Quelle ist der URL eines Webstore-Elements. *Verbindung* ist optional ein `Connection-String` oder `ADODB.Connection`-Objekt. *Verbindungsmodus* wählt den Schreib- und Lesemodus. *Erstellungsoptionen* legt fest, ob das Element neu erzeugt und ob ggf. ein unter diesem Namen vorhandenes Element überschrieben werden soll. Die *Öffnungsoptionen* ermöglichen das asynchrone Öffnen eines Elements.

Parameter

Symbolische Konstante	Wert	Erläuterung
<code>adModeRead</code>	1	Nur lesend
<code>adModeWrite</code>	2	Nur schreibend
<code>adModeReadWrite</code>	3	Lesen und schreibend
<code>adModeShareDenyNone</code>	16	Keine Beschränkung für parallele Benutzer
<code>adModeShareDenyRead</code>	4	Verhindert, dass andere Benutzer parallel lesend zugreifen können
<code>adModeShareDenyWrite</code>	8	Verhindert, dass andere Benutzer parallel schreibend zugreifen können
<code>adModeShareExclusive</code>	12	Verhindert, dass andere Benutzer parallel überhaupt zugreifen können
<code>adModeRecursive</code>	0x400000	Legt fest, ob die vorgenannten Sperroptionen auch für eventuelle Unterelemente gelten. Möglich in Verbindung (OR-Verknüpfung) mit <code>adModeShareDenyNone</code> , <code>adModeShareDenyWrite</code> und <code>adModeShareDenyRead</code> .

Tabelle 9.10:
Verbindungsmodus
(besser Plural –
modi) für »Open()«

Tabelle 9.11:
Erstellungsoptionen für »Open()«

Symbolische Konstante	Wert	Erläuterung
adCreateCollection	8192	Erzeugt ein neues Element mit Unter-elementen (z.B. einen Ordner im Exchange-Webstore oder ein Verzeichnis in einem Dateisystem)
adCreateNonCollection	0	Erzeugt ein neues Element ohne Unter-elemente (z.B. eine Nachricht im Exchange-Webstore oder eine Datei in einem Dateisystem)
adCreateStructDoc	-2147483648	Erzeugt ein neues COM-Structured Storage-Element
adCreateOverwrite	67108864	Überschreiben, wenn Element schon vorhanden. Möglich in Kombination (OR-Verknüpfung) mit adCreateCollection, adCreateNonCollection und adCreateStructDoc.
adFailIfNotExists	-1	Fehlermeldung, wenn Element schon vorhanden (Standardoption)
adOpenIfExists	33554432	Element nicht anlegen, sondern öffnen, wenn Element schon vorhanden. Möglich in Kombination (OR-Verknüpfung) mit adCreateCollection, adCreateNonCollection und adCreateStructDoc.

Beispiel

Das Beispiel zeigt den lesenden und schreibenden Zugriff auf eine Eigenschaft eines Ordners im Exchange 2000-Webstore. Weitere Anwendungsbeispiele der Record-Klasse finden Sie im Kapitel zur Exchange 2000-Programmierung (Kapitel 19).

```
' ### Ordnerattribute ändern (Ändern des Kommentars zu einem Ordner im
Exchange 2000-Webstore)
Sub ADOEX2K_Ordner_AttributAenderungen()
Dim rec ' As ADODB.Record
Dim url ' As String

' --- Verbindungsaufbau
' Angaben zum Ordner
Const DOMAIN = "sonne2000.IT-Visions.de"
Const wurzel = "public"
Const ORDNER = "Lehre/Wintersemester2001_2002/VTBDV1"
' URL zusammenbauen
url = "http://" & DOMAIN & "/" & wurzel & "/" & ORDNER
```

Transaktionssteuerung

```
say "Zugriff auf Ordner: " & url
' Instanziierung der Klasse
Set rec = CreateObject("ADODB.Record")
' Öffnen des Records zum Lesen und Schreiben
Const adModeReadWrite = 3
rec.Open url, , adModeReadWrite
say "Ordner geöffnet: " & rec.ActiveConnection

' --- Attribut ändern
' Zugriff auf Attribut
Dim f ' As ADODB.Field
Set f = rec.fields("DAV:comment")
' Wert vorher
say "Alter Kommentar: " & f.Value
' Wert ändern
f.Value = "TEST " & Now
rec.fields.Update
' Wert nachher
say "Neuer Kommentar: " & f.Value
End Sub
```

*Listing 9.16: Ändern des Kommentars zu einem Ordner im Exchange 2000-Webstore
[CD:/code/komponenten/ADOEX2K/ADOEX2K_AttributAenderungen.wsf]*

9.9 Transaktionssteuerung

ADO unterstützt auch die Definition von Transaktionen. Transaktionen sind atomare Einheiten. Zu einer Transaktion gehören mehrere Befehle, die entweder alle komplett und korrekt ausgeführt werden oder aber von denen keiner ausgeführt wird. Es soll nicht zu einem inkonsistenten Zustand kommen, in dem ein Teil der Befehle ausgeführt wurde, ein anderer aber nicht. In diesem Fall wird ein so genannter Rollback ausgeführt.

Nicht alle Provider unterstützen Transaktionen. Falls der Provider Transaktionen nicht unterstützt, erzeugt die Verwendung von Transaktions-Methoden einen Fehler.

Die **Connection-Klasse** besitzt drei Methoden für die Steuerung von Transaktionen:

- ▶ **BeginTrans()** markiert den Anfang einer Transaktion.
- ▶ **CommitTrans()** markiert das erfolgreiche Ende einer Transaktion.
- ▶ **RollbackTrans()** markiert den Fehlerfall, in dem alle seit Beginn der Transaktion vorgenommenen Änderungen rückgängig gemacht werden sollen.

**Transaktions-
Methoden**



Wenn eine ADO-Anwendung plan- und außerplanmäßig (durch Absturz) beendet wird, so führt ADO automatisch ein Rollback aus.

Beispiel für den Einsatz von Transaktionen

Beispiel Ein Beispiel zur Transaktionssteuerung ist naturgemäß etwas länger, weil eine Transaktion dann besonders eindrucksvoll ist, wenn mehrere Änderungsbefehle innerhalb einer Verbindung ausgeführt werden. Eine Tabelle wird zunächst gelöscht, dann werden zehn zufällige Datensätze eingefügt. Der Benutzer hat am Ende die Wahl: Wählt er die Rücknahme aller Änderungen, so kann diesem Wunsch durch einen einzigen Befehl entsprochen werden.

```
Dim objCmd, objrs, objcon ' Objekte
Dim i ' Zählvariable
' ----- Connection-String definieren: Wo sind die Daten
gespeichert?
conString = "File Name=d:\buch\data\katalog.udl;"
' --- Connection instanziiieren und öffnen
Set objcon = CreateObject("ADODB.Connection")
objcon.Open conString, "hs", "egal"
' --- Command instanziiieren und verbinden
Set objCmd = CreateObject("ADODB.Command")
objCmd.ActiveConnection = objcon
' --- Recordset instanziiieren, verbinden und öffnen
Set objrs = CreateObject("ADODB.Recordset")
objrs.ActiveConnection = objcon
objrs.Open "Select * from Bestellungen", , adOpenDynamic, _
adLockReadOnly
' --- Ausgabe des Anfangszustands
say "Inhalt vorher:"
list_table objrs
' --- Transaktion beginnen
objcon.BeginTrans
' --- Änderung 1
objCmd.CommandText = "Delete from Bestellungen"
objCmd.Execute
say "Inhalt nach Löschen aller Datensätze:"
list_table objrs
' -- Änderung 2
For i = 1 To 10
    sql1 = "Insert into Bestellungen(KundenID,PID,Menge) values(" & i
    -
    & "," & i & "," & Int(Rnd(Second(Now)) * 10) & ")"
    objCmd.CommandText = sql1
    objCmd.Execute
Next
```


ADO-Fehlerbehandlung

```
objrs.Requery
say "Inhalt nach Einfügen neuer Datensätze:"
list_table objrs
' --- Ende der Transaktion
If MsgBox("Änderungen übernehmen?", vbYesNo) = vbYes Then
    objcon.CommitTrans
Else
    objcon.RollbackTrans
End If
' --- Ausgabe des Endzustands
objrs.Requery
say "Endzustand:"
list_table objrs
' --- Aufräumen
objrs.Close
objcon.Close
```

Listing 9.17: Transaktionssteuerung mit ADO [ado_trans.wsf]

9.10 ADO-Fehlerbehandlung

Ein `Error`-Objekt enthält nähere Informationen zu den Fehlern, die beim Datenzugriff aufgetreten sind. Da bei einem Datenzugriff auch mehrere Fehler auftreten können, existiert die `Errors`-Auflistung, in der alle Fehler aufgelistet werden. Die aufrufende Anwendung zeigt dagegen jeweils nur den letzten aufgetretenen Fehler an. Außerdem erzeugt OLE DB auch Warnungen in dieser Collection, die nicht zu einem kritischen Fehler in der Anwendung führen.

Fehler anzeigen

Die `Errors`-Auflistung existiert nur in einem `Connection`-Objekt. Für alle Aktionen, für die keine explizite Verbindung erzeugt wurde, gibt es keinen Zugriff auf die Fehlerliste.

Die Errors-Auflistung

Durch Zugriff auf die `Count`-Eigenschaft der `Errors`-Auflistung kann schnell festgestellt werden, ob ein OLE DB-Fehler aufgetreten ist. Mit der `Clear()`-Methode werden die `Error`-Objekte aus der Auflistung gelöscht, nachdem sie ausgewertet wurden.

Es bietet sich an, eine allgemeine Fehlerbehandlungsroutine für VB- und OLE DB-Fehler zu schreiben. Dies leistet `eHandler()`. Die Routine erwartet einen Zeiger auf das Visual Basic `Err`-Objekt und einen Zeiger auf eine geöffnete ADO-Connection. Ist der zweite Parameter `Null`, `Nothing` oder `Empty`, wird die Überprüfung der ADO-Connection nicht ausgeführt.

```
Sub ehandler(objErr, objCon)
' --- VB-Fehler?
If objErr.Number <> 0 Then
    say "### Es ist ein VB-Fehler aufgetreten:"
    say "Fehlernummer:      " & objErr.Number
```

ActiveX Data Objects (ADO)

```
        say "Beschreibung:      " & objErr.Description
        say "Source:           " & objErr.Source
        say Chr(13)
        objErr.Clear
    End If
    ' --- OLE DB-Fehler?
    If Not (objCon Is Nothing Or _
        IsNull(objCon) Or IsEmpty(objCon)) _
        And objCon.Errors.Count > 0 Then
        say "### Es gibt " & objCon.Errors.Count & " OLE DB-Meldung(en)"
        ' --- Iteration über alle Fehler
        For Each Fehler In objCon.Errors
            say "OLE DB-Fehler      #" & i & " von " & _
                objCon.Errors.Count
            say "Fehlernummer (Hex): " & Hex(Fehler.Number)
            say "Beschreibung:      " & Fehler.Description
            say "Quelle:           " & Fehler.Source
            say "NativeError:       " & Fehler.NativeError
            say "SQLState:         " & Fehler.SQLState
            say Chr(13)
        Next
    End If
End Sub
```

Listing 9.18: Allgemeine Fehlerbehandlungsroutine für VB- und ADO-Fehler

Verwendung von eHandler()

Das folgende Beispiel zeigt den Einsatz von `eHandler()`. Nachdem der Anfang der Fehlerbehandlung mit `On Error Resume Next` aktiviert wurde, erfolgt an allen wichtigen Stellen der Aufruf von `eHandler()`. Testen Sie `eHandler()`, indem Sie in das SQL-Statement oder den Connection-String einen Fehler einbauen!

```
' Deklarationen
Dim objCmd, objCon
' Fehlerbehandlung ein
On Error Resume Next
' 0. --- Vorbereitungen
' - Befehl
Const sql = "Select * from Bestellungen"
' - Connection-String
Const conString = "File Name=d:\buch\data\katalog.udl;"
' 1. --- Instanziierungen
Set objCon = CreateObject("ADODB.Connection")
ehandler Err, objCon
Set objCmd = CreateObject("ADODB.Command")
ehandler Err, objCon
' 2. --- Verbindung öffnen
objCon.Open conString, "hs", "egal"
```

ADO-Fehlerbehandlung

```
ehandler Err, objCon  
' 3. --- Befehl ausführen  
objCmd.ActiveConnection = objCon  
objCmd.CommandType = 1 'adCmdText  
objCmd.CommandText = sql  
objCmd.Execute  
ehandler Err, objCon  
' 4.--- Verbindung schließen  
objCon.Close  
' 5. --- Objekte vernichten
```

Listing 9.19: Verwendung der Routine ehandler() [ado_error.wsf]

10 Taskscheduler-Komponente

Der Taskscheduler (MSTASK.EXE) ist ein neueres Windows-Werkzeug zur zeitgesteuerten Ausführung von Aufgaben. Er ist eine verbesserte Variante des NT-Schedulers (Dienstname *Schedule*, ATSVC.EXE), der über das Kommandozeilentool AT.EXE oder das GUI WINAT.EXE (Bestandteil des NT4-Resource Kit) verwaltet werden kann.

Geplante
Vorgänge
automatisieren

Der Taskscheduler wird auch *Scheduling Agent* genannt. Intern heißt auch der neue Dienst nur *Schedule*.

10.1 Grundlagen des Taskschedulers

Der Taskscheduler verfügt gegenüber dem alten NT-Scheduler über folgende Verbesserungen:

Features

- ▶ Er ist nicht nur für NT4 und Windows2000/XP/.NET, sondern auch für Windows 95/98/ME verfügbar (auf letztgenannten Systemen mit Einschränkungen).
- ▶ verfeinerte Zeitplanoptionen
- ▶ zusätzliche Startoptionen auf Basis von Systemereignissen
- ▶ Optionen zum Unterbrechen bzw. Löschen eines Tasks auf Grund bestimmter Systemereignisse
- ▶ bessere Integration in die Windows-Benutzeroberfläche
- ▶ Festlegen eines Sicherheitskontextes für jeden einzelnen Task. Bisher konnte nur ein Sicherheitskontext für alle Tasks definiert werden (nur unter NT4/Windows2000).
- ▶ Vergabe der administrativen Rechte für jeden einzelnen Task (nur unter NT4/Windows2000)
- ▶ Wizard zur Erstellung von Tasks

Es gibt leider zahlreiche Probleme mit dem Taskscheduler. Sehen Sie dazu in der Microsoft Knowledge Base unter dem Stichwort Taskscheduler nach. Sie werden dort zahlreiche Problemberichte finden. Betrachten Sie insbesondere [Q235536] und [Q236773]. In diesen Beiträgen werden die Unterschiede zwischen den beiden Zeitplandiensten genau erklärt.



Installation

Bezugsquellen Der Taskscheduler ist Bestandteil von Windows 98/ME sowie von Windows2000/XP/.NET. Für Windows 95 und NT 4.0 ist er Bestandteil der Desktop-Aktualisierung im Internet Explorer ab Version 4.0. Leider wird der Anwender beim Installieren des Internet Explorers nicht auf den Umstand hingewiesen, dass der Zeitplandienst ausgetauscht werden soll.

Konzepte

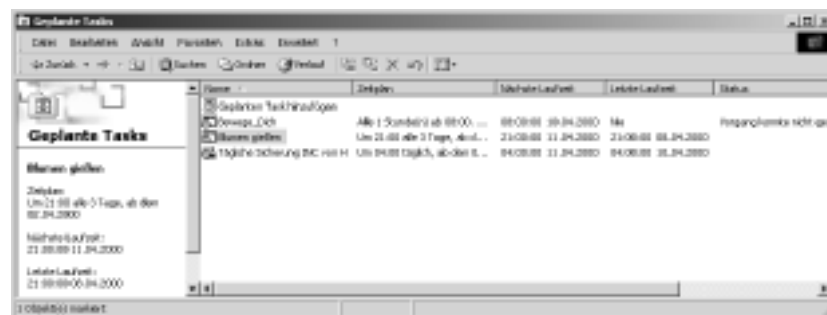
Tasks und Triggers Die grundlegenden Konzepte und Begrifflichkeiten des Taskschedulers werden nun erläutert:

- ▶ Ein **Task** ist die Bindung der Ausführung eines Programms, einer Batch-Datei oder eines Skripts an bestimmte Ereignisse.
- ▶ Ein **Trigger** ist ein Satz von Regeln, bei deren Eintreffen ein Task zur Ausführung kommt. Es gibt zwei Typen von Triggern:
 - ▶ zeitgesteuerte Trigger und
 - ▶ ereignisgesteuerte Trigger (beim Start des Computers, bei der Anmeldung oder im Leerlauf)

Zeitgesteuerte Trigger haben einen bestimmten Startrhythmus und können als erweiterte Zeitplanoption innerhalb dieses Rhythmus noch in einem bestimmten Minuten- oder Stundentakt wiederholt werden.

Job-Dateien Tasks und die dazugehörigen Trigger werden in Form von binären Dateien mit der Dateiextension .JOB im Dateisystem – üblicherweise in %WINDIR%/TASKS – gespeichert. Je nach Windows-Version finden Sie an unterschiedlichen Stellen im System Verknüpfungen zu diesem Ordner (z.B. unter ARBEITSPLATZ/SYSTEMSTEUERUNG/GEPLANTE VORGÄNGE).

Abbildung 10.1:
Ansicht der geplanten
Vorgänge unter
Windows2000



Die bisherigen Werkzeuge AT.EXE und WINAT.EXE sind nur in geringem Umfang kompatibel mit dem neuen Taskscheduler. Mit at/WinAT angelegte Tasks werden im Taskscheduler GUI angezeigt. Mit dem Taskscheduler angelegte Tasks können jedoch nicht von den beiden alten Werkzeugen dargestellt werden. Tasks, die mit at oder WinAT angelegt wurden, verschwinden aus der Anzeige, sobald sie mit dem Taskscheduler GUI modifiziert wurden (Microsoft definiert dies in [Q220149] als korrektes Verhalten.) Sie erhalten also mit at und WinAT auf einem System, auf dem der Taskscheduler installiert ist, keine korrekte Sichtweise mehr auf die konfigurierten Tasks.

at.exe und WinAT.exe

Log und Konfiguration

Der Taskscheduler führt ein Log seiner Aktivitäten üblicherweise in %WINDIR%/SCHEDLOG.TXT (unter NT: %SYSTEMROOT%). Die Registry-Einträge, die diese Verzeichnisse und andere globale Eigenschaften des Taskschedulers festlegen, befinden sich unter HKEY_LOCAL_MACHINE\SOFTWARE\MICROSOFT\SCHEDULINGAGENT.

Log und Registry

10.2 Programmierung

Mit dem Taskscheduler wird eine COM-Komponente zur Programmierung in Form der MSTASK.DLL bereitgestellt, die jedoch keine COM-Automation unterstützt (vgl. Scheduling Agent Service Class im COM-Viewer). Im Rahmen des Site Servers 3.0 liefert Microsoft jedoch eine verbesserte, automationsfähige Komponente (TASKSCHEDULER.DLL) aus, die hier beschrieben werden soll.

mstask.dll versus Taskscheduler.dll

Name und Abkürzung	Taskscheduler Komponente
Name der Komponentendatei	TASKSCHEDULER.DLL
Interner Name der Typbibliothek	TASKSCHEDULERLib
Helpstring der Typbibliothek	Taskscheduler 1.0 Type Library
Abweichende ProgID	Scheduler
Hersteller	Microsoft
Lizenzierung	Microsoft Site Server 3.0. Aus lizenzrechtlichen Gründen ist die DLL nicht einzeln auf der CD zu diesem Buch enthalten.
Besprochene Version	7.0.795.0
NT4 und Windows 95/98/ME	1. Internet Explorer 5.0 2. Regsvr32 TASKSCHEDULER.DLL

*Tabelle 10.1:
Schnellinfo
Taskscheduler*

Tabelle 10.1:
Schnellinfo
Taskscheduler
(Forts.)

Name und Abkürzung Windows2000	Taskscheduler Komponente Einzelregistrierung der DLL: regsvr32 TASKSCHEDULER.DLL
Dokumentation	Nicht vorhanden; die Dokumentation der nicht automationsfähigen Komponente finden Sie in der Microsoft MSDN Library [MSL00] unter "Setup and System Administration/Task Scheduler".



Die *Taskscheduler-Komponente* ist ein typisches Beispiel für eine Inkonsistenz zwischen Registry-Information und Typbibliothek. Der korrekte Name der Komponente ist *TASKSCHEDULERLib* mit der Einstiegsklasse *Scheduler*. In der Registry wird diese Klasse jedoch unter *Scheduler.SchedulingAgent.1* registriert. Dies führt dazu, dass unter VBS eine Instanz mit

```
Set objSch = CreateObject("Scheduler.SchedulingAgent.1")
```

erzeugt werden muss; bei Verwendung der Typbibliothek und des New-Operators unter VBA/VBS muss die Instanziierung so aussehen:

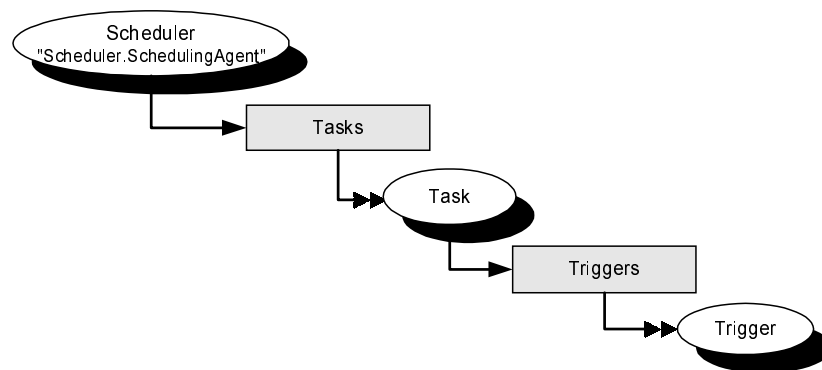
```
Dim objSch As New TASKSCHEDULERLib.Scheduler ' nur VB6/VBA/VB.NET!
```

10.3 Objektmodell

Taskscheduler-Objektmodell

Abbildung 10.2 zeigt das einfach und geradlinig aufgebaute Objektmodell der Komponente. Einstiegspunkt und einzige extern instanziierte Klasse ist *Scheduler*. Die Klasse enthält eine *Tasks-Collection* mit einzelnen *Task*-Objekten. Jedes *Task*-Objekt enthält eine *Triggers-Collection*, in der die zu dem *Task* definierten Trigger enthalten sind.

Abbildung 10.2:
Objektmodell der
Taskscheduler-
Komponente



Die folgende Tabelle zeigt die unterschiedlichen Trigger-Typen und die korrekten Werte für das Attribut *TriggerType* eines *Trigger*-Objekts.

Objektmodell

Typbibliothek

Die Typbibliothek in der TASKSCHEDULER.DLL enthält leider keine einzige Konstantendefinition.



Abbildung 10.3:
Einblick in die
Typbibliothek der
Taskscheduler-
Komponente



Die in den nachfolgenden Tabellen genannten Konstanten stammen aus der nicht COM-automationsfähigen Taskscheduler-Komponente, die mit dem Betriebssystem mitgeliefert wird. Damit hören die Gemeinsamkeiten aber auch schon auf. Die nicht automationsfähige Komponente hat ein anderes Objektmodell, beispielsweise sind die Trigger-Informationen nicht in einem Objekt, sondern in einer Struktur abgelegt. Diese Dokumentation finden Sie in der MSDN Library unter dem Punkt "Setup and System Administration/ Task Scheduler".

Konstante	Wert	Bedeutung
TASK_TIME_TRIGGER_ONCE	0	Task läuft nur einmal
TASK_TIME_TRIGGER_DAILY	1	Task läuft täglich in einem bestimmten Tagesintervall
TASK_TIME_TRIGGER_WEEKLY	2	Task läuft wöchentlich an bestimmten Wochentagen und in einem bestimmten Wochenintervall

Tabelle 10.2:
Trigger-Typen

Tabelle 10.2:
Trigger-Typen
(Forts.)

Konstante	Wert	Bedeutung
TASK_TIME_TRIGGER_MONTHLYDATE	3	Task läuft monatlich an einem bestimmten Tag (1 bis 31) des Monats. Begrenzung auf bestimmte Monate möglich.
TASK_TIME_TRIGGER_MONTHLYDOW	4	Task läuft monatlich immer an einem bestimmten Wochentag in der bestimmten Woche des Monats. Begrenzung auf bestimmte Monate möglich (Bsp.: Montags in der 2. und 4. Woche des Monats).
TASK_EVENT_TRIGGER_ON_IDLE	5	Task startet, wenn das System eine bestimmte Anzahl von Minuten im Leerlauf war
TASK_EVENT_TRIGGER_AT_SYSTEMSTART	6	Task startet beim Systemstart
TASK_EVENT_TRIGGER_AT_LOGON	7	Task startet bei der Anmeldung eines Benutzers

Trigger-Klasse

Die Trigger-Klasse besitzt zahlreiche Attribute (siehe Tabelle 10.3), wobei einige Attribute von allen Trigger-Typen benutzt werden, während andere spezifische Eigenschaften eines bestimmten Typs enthalten. Beim Triggertyp `ON_IDLE` kann die typspezifische Angabe für die Leerlaufdauer, nach der der Task gestartet werden soll, nicht für jeden Trigger individuell eingestellt werden. Die Angabe wird aus dem Attribut `IdleWaitMinutes` aus dem übergeordneten Task-Objekt übernommen.

Tabelle 10.3:
Attribute des
Trigger-Objekts

Attribut	Erläuterung
BeginTime	Datum und Uhrzeit der erstmaligen Ausführung des Tasks
Days	Erlaubt die Spezifikation von Monatstagen, an denen ein Task laufen soll. Dabei steht jedes der Bits 0 bis 30 für einen Monatstag. Dieses Attribut wird derzeit nur für den Tasktyp <code>MONTHLYDATE</code> verwendet.
DaysInterval	Tagesrhythmus zwischen zwei Läufen des Tasks beim Tasktyp <code>DAILY</code> . Beispiel: Der Wert 5 bedeutet, der Task läuft alle fünf Tage.
DaysOfTheWeek	Für den Tasktyp <code>WEEKLY</code> und <code>MONTHLYDOW</code> können hier die Wochentage definiert werden, an denen das Skript laufen soll. Bit 0 bis 6 stehen für jeweils einen Wochentag, wobei Bit 0 Sonntag ist und Bit 6 Samstag (amerikanische Wochenzählung).

Objektmodell

Attribut	Erläuterung
Duration	Zeitraum in Minuten, innerhalb dessen der Task zu dem in <code>Interval</code> definierten Intervall wiederholt werden soll. Die Dauer kann auch mehrere Stunden und Tage betragen. Während über das administrative GUI maximal 9999 Stunden und 59 Minuten eingestellt werden können, lässt das Objektmodell alle Werte des Datentyps <i>Long</i> zu. Negative Werte sind jedoch nicht sinnvoll.
EndTime	Datum der letztmaligen Ausführung; ist nur aktiv, wenn Bit 0 in <code>Flags</code> gesetzt ist
Flags	Bit 0: 0 = Trigger gilt endlos; 1 = Trigger hat definiertes Ende; Bit 1: 0 = Task nicht beenden, falls er noch ausgeführt wird; 1 = Task beenden, falls er noch ausgeführt wird. Diese beiden letzten Optionen sind nur in Verbindung mit einer eingestellten Wiederholung möglich und geben an, ob bei Eintreten des Wiederholungs-Zeitereignisses eine möglicherweise noch laufende Instanz des Tasks beendet werden soll. Es kann immer nur eine Instanz des Tasks gestartet sein!
Interval	Anzahl der Minuten, nach der die Aufgabe wiederholt werden soll. Wert muss zwischen 0 und 59 liegen. 0 bedeutet keine Wiederholung. Die Wiederholung endet nach der in <code>Duration</code> eingestellten Minutenanzahl.
Months	Für die Tasktypen <i>MONTHLYDATE</i> und <i>MONTHLYDOW</i> können hier die Monate festgelegt werden, in denen der Task starten soll. Jeweils ein Bit steht für jeden Monat, Bit 0 für Januar, Bit 11 für Dezember. Der Wert $2 + 8 + 32 + 128 + 512 + 2048 = 2730$ startet den Task also im Februar, April, Juni, August, Oktober und Dezember.
TriggerString	Enthält die Trigger-Information als String, so wie er in der Spalte <code>TASK PLANEN</code> im GUI angezeigt wird. Sofern ein Task mehrere Trigger besitzt, wird im GUI nur »Mehrfache Planungszeiten« angezeigt. Dieses Attribut ist read-only!
TriggerType	Enthält den Triggertyp, siehe Tabelle »Attribute des Trigger-Objekts«
WeeksInterval	Wochenrhythmus zwischen zwei Läufen des Tasks beim Tasktyp <i>WEEKLY</i> . Beispiel: Der Wert 5 bedeutet, der Task läuft alle fünf Wochen.
WhichWeek	Einstellung für den Tasktyp <i>MONTHLYDOW</i> . Task läuft 1 = am ersten, 2 = am zweiten, 3 = am dritten, 4 = am vierten, 5 = am letzten der bezeichneten <code>DaysOfTheWeek</code> .

Tabelle 10.3:
Attribute des
Trigger-Objekts
(Forts.)

Das Objektmodell geht in seinen Möglichkeiten etwas über das grafische Benutzerinterface hinaus: So ist es bei den Tasktypen *MONTHLYDATE* und *MONTHLYDOW* im GUI nur möglich, jeweils einen Kalender- bzw.



Wochentag für die Ausführung zu spezifizieren. Eine Mehrfachauswahl ist nicht gestattet. Das Objektmodell ermöglicht es aber, über die Attribute `Days` und `DaysOfTheWeek` mehrere Kalender- bzw. Wochentage anzuwählen. Die Spalte `TASK PLANEN` im GUI zeigt jedoch die über das Objektmodell eingestellten Mehrfachwerte korrekt an.

10.4 Liste der Tasks und ihrer Trigger

Das erste Taskscheduler-Skript zeigt die Auflistung aller definierten Tasks zusammen mit ihren Triggern.

```
Dim objSch, objTrig, objTask
Dim i ' Zähler für Trigger
' --- Instanz des Schedulers erzeugen
Set objSch = CreateObject("Scheduler.SchedulingAgent.1")
' --- Iteration über alle Tasks
For Each objTask In objSch.Tasks
    say objTask.JobName & ":" & objTask.ApplicationName
    say "...mit " & objTask.Triggers.Count & " Triggern."
    ' Iteration über alle Trigger
    i = 1
    For Each objTrig In objTask.Triggers
        say "Trigger #" & i _
            & " (Typ " & objTrig.TriggerType _
            & ") " & objTrig.TriggerString
        i = i + 1
    Next
    say "-----"
Next
```

Listing 10.1: Ausgabe aller eingerichteten Tasks mit ihren Triggern [task_show.wsf]

10.5 Erzeugung neuer Tasks

Taskerzeugung

Zur Erzeugung eines neuen Tasks sind folgende Schritte notwendig:

1. Erzeugung eines neuen Task-Objekts mit der Methode `Add()` innerhalb der Tasks-Auflistung des Scheduler-Objekts
2. Setzen der Attribute für das Task-Objekt. Der Task erscheint danach zwar schon in der GUI, ist aber als »TASK OHNE LAUFZEIT« markiert und wird nie ausgeführt.
3. Erzeugung mindestens eines Trigger-Objekts mit der Methode `Add()` innerhalb der Triggers-Auflistung des Task-Objekts
4. Festlegen des Trigger-Typs und der dazu passenden Attribute

Beispiel 1: Zeitgesteuerte Trigger

Das folgende Skript legt einen Task für ein einfaches VBS-Skript an, das Sie durch eine einfache Dialogbox alle 60 Minuten daran erinnert, sich kurz von Ihrem Bildschirm zu lösen und Ihrer Gesundheit zuliebe einige gymnastische Übungen durchzuführen.

```
' --- Instanz des Schedulers erzeugen
Set objSch = CreateObject("Scheduler.SchedulingAgent.1")
' 1. --- Task anlegen
Set objTask = objSch.Tasks.Add("Bewege_Dich.job")
' 2. --- Attribute setzen
objTask.Flags = 1 ' TASK_FLAG_INTERACTIVE
objTask.ApplicationName = "D:\Buch\Scheduler\bewege_Dich.wfs"
objTask.WorkingDirectory = "D:\Buch\"
objTask.Comment = "per Skript angelegter Task!"
' 3. --- Trigger anlegen
Set objTrig = objTask.Triggers.Add
' 4.a) --- Triggertyp setzen
objTrig.TriggerType = 1 ' TASK_TIME_TRIGGER_DAILY
objTrig.DaysInterval = 1 ' läuft jeden! Tag
objTrig.Flags = 0 ' Task läuft endlos
objTrig.BeginTime = Date & " 08:00:00" ' Heute beginnen
' 4.b) --- Wiederholung setzen
objTrig.Duration = 60 * 9 ' 9 Stunden Laufzeit
objTrig.Interval = 60 ' Alle 60 Minuten
```

Listing 10.2: Anlegen eines zeitgesteuerten Tasks [task_create.wsf]

```
MsgBox "Es ist Zeit, Dich mal wieder zu bewegen!", , "Auf, auf!"
```

Listing 10.3: bewege_Dich.wsf

Der obige Task läuft unbegrenzt. Im Austausch mit der Zeile `objTrig.Flags = 0` kann auch ein Enddatum definiert werden:

```
objTrig.EndTime = #1/30/2000#
objTrig.Flags = 1 'Task hat Ende
```

Abhängigkeiten

Leider gibt es einige tückische Abhängigkeiten bei den Attributwerten in der Taskscheduler-Komponente. So muss im obigen Beispiel

- ▶ Flags mit 0 initialisiert werden, sonst kann der Trigger nicht angelegt werden
- ▶ Duration vor Interval gesetzt werden
- ▶ die EndTime gesetzt sein, bevor mit `objTrig.Flags = 1` angezeigt wird, dass es ein Ende gibt

Die Missachtung dieser Abhängigkeiten kann zwei unterschiedliche Konsequenzen haben:



- Das Skript bricht mit einem Fehler ab.
- Das Skript läuft erfolgreich durch; der Task ist jetzt nicht funktionsfähig, was man daran erkennt, dass im GUI in der Spalte NÄCHSTE LAUFZEIT keine Zeitangabe, sondern »NIE« erscheint.

Beispiel 2: Ereignisgesteuerte Trigger

TRIGGER_ON_IDLE Im zweiten Beispiel wird ein ereignisgesteuerter Trigger (TRIGGER_ON_IDLE) verwendet, um nach jeweils zehnminütigem Leerlauf das Defragmentierungswerkzeug unter Windows 98 aufzurufen. Durch die Task-Flags wird auch festgelegt, dass der Task sich beenden soll, sobald der Leerlaufzustand endet.

```
' --- Instanz des Schedulers erzeugen
Set objSch = CreateObject("Scheduler.SchedulingAgent.1")
' 1. --- Task anlegen
Set objTask = objSch.Tasks.Add("defrag.job")
' 2. --- Task-Attribute setzen
objTask.Flags = TASK_FLAG_START_ONLY_IF_IDLE +
TASK_FLAG_KILL_ON_IDLE_END
objTask.ApplicationName = "C:\WINDOWS\DEFRAG.EXE"
objTask.Comment = "per Skript angelegter Task!"
objTask.WorkingDirectory = ""
objTask.IdleWaitMinutes = 10
objTask.IdleWaitDeadLineMinutes = 10
' 3. --- Trigger anlegen
Set objTrig = objTask.Triggers.Add
' 4. --- Triggerattribute setzen
objTrig.TriggerType = 5 'TASK_EVENT_TRIGGER_ON_IDLE
objTrig.BeginTime = "1.1.2000 00:00:00"
objTrig.Flags = 0 ' Task läuft endlos
MsgBox "Task mit TRIGGER_ON_IDLE wurde angelegt!"
```

Listing 10.4: Anlegen eines ereignisgesteuerten Tasks [task_defrag.wsf]

10.6 Manueller Start und Stopp

Start und Stopp Tasks können auch skriptgesteuert gestartet und beendet werden. Das Objektmodell enthält bei der Methode `Run()` einen Fehler: Diese kann erst benutzt werden, nachdem auf ein beliebiges Attribut des Jobs zugegriffen wurde.

```
Set objSch = CreateObject("Scheduler.SchedulingAgent.1")
Set objTask = objSch.Tasks("backup.job")
x = objTask.Status ' Run erst nach Attributzugriff verfügbar
objTask.Run
```

Listing 10.5: Manueller Start eines Tasks [task.starten.wsf]

10.7 Löschen eines Tasks

Das Löschen eines Tasks erfolgt wie üblich über ein `Delete()` auf der `Tasks-Collection`. Es können auch einzelne Trigger aus der `Triggers-Collection` gelöscht werden.

Task löschen

```
Set objSch = CreateObject("Scheduler.SchedulingAgent.1")
' --- Löschen
objSch.Tasks.Delete "Bewege_Dich.job"
```

Listing 10.6: Löschen eines Tasks[task.löschen.wsf]

10.8 Benutzerkontext setzen

Gerade für die unbeaufsichtigte Ausführung von Tasks ist es wichtig, unter welchem Benutzerkontext der Task läuft. Daraus ergeben sich die Möglichkeiten für das auszuführende Skript oder Programm, auf Ressourcen des eigenen Systems und anderer Systeme zuzugreifen.

Für den alten NT-Scheduling-Dienst besteht wie für alle NT-Dienste die Wahlmöglichkeit, den Dienst unter dem Systemkonto oder unter einem bestimmten Benutzerkonto laufen zu lassen, das über das Recht »Anmelden als Dienst« verfügt. Diese Einstellung galt dann aber für alle Tasks auf dem System. Der neue Taskscheduler erlaubt die Einstellung des Benutzerkontextes pro Task. Die globale Einstellung gilt nur noch für die mit alten Werkzeugen eingerichteten Tasks. Sie können den Benutzerkontext über die Methode `SetAccountInformation()` setzen.

**SetAccount-
Information()**

```
objTask.SetAccountInformation "hs", "egal" ' Benutzer, Kennwort
```

Sie müssen die Anweisung jedoch vor dem Anlegen des ersten Triggers aufrufen, da `SetAccountInformation()` die zuvor angelegten Trigger wieder löscht.

Die Festlegung des Benutzerkontextes für einen geplanten Vorgang ist eine Möglichkeit, eine Impersonifizierung auch für WSH-Skripte durchzusetzen, die keine eigene Identitätseinstellung besitzen. Ein WSH-Skript läuft normalerweise unter dem Benutzerkontext des Benutzers, der das Skript gestartet hat.



11 Registry-Zugriff mit RegCol

Die Bearbeitung der Registry ist eine wichtige Aufgabe im Rahmen der Automatisierung von Windows, da dort fast alle Konfigurationsinformationen gespeichert sind. Umso bedauerlicher ist es, dass Microsoft bisher kein attraktives Objektmodell für die Registry liefert: Weder die Registry-Methoden der Klasse `WSHShell` aus der Komponente *WSH Runtime Objects* noch die bei Visual Studio mitgelieferte Komponente *Registry Access Functions* (REGTOOL5.DLL) wird den Ansprüchen der Automatisierung gerecht. An dieser Stelle soll daher eine Komponente eines anderen Anbieters vorgestellt werden: Die Komponente *RegCol* der Firma Riposte bildet die Registry in genau dem Objektmodell ab, das intuitiv nahe liegt, wenn man den Baum der Registry öffnet.

RegCol

Name und Abkürzung	RegCol
Name der Komponentendatei	RegCol.dll
Interner Name der Typbibliothek	RegCol
Helpstring der Typbibliothek	M&W RegCol Object Library
Hersteller	Mey & Westphal RIPOSTE, Hamburg www.riposte.com
Lizenzierung	Kommerzielles Produkt
Besprochene Version	1.5
alle Window-Versionen	Einzelregistrierung der DLL: regsvr32 regcol.dll [CD: /install/komponenten/regcol]
Dokumentation	HTML-Help-Datei [CD: /install/komponenten/regcol]

Tabelle 11.1:
Schnellinfo RegCol

11.1 RegCol-Objektmodell

Die Registry besteht aus einer Liste von Registrierungsschlüsseln, die wiederum andere Schlüssel enthalten können. In RegCol besitzt daher die Klasse `Registry` eine Collection `RegistryKeys` aus `RegistryKey`-Objekten. Jedes `RegistryKey`-Objekt besitzt wiederum eine `RegistryKeys`-Collection.

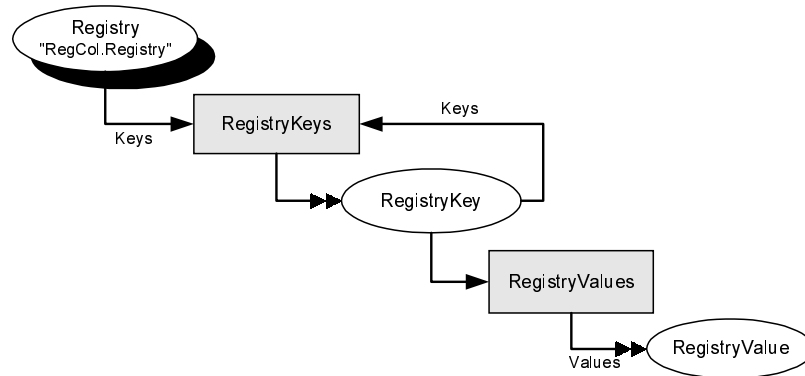
Schlüssel

Ein Schlüssel kann neben Unterschlüsseln auch Werte enthalten. Die Registry arbeitet mit so genannten *benannten Werten*, d.h., jeder einzelne Wert hat einen Namen, um ihn von anderen Werten innerhalb desselben Schlüssels unterscheiden zu können. RegCol definiert dazu innerhalb der Klasse `RegistryKey` eine Collection `RegistryValues` mit `RegistryValue`-Objekten. Da die benannten Werte in der Registry nicht mehr hierarchisch untergliedert werden können, endet das Objektmodell hier auch schon.

Werte

Registry-Zugriff mit RegCol

Abbildung 11.1:
Objektmodell von
RegCol



Rechte

Was RegCol leider nicht kann, ist das Lesen und Verändern der Rechte auf Registrierungsschlüssel. Dazu können Sie jedoch die Komponente *ADsSecurity* von Microsoft verwenden (siehe Kapitel 13.11). Für Umgebungen, in denen ein Zugriff auf das Win32-API besteht, ermöglicht das hier nicht weiter dokumentierte Attribut *hKey* in den Klassen *RegistryKey* und *RegistryValue* einen Zugriff auf das interne Handle des Keys. Damit ist ein Zugriff auf ergänzende Win32-API-Funktionen möglich. Ein Skriptentwickler hat diese Möglichkeit aber nicht.

Abbildung 11.2:
RegCol-
Typbibliothek im
VB6-Objektkatalog



Klasse Registry

Die Klasse `Registry` bietet neben dem Zugriff auf die `RegistryKeys`-Collection auch noch zahlreiche Methoden für zentrale Funktionen in Zusammenhang mit der Registry an.

Registry

Methoden	Erläuterung
<code>Add(Path, [Data], [Type As DataTypeConstants = dtREG_NONE], [Options As AddOptionConstants = aoREG_OPTION_NON_VOLATILE]) As RegistryValue</code>	Fügt den durch den Pfad (<code>Path</code>) bestimmten Wert (<code>Data</code>) in der Registry an und liefert das zugehörige <code>RegistryValue</code> -Objekt zurück. Falls der gewünschte Wert schon besteht, wird er mit den neuen Werten überschrieben. Fehlende, »vor« dem Wert liegende Schlüssel werden automatisch mit erzeugt.
<code>Connect([ComputerName As String])</code>	Verbindet das <code>Registry</code> -Objekt mit der Registry eines anderen Computers in einem Netzwerk. Anschließend greifen alle Methoden des Objekts und die aller davor erzeugten auf die entfernte Registry zu. Für den Programmierer ist völlig transparent, auf welchem PC sich die Registry befindet.
<code>Copy(SourcePath As String, TargetPath As String) As RegistryValue</code>	Kopiert den durch den Quellpfad (<code>SourcePath</code>) bestimmten Wert in den Zielpfad (<code>TargetPath</code>) und liefert das zugehörige <code>RegistryValue</code> -Objekt zurück. Falls der Wert im Zielpfad schon existiert, wird er mit den neuen Werten überschrieben. Fehlende, vor dem durch den Zielpfad bestimmten Wert liegende Schlüssel werden automatisch mit erzeugt.
<code>Export(FileName As String, [Key As String], [Subkeys As Boolean = True])</code>	Exportiert Teile der Registry in eine REG-Datei
<code>Find(What, [Key As String], [Subkeys As Boolean = True], [Options As FindOptionConstants = foAll])</code>	Sucht nach jedem Vorkommen des angegebenen Werts (<code>What</code>) in der Registry. Bei einem Fund wird der Event <code>Found()</code> ausgelöst.
<code>Import(FileName As String)</code>	Importiert Teile der Registry aus einer REG-Datei
<code>Move(SourcePath As String, TargetPath As String) As RegistryValue</code>	Verschiebt den im Quellpfad (<code>SourcePath</code>) angegebenen Wert in den Zielpfad (<code>TargetPath</code>) und liefert das zugehörige <code>RegistryValue</code> -Objekt zurück. Falls der Wert im Zielpfad schon existiert, wird er mit den neuen Werten überschrieben. Fehlende, vor dem durch den Zielpfad bestimmten Wert liegende Schlüssel werden automatisch mit erzeugt.
<code>Remove(Path As String)</code>	Entfernt den durch den Pfad (<code>Path</code>) bestimmten Wert unwiderruflich aus der Registry

Tabelle 11.2:
Methoden der Klasse
`Registry`

Collection-Klassen

In dem Objektmodell gibt es zwei Collection-Klassen, einmal für die Schlüssel und einmal für die Werte. Die Collection-Klasse `RegistryValues` unterstützt die gleichen Methoden wie die `RegistryKeys`-Collection. Daher ist nur letztgenannte Klasse hier ausführlich dokumentiert.

Tabelle 11.3:
Methoden der
Collection-Klasse
`RegistryKeys`

Methodenname	Erläuterung
<code>Add(Key As String)</code>	Fügt den durch den <code>Key</code> bestimmten Schlüssel in der Registry an und liefert das zugehörige <code>RegistryKey</code> -Objekt zurück. Falls der gewünschte Schlüssel schon besteht, wird er überschrieben. Fehlende, vor in der Hierarchie vorher liegende Schlüssel werden automatisch mit erzeugt.
<code>Copy(SourceIndex, TargetKey As String, [SubKeys As Boolean = True]) As RegistryKey</code>	Kopiert den durch den Quellindex (<code>SourceIndex</code>) bestimmten Schlüssel in den Zielschlüssel (<code>TargetKey</code>) und liefert das zugehörige <code>RegistryKey</code> -Objekt zurück. Falls der Schlüssel im Zielschlüssel schon existiert, wird er überschrieben. Mit <code>SubKeys=True</code> werden alle Unterschlüssel eingeschlossen.
<code>Count() As Long</code>	Retourniert die Anzahl der <code>RegistryKey</code> -Objekte in der aktuellen <code>RegistryKeys</code> -Collection
<code>Item(Index) As RegistryKey</code>	Retourniert den durch den <code>Index</code> bestimmten Schlüssel. Hier kann sowohl ein absoluter als auch ein relativer Pfad angegeben werden.
<code>Move(SourceIndex, TargetKey As String) As RegistryKey</code>	Verschiebt den durch den Quellindex (<code>SourceIndex</code>) bestimmten Schlüssel in den Zielschlüssel (<code>TargetKey</code>) und liefert das zugehörige <code>RegistryKey</code> -Objekt zurück. Dies schließt eventuell darunter liegende Schlüssel und Werte mit ein. Falls der Schlüssel im Zielschlüssel schon existiert, wird er überschrieben.
<code>Remove(Index)</code>	Entfernt den durch den <code>Index</code> bestimmten Schlüssel unwiderruflich aus der Registry. Dies schließt eventuell darunter liegende Schlüssel und Werte mit ein.

RegistryKey und RegistryValue

Die beiden Einzelklassen `RegistryKey` und `RegistryValue` besitzen nur Attribute und keine Methoden.

Lesen der Registry

Attributname	Erläuterung
Keys As RegistryKeys	Die Keys-Collection ermöglicht den Zugriff auf die untergeordneten Schlüssel.
Name As String	Name des Schlüssels
Values As RegistryValues	RegistryValues-Collection der Werte des Schlüssels

Tabelle 11.4:
Attribute der Klasse
RegistryKey

Attributname	Erläuterung
Data As Variant	Retourniert oder setzt die Daten des Werts
Length As Long	Retourniert die Länge der Daten des Werts
Name As String	Retourniert den Namen des Werts
Type As DataTypeConstants	Retourniert den Typ der Daten des Werts

Tabelle 11.5:
Attribute der Klasse
RegistryValue

11.2 Lesen der Registry

RegCol unterstützt drei alternative Formen des Zugriffs auf einen Schlüssel bzw. einen Wert:

- Adressierung über absolute Pfade
- Adressierung über relative Pfade von einem Schlüssel ausgehend
- rekursiver Abstieg über den Registry-Baum

Zugriff über absolute Pfade

Ein absoluter Pfad muss immer mit einem Backslash (»\«) beginnen. Der direkte Zugriff auf einen Schlüssel erfolgt über ein anderes Objekt als der Zugriff auf einen Wert:

Absolute Pfade

- absoluter Zugriff auf einen Schlüssel über die Keys-Collection des Registry-Objekts

```
Const testkey = "\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows
Script Host\Settings"
Const testval = "ActiveDebugging"
Set k = REG.keys.Item(testkey)
say k.Name
```

- absoluter Zugriff auf einen Wert über das Item-Attribut der Registry-Klasse

```
Set v = REG.Item(testkey & "\" & testval)
say v.Name & "=" & v.Data
```

Auf den Standardwert eines Schlüssels wird zugegriffen, wenn die Pfad-angabe mit einem Backslash »\« endet.

Zugriff über einen relativen Pfad

Relative Pfade

Wenn Sie ein RegistryKey-Objekt im Zugriff haben, können Sie von dort aus über einen relativen Pfad in den darunter liegenden Baum hinabreichen. Ein relativer Pfad darf nicht mit einem Backslash beginnen.

```
Set k = REG.keys.Item("\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\")
Set k = k.keys("WBEM\CIMOM")
Set v = k.values("Build")
say "Build=" & v.Data
```



Auf einen Wert kann nicht direkt über einen relativen Pfad zugegriffen werden. Es wäre also nicht erlaubt, `k.values("WBEM\CIMOM\Build")` zu verwenden. Sie müssen erst immer an den Schlüssel binden und danach über die Values-Collection den Wert auslesen.

Rekursiver Abstieg

Rekursiver Abstieg

Ein RegistryKey-Objekt besitzt sowohl eine Keys- als auch eine Values-Collection, mit der alle enthaltenen Unterschlüssel und Werte ausgegeben werden können.

```
Dim REG ' As regcol.Registry
Dim k ' As regcol.RegistryKey
Dim v ' As regcol.RegistryValue
Set REG = CreateObject("RegCol.Registry")
Set k = REG.keys.Item
("\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Ads\Providers")
For Each k In k.keys
    say k.Name
    For Each v In k.values
        say v.Name & "=" & v.Data
    Next
Next
```

Listing 11.1: Liste der installierten ADSI-Provider [regcol_adsiprovider.wsf]

```
Set REG = CreateObject("RegCol.Registry")
r.Connect "SONNE2000"
For Each k In r.keys
    say k.Name
Next
```

Listing 11.2: Liste der HKEYs auf einem entfernten System [regcol_HKEYS.wsf]

Änderungen in der Registry

Die folgende Hilfsroutine übergibt ausgehend von einem übergebenen RegistryKey-Objekt rekursiv alle Werte und Unterschlüssel. Die Anzahl der Schlüssel und Werte wird dabei gezählt und als Referenzparameter an den Aufrufer zurückgegeben.

**Rekursive
Ausgabe**

```
' ### Rekursive Liste aller Unterschlüssel und Werte
Sub regcol_rek(key, ebene, keyZaehler, valZaehler)
Dim subkey ' As RegistryKey
Dim v ' As RegistryValue
keyZaehler = keyZaehler + 1
say keyZaehler & Space(ebene * 2) & key.Name
For Each v In key.values
    valZaehler = valZaehler + 1
    say valZaehler & Space(ebene * 2) & "- " & v.Name & "=" & v.Data
Next
For Each subkey In key.keys
    regcol_rek subkey, ebene + 1, keyZaehler, valZaehler
Next
End Sub
```

Listing 11.3: Rekursive Liste aller Unterschlüssel und Werte

```
Dim keyZaehler, valZaehler
Dim REG ' As regcol.Registry
Dim k ' As regcol.RegistryKey
Set REG = CreateObject("RegCol.Registry")
Set k = REG.keys.Item("\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\ADs")
regcol_rek k, 0, keyZaehler, valZaehler
say "Anzahl der Schlüssel:" & keyZaehler
say "Anzahl der Werte:" & valZaehler
```

Listing 11.4: Test der Routine regcol_rek(key, ebene) [regcol_rek]

11.3 Änderungen in der Registry

Das Beispiel zeigt das Anlegen eines Schlüssels mit einem Unterschlüssel und zwei Werten. Danach wird ein Wert ausgegeben, geändert und wieder ausgegeben. Zum Schluss wird der gesamte angelegte Schlüssel wieder gelöscht, so dass ein erneuter Zugriff auf den Wert zu einem Fehler führt.

**Anlegen, Ändern,
Löschen**

```
Dim REG ' As regcol.Registry
Dim k ' As regcol.RegistryKey
Dim v ' As regcol.RegistryValue
Const dtREG_DWORD = 4
Const dtREG_SZ = 1
Set REG = CreateObject("RegCol.Registry")
' --- Neuer Schlüssel
Const firmenkey = "\HKEY_LOCAL_MACHINE\Software\IT-Visions\"
```

Registry-Zugriff mit RegCol

```
Set k = REG.keys.Add(firmenkey)
' --- Neuer Unterschlüssel
Set k = k.keys.Add("Windows Scripting")
' -- Zwei neue Werte über den Unterschlüssel
Set v = k.values.Add("", "Registry-Eintrag zum Buch", dtREG_SZ)
Set v = k.values.Add("Autor", "Holger Schwichtenberg", dtREG_SZ)
Set v = k.values.Add("Erscheinungsjahr", 2000, dtREG_DWORD)
' -- Wert ausgeben
say REG.Item(firmenkey & "\Windows-scripting\Autor").Data
' -- Wert ändern und ausgeben
k.values("Autor") = "H.Schwichtenberg"
say REG.Item(firmenkey & "\Windows-scripting\Autor").Data
' -- Komplette Einträge wieder löschen
REG.keys.Remove (firmenkey)
' -- Ausgabe führt nun zu Fehler: Invalid Key
say REG.Item(firmenkey & "\Windows-scripting\Autor").Data
```

Listing 11.5: Beispiel zum Anlegen und Löschen von Schlüsseln und Werten mit RegCol [regcol_add.wsf]



Pro Schlüssel ist ein Wert ohne Namen erlaubt, der so genannte *Standardwert*. Den Standardwert eines Schlüssels beschreiben Sie, indem Sie statt eines Wertnamens einen Leerstring übergeben.

12 Zugriff auf die NT-Ereignisprotokolle

Für den Zugriff auf die Ereignisprotokolle in einem NT4- oder Windows2000-System werden hier zwei Lösungen vorgestellt:

- *STMAAdmin* realisiert ein Objektmodell für den Lese- und Schreibzugriff auf die Ereignisprotokolle.
- Der *EventLog OLE DB-Provider* der Firma ACES erlaubt nur den Lesezugriff.



Abbildung 12.1:
Typischer Eintrag
im System-Protokoll

12.1 EventLog-Zugriff mit STMAAdmin

STMAAdmin ist eine COM-Komponente, die unter Abstraktion von den komplexen Routinen des Win32-API ein einfaches Objektmodell zum Zugriff auf das Ereignisprotokoll des Windows NT-Betriebssystems bereitstellt.

**Automatisierung
des Ereignis-
protokolls**

Wie der NT4-Ereignisanzeige (EVENTVWR.EXE) bzw. das Windows2000-MMC-Snap-In »Ereignisanzeige« kann auch STMAAdmin remote auf Ereignisprotokolle anderer Rechner zugreifen, sofern das Benutzerkonto, unter dem das Programm abläuft, die nötigen administrativen Rechte hat.

Zugriff auf die NT-Ereignisprotokolle

Tabelle 12.1:
Schnellinfo Event-
Log-Zugriff mit
STMAdmin

Name und Abkürzung	STMAdmin Objects
Name der Komponentendatei	STMAdmin.dll
Interner Name der Typbibliothek	STMADMINLib
Helpstring der Typbibliothek	STMAdmin 1.0 Type Library
Hersteller	Sean McLeod, sean_mcleod@bigfoot.com
Lizenzierung	Freeware
Besprochene Version	1.0
NT4, Windows2000 und Windows 95/98/ME	Einzelregistrierung der DLL: regsvr32.exe STMAdmin.dll
Dokumentation	Nicht vorhanden

Abbildung 12.2:
Typbibliothek im
VB6-Objektkatalog



Objektmodell

Klassen STMAdmin implementiert vier Klassen:

- ▶ EventLog repräsentiert eines der Ereignisprotokolle auf einem bestimmten Rechner.
- ▶ EventLogRecords ist eine Collection von EventLogRecord-Objekten.
- ▶ Ein EventLogRecord ist ein einzelner Eintrag in einem Ereignisprotokoll.
- ▶ EventLogRecordStrings ist eine Collection von Zeichenketten, die zu einem bestimmten EventLogRecord abgelegt sind.

Abbildung 12.3 zeigt das Objektmodell. Lediglich das Stammobjekt EventLog-Klasse ist von außen instanzierbar. Alle anderen Klassen sind abhängige Klassen, die nur von den übergeordneten Klassen instanziiert werden können. Ein EventLogRecord-Objekt enthält ein Attribut mit dem Namen Records. Records verweist auf eine EventLogRecords-Collection, die wiederum die einzelnen EventLogRecord-Objekte enthält. Die Klasse EventLogRecords ist eine Erweiterung der Visual Basic-Standard-Collection, da zusätzlich eine Sortierfunktion implementiert ist. Die Zahl Acht in dem Ausdruck `objEventLog.Records(8)` bedeutet dabei die Sortierreihenfolge »Rückwärts« (vom jüngsten zum ältesten Eintrag absteigend), während eine Vier die Sortierung vom ältesten zum jüngsten initiieren würde.

Objektmodell

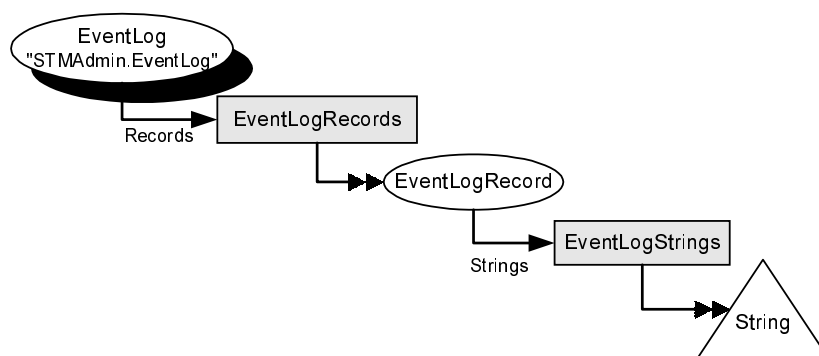


Abbildung 12.3:
Objektmodell von
STMAAdmin

Log öffnen

Die Methode `Open()` eines zuvor mit `CreateObject()` instanziierten EventLog-Objekts öffnet ein Ereignisprotokoll auf einem bestimmten Computer.

Open()

LogName sind im Normalfall *System*, *Application* und *Security*. Auf einem Windows2000-Server gibt es je nach Installationsumfang weitere Ereignisprotokolle (*Directory Service*, *DNS Server*, *File Replication Service*), die über STMAAdmin geöffnet werden können. Die Protokolle *Security* und *Application* können auf einem deutschen System auch über die deutschen Begriffe *Sicherheit* und *Anwendung* angesprochen werden. Ohne den optionalen Computernamen erfolgt der Zugriff auf das lokale Protokoll.

```
EventLog.Open LogName, [Computer]
```

Einträge erzeugen

Die Methode `ReportEvent()` erzeugt einen neuen Eintrag. Die Syntax lautet:

ReportEvent()

```
EventLog.ReportEvent(strSourceName, intEventID,  
strEventDescription, [intEventType], [strMachineName] )
```

Zugriff auf die NT-Ereignisprotokolle

Ereignistypen Während `strSourceName`, `intEventID` und `strEventDescription` frei wählbar sind, gibt es sechs vordefinierte Event-Typen: *SUCCESS* (0), *ERROR* (1), *WARNING* (2), *INFORMATION* (4), *AUDIT_SUCCESS* (8) und *AUDIT_FAILURE* (16).

Beispiele

Erzeugung eines Eintrags Listing 12.1 zeigt die Erzeugung eines neuen Log-Eintrags vom Typ *Information* im Anwendungsprotokoll. Listing 12.2 zeigt das Auslesen des Systemprotokolls.

```
Dim EventLog ' As STMADMINLib.EventLog
Dim rec ' As STMADMINLib.EventLogRecord
Dim str ' As Variant
Set EventLog = CreateObject("STMAAdmin.EventLog")
EventLog.Open "Anwendung", "Sonne2000"
' -- Erzeugen eines neuen Eintrags (Typ: Information)
EventLog.ReportEvent "WindowsScripting", 123, "AllesOK!", 4
```

Listing 12.1: Erzeugung eines neuen Log-Eintrags [STMAAdmin_CreateAndRead.wsf]

```
Set EventLog = CreateObject("STMAAdmin.EventLog")
EventLog.Open "System", "Mars"
' -- Ausgabe der Anzahl der Einträge
say "Anzahl der Einträge: " & EventLog.NoRecords
' Iteration über alle Einträge
For Each objRec In EventLog.Records(8)
' Ausgabe eines einzelnen Eintrags
  say "-----"
  say "EventID: " & objRec.EventID
  say "Name der Quelle: " & objRec.SourceName
  say "Computername: " & objRec.ComputerName
  say "Event Description: " & objRec.Event
  say "Anzahl Details-Strings: " & objRec.numStrings
  say "Datum/Uhrzeit: " & DateAdd("s", objRec.TimeGenerated, _
" 1.1.1970")
  a = 1 ' Laufende Nummer des Strings
  say "Strings: "
  For Each str In objRec.Strings
    say a & ": " & str
    a = a + 1
  Next
Next
' -- Log schließen
EventLog.Close
```

Listing 12.2: Auslesen des System-Logs mit der STMAAdmin-Komponente [STMAAdmin_CreateAndRead.wsf]

OLE DB-Provider für das NT-EventLog

Der von `objRec.TimeGenerated` zurückgelieferte Wert repräsentiert die Anzahl der Sekunden, die seit dem 1.1.1970 vergangen sind, und muss daher zunächst mit der VB-Funktion `DateAdd()` bearbeitet werden. Die Schleife bricht vorzeitig ab, wenn die Anzahl der maximal auszugebenden Einträge erreicht ist. Diesen Wert kann der Benutzer mit Hilfe eines Textfelds und des AKTUALISIEREN-Buttons beeinflussen.

Zeitangaben

Wichtig ist der Unterschied zwischen den Attributen `objRec.Event` und `objRec.Strings`. Während `objRec.Event` einen kompletten Beschreibungstext des Ereignisses enthält, wie er auch im EventViewer angezeigt wird, ist `objRec.Strings` eine Liste von Detailinformationen zu diesem Event. Diese Detailinformationen sind in `objRec.Event` enthalten, jedoch über die Strings-Collection einfacher auswertbar. `Strings` ist eine Collection vom Typ `EventLogRecordStrings`, die nicht Objekte, sondern Zeichenketten enthält.

`objRec.Event`
versus `rec.Strings`

12.2 OLE DB-Provider für das NT-EventLog

Die Firma *ACES Development* bietet im Rahmen ihres Produkts *ACES WinNT Automation Kit* [ACE00] einen OLE DB-Provider an, der einen lesenden Zugriff auf das NT-EventLog ermöglicht. Eine Evaluationsversion ist kostenlos erhältlich [CD:/install/komponenten/ACES/], liefert jedoch bei jeder Objektinstanziierung eine unangenehme Dialogbox.

Per ADO ins Ereignisprotokoll

Connection String für Ereignisprotokolle

Die ProgID des Providers ist *EventLogOLEDB.1*. Als weitere Angaben benötigt der Provider den Namen des Servers und den Typ des EventLogs, z.B. *System*, *Application* oder *Security*. Es gibt zwei Möglichkeiten, diese Angaben an den Provider zu übergeben. Der Name des Servers kann in einem Connection-String als `Location`-Attribut übergeben werden; dann steht an der Stelle des SQL-Statements nur der Protokolltyp.

Connection String

```
conString = "Provider=EventLogOLEDB.1;Location=Server1"
sql = "logtyp"
rs.Open sql, conString
```

In der zweiten Fassung werden Servername und Logname als UNC-Pfad an der Stelle des SQL-Statements übergeben.

```
conString = "Provider=EventLogOLEDB.1 "
sql = "\\server\logtyp"
rs.Open sql, conString
```

Zugriff auf die NT-Ereignisprotokolle

Beispiel

Das folgende Beispiel demonstriert die Anwendung des Providers via ActiveX Data Objects (ADO):

```
Set rs = CreateObject("ADODB.Recordset")
ConString = "Provider=EventLogOLEDB.1"
rs.Open "\\mars\system", ConString
Do While Not rs.EOF
    For Each f In rs.Fields
        say f.Name & "=" & rs(f.Name)
    Next
Loop
rs.MoveNext
rs.Close
```

Listing 12.3: Verwendung des OLE DB-Providers für das EventLog [ado_EventLog.wsf]

Feldbeschreibung

Feldnamen Die Tabelle zeigt die Feldnamen, die der Provider bereitstellt. Die Feldnamen werden klarer, wenn man diese mit einem typischen Ereignisprotokoll-eintrag vergleicht.

*Tabelle 12.2:
Felder in einem
EventLog-Recordset*

Feldname	Datentyp	Erläuterung
Date	Date/Time	Datum und Uhrzeit des Ereignisses
Id	Long	Eindeutige Nummer des Ereignisses
Type	Long	Art des Ereignisses: 1 – Fehler; 2 – Warnung; 4 – Information; 8 – Überwachung Erfolg; 16 – Überwachung Fehler
Category	Long	Kategorie (von der Ereignisquelle unterschiedlich belegt)
Source	String	Name der Systemkomponente, die das Ereignis ausgelöst hat
Computer	String	Name des Computers, der das Ereignis ausgelöst hat
User	String	Name des Benutzers, der das Ereignis ausgelöst hat
Descr	String	Beschreibung des Ereignisses
Data	Binary	Von der Quelle übermittelte Daten in binärer Form. Diese werden nicht im Ereignisprotokoll angezeigt.

13 Active Directory Service Interface (ADSI)

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. Als konkrete Verzeichnisdienste werden NT4, Exchange 5.5, Windows2000 Active Directory und der Internet Information Server 4.0/5.0/6.0 vorgestellt. Kapitel 13.11 liefert die Darstellung von Zusatzkomponenten, die in engem Zusammenhang mit ADIS stehen..

ADSI

Mit ADSI stellt Microsoft endlich eine leistungsstarke Schnittstelle zur Automatisierung vieler administrativer Aufgaben zur Verfügung. Man merkt allerdings, dass ADSI eigentlich für Windows2000 geschrieben wurde. Die Möglichkeiten unter NT4 sind eingeschränkt.



13.1 Verzeichnisdienste

Ein Verzeichnis ist – sehr allgemein gesagt – ein hierarchischer Speicher für Informationen. Der Begriff *Verzeichnisdienst* umfasst neben der Speicherung auch den Zugriff auf die Informationen.

Was ist ein Verzeichnisdienst?

Verzeichnisdienste werden im Enterprise Computing immer populärer. Ein Verzeichnisdienst (*Directory Service*) dient in Unternehmen der Verwaltung von Benutzern und Ressourcen (z.B. Computer, Drucker, 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 zu Grunde, 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

X.500

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 auf Grund der Komplexität des Standards nur in wenigen Verzeichnisdiensten vorhanden. Weitere Informationen zu X.500 finden Sie in [CHA96] und [TEB00].

Verzeichnisdienste in der Praxis

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.

13.1.1 Aufbau eines Verzeichnisbaums

Verzeichnisbäume

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.

Namespace

Ein Namespace ist eine Zusammenfassung mehrerer Verzeichnisbäume eines Verzeichnisdienstes. Beispielsweise bildet Windows NT 4.0 einen Namespace, der aus den Domänen und Arbeitsgruppen besteht.

Attribute

Jeder Eintrag in einem Verzeichnisbaum hat bestimmte Attribute. Jedes Attribut hat einen Wert, manche Verzeichnisdienste erlauben auch Mehrfachwerte für Attribute z.B. Ablage mehrerer Nummern für die Telefonnummer).

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.

Verzeichnisdienste

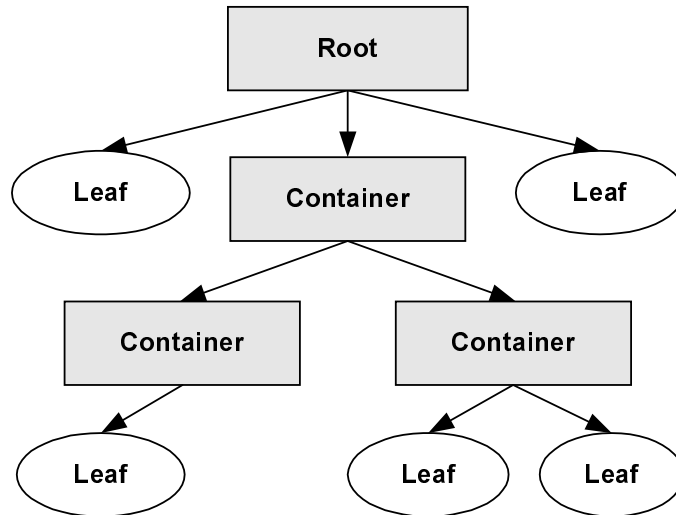


Abbildung 13.1:
Aufbau eines
Verzeichnisdienstes

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 einem Message Store (vgl. Kapitel 18 zu CDO). 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

13.1.2 Verzeichnisdienstschemata

Verzeichnisdienste besitzen ein Schema, das eine abstrakte Beschreibung des Verzeichnisdienstes auf einer Metaebene darstellt. Das Schema ähnelt einem Wörterbuch und enthält:

- Die Definition der Attribute und ihrer Syntax

Die Syntax besteht aus den erlaubten Datentypen, dem erlaubten Wertebereich und der Unterscheidung zwischen einem Einfach- und einem Mehrfachattribut.

Attribute

	► 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).
	► Die Vererbungshierarchie der Klassen
Vererbungshierarchie	Auch in Verzeichnisdiensten können Klassen von anderen Klassen abgeleitet sein. Meist ist auch Mehrfachvererbung möglich.
	► Die Containment-Hierarchie
Containment-Hierarchie	Es wird definiert, welche Instanzen einer anderen Klasse ein konkretes Objekt enthalten können (Beispiel: ein Eintrag <i>Familie</i> kann Objekte der Klasse <i>Kinder</i> enthalten, nicht aber Objekte der Klasse <i>Garten</i>). 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 <i>Schemamodifikation</i> .

13.1.3 Objektidentifikation

DN und RDN	Verzeichnisdienste sind in der Regel hierarchisch. Einzelne Einträge in einem Verzeichnisdienst werden daher über hierarchische Pfadangaben, so genannte <i>Fully Qualified Distinguished Names (FQDN)</i> oder einfacher <i>Distinguished Names (DN)</i> gebildet. Davon abzugrenzen ist der <i>Relative Distinguished Name (RDN)</i> . 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.

13.1.4 Meta-Directories

Meta-Directories	Ein Meta-Directory ist ein Dienst, der verschiedene Verzeichnisdienste zu einem zusammenfasst. Anstatt Konnektoren zwischen verschiedenen Ver-

zeichnisdiensten 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* angekündigt.

13.1.5 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 auf Grund 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 (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 Verzeichnisse 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. Windows2000 Active Directory, Microsoft Exchange seit Version 5.0, Microsoft Site Server, Lotus Notes, Netscape Directory Server).

Akzeptanz

LDAP-Clients

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:

LDAP-Anwendungen

- ▶ durch Eingabe eines beliebigen LDAP-URLs an der URL-Zeile des Internet Explorers (z.B. `LDAP://kom.wi-inf.uni-essen.de`)

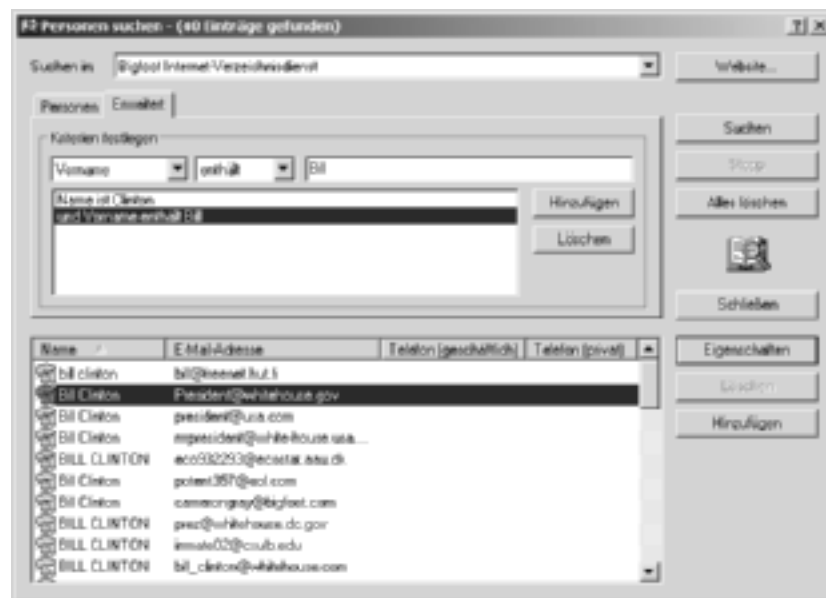
LDAP im IE

Active Directory Service Interface (ADSI)

- ▶ 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 13.2:
Die Suche nach
Bill Clinton liefert
einige Ergebnisse im
LDAP-Client des
Internet Explorers
5.0.



ADB Der mit der ADSI-Installation mitgelieferte *Active Directory Browser* (ADB) ist dagegen ein vollwertiger LDAP-Client. Der ADB wird in Kapitel 13.4 vorgestellt.

LDAP-Programmierung

WLDAP32.DLL 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 DCOM-Komponenten zu abstrahieren, die auch für den Einsatz in Scripting-Umgebungen geeignet sind, wird LDAP durch die COM-Komponente ADSI gekapselt.

LDAP-Standardattributnamen

Attribute 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.

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 13.1:
Ausgewählte
LDAP-Standard-
attribute für
Benutzer

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 dass mit diesem Parameter zwischen den Servern gewählt werden kann. Der Aufbau des LDAP-URLs ist in [RFC1959] dokumentiert.

LDAP-URLs

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*.

Attribut-Wert-Paare

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 Eintrags 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.

Ein Schlüsselattribut für jede Klasse

Tabelle 13.2:
Beispiele für
LDAP-Pfade

Eintragsart	Beispiel Top-Down-Syntax (auch: Big Endian)	Beispiel Bottom-Up-Syntax (auch: Little Endian)
Root-Element eines LDAP-Servers	LDAP://SONNE	LDAP://SONNE
Recipients-Container eines Exchange Servers	LDAP://SONNE/ O=UNIVERSUM/ OU=SONNENSYSTEM/ CN=RECIPIENTS/ CN=MITARBEITER	LDAP://SONNE/ CN=MITARBEITER, CN=RECIPIENTS, OU=SONNENSYSTEM, O=UNIVERSUM
Benutzer in einem Active Directory	LDAP://SONNE2000/ DC=DE/ DC=IT-VISIONS/ 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=SONNENSYSTEM/ CN=CONFIGURATION/ CN=PROTOCOLS/ CN=POP3	LDAP://SONNE/ CN=POP3, CN=PROTOCOLS, CN=CONFIGURATION, OU=SONNENSYSTEM, O=UNIVERSUM

13.2 Die ADSI-Grundlagen

ADSI-Architektur

Das *Active Directory Service Interface* (ADSI) ist eine COM-Komponente für den Zugriff auf Verzeichnisdienste. Dabei ist ADSI nicht, wie der Name vermuten ließe, auf das Active Directory in Windows2000 beschränkt. Die ADSI-Architektur (siehe Abbildung 13.3) 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.

Tabelle 13.3:
Schnellinfo ADSI-
Komponente

Name und Abkürzung	Active Directory Service Interface (ADSI)
DLL	activeds.DLL (Typbibliothek: activeds.tlb)
Name der Typelibrary	Active DS
Helpstring der Typelibrary	Active DS Type Library
Hersteller	Microsoft
Besprochene Version	2.5
NT4	Installation des ADSI-Add-ons, Version 2.5 [CD:/install/komponenten/adsi/]

Windows 95/98/ME	Installation in einer speziellen Version des ADSI-Add-ons, Version 2.5 [CD:/install/komponenten/ads/]
Windows2000	Version 2.5 enthalten
Whistler (Windows XP / Windows.NET)	Version 2.5 enthalten
Dokumentation im MSDN	MSDN Library [MSL00] »Messaging and Collaboration«

Tabelle 13.3:
Schnellinfo ADSI-
Komponente
(Forts.)

13.2.1 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 ein Programm, das ADSI nutzt, ADSI-Client genannt. Da ADSI in weiten Teilen COM-Automation unterstützt, können auch Skripte als ADSI-Clients eingesetzt werden.

**ADSI-Provider sind
Stellvertreter**

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 Namespace-ID.

Vergleich mit ADO

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 (vgl. Kapitel 13.11) und CDO for Exchange Management (vgl. Kapitel 19.4). ADSI-Extensions können allerdings nicht mit Skriptsprachen oder Visual Basic entwickelt werden.

ADSI-Extensions

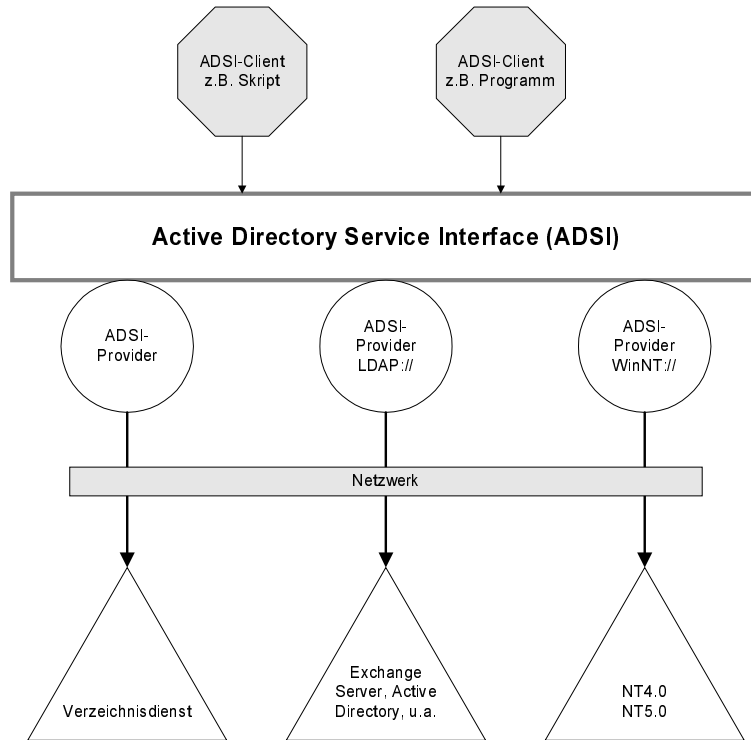
13.2.2 Installation

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 Windows2000, unabhängig davon, ob das Windows2000 Active Directory installiert ist. Die entsprechenden Add-ons für die anderen Plattformen finden Sie auf der Buch-CD [CD:/install/komponenten/ads/]. Kommende Updates erhalten Sie auch über die Microsoft ADSI-Website [ADS00]. Aktuell ist die ADSI-Version 2.5.

**Verfügbarkeit für
verschiedene
Plattformen**

Active Directory Service Interface (ADSI)

Abbildung 13.3:
ADSI-Architektur



(c) Holger Schwichtenberg 2000



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.

13.2.3 Verfügbare ADSI-Provider

Für ADSI 2.5 liefert Microsoft folgende ADSI-Provider:

Tabelle 13.4:
Verfügbare ADSI-
Provider von
Microsoft

Provider	Namespace-ID	Provider-DLL
Windows NT	WINNT	ADSNT.DLL
Lightweight Directory Access Protocol	LDAP und GC	ADSLDP.DLL
Novell NetWare 3.x	NWCOMPAT	ADSNW.DLL
Netware Directory Service 4.x	NDS	ADSNDS.DLL
Microsoft Internet Information Server Metabase	IIS	ADSIIS.DLL

Mit Ausnahme des IIS-Providers werden alle Provider durch das ADSI-Add-on bzw. durch das Windows2000-Setup installiert. Der IIS-Provider wird mit dem Internet Information Server bzw. den entsprechenden Client-Tools (MMC-Snap-In) installiert.

**Ausnahme
IIS-Provider**

Eine Liste der installierten ADSI-Provider befindet sich in der Registry unter HKEY_LOCAL_MACHINE\SOFTWARE\MICROSOFT\ADS\PROVIDERS. Die obige Tabelle enthält auch die Namespace-ID, die aus Sicht von COM die ProgID der Providerkomponente 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 (siehe Kapitel 2).

**Installierte
Provider**

Mit dem LDAP-Provider ist grundsätzlich der Zugriff auf alle LDAP-fähigen Verzeichnisdienste möglich, also insbesondere auf Windows2000 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.

LDAP und GC

Der WinNT-Provider dient sowohl dem Zugriff auf ein Windows NT4-Verzeichnis als auch dem auf ein Windows2000-System ohne installiertes Active Directory. Ein Windows2000-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.

WinNT

13.2.4 ADSI Software Development Kit

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, siehe Kapitel 13.4). Das ADSI-SDK ist Teil des Plattform-SDKs im Microsoft Developer Network (MSDN), aber auch separat verfügbar [ADS00]. 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.

ADSI-SDK

13.3 Das ADSI-Meta-Objektmodell

Einen Verzeichnisdienst in einer COM-Komponente abzubilden ist ein naheliegender Ansatz, da Syntax und Semantik von Verzeichnisdienstklassen und -objekten 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 die API des Verzeichnisdienstes bereitgestellt.

ADSI und COM

Active Directory Service Interface (ADSI)

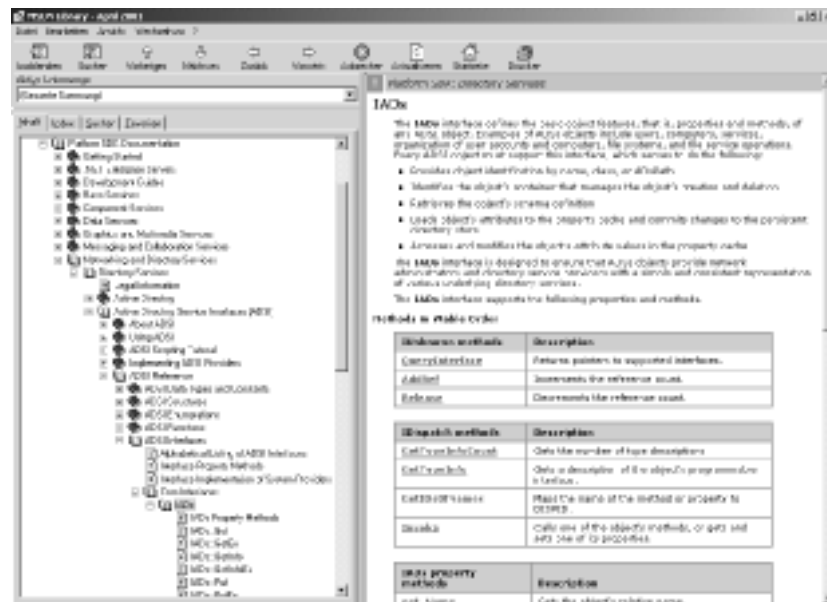
Standard-schnittstellen

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 (vgl. Kapitel 2) 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

Abbildung 13.4:
Die MSDN Library
dokumentiert zu
ADSI Schnittstellen,
keine Klassen.



Meta-Schnittstellen

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.

Die in Abbildung 13.5 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).

Beispiel

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 Eigenschaften 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

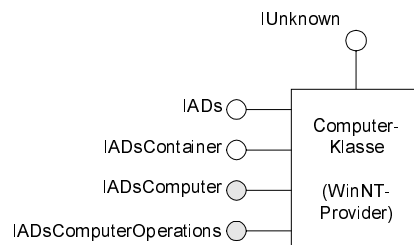


Abbildung 13.5:
Eine ADSI-Klasse
mit Meta-Schnittstellen und konkreten Schnittstellen

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.

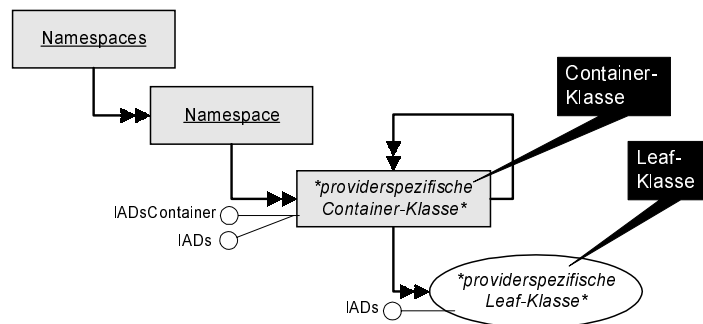
Skriptentwickler haben es einfacher

Meta-Objektmodell

Meta-Objektmodell

Die Abbildung 13.6 zeigt das ADSI-Meta-Objektmodell. Neben den konkreten Container-Klassen `Namespaces` und `Namespace` enthält das Meta-Objektmodell eine Container-Klasse und eine Leaf-Klasse. Beide haben keinen konkreten Namen, da diese Klasse ja erst durch den Provider realisiert wird. Dargestellt sind aber die Schnittstellen, die Container- bzw. Leaf-Klasse implementieren müssen.

Abbildung 13.6:
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.

13.3.1 Überblick über die ADSI-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.

Schnittstellentypen

Die Standardschnittstellen lassen sich in fünf Gruppen einteilen:

- Schnittstellen, die Basisfunktionalitäten für ADSI bereitstellen und den Meta-Zugriff auf ADSI-Klassen ermöglichen (Tabelle 13.5)

Das ADSI-Meta-Objektmodell

- ▶ Schnittstellen, die persistente Daten in Verzeichnisobjekten widerspiegeln (Tabelle 13.6). Zu einigen Verzeichnisobjekten gibt es dabei eine weitere Schnittstelle, die Aktionen auf diesen Verzeichnisobjekten bereitstellt (Tabelle 13.7).
- ▶ Schnittstellen, die keine persistenten Daten in Verzeichnisobjekten widerspiegeln, sondern dynamische Daten beinhalten und Funktionen auf Verzeichnisobjekten implementieren. In 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 (Tabelle 13.8).
- ▶ Schnittstellen zur Implementierung von Sicherheitsfunktionalitäten (Tabelle 13.9)
- ▶ Schnittstellen zur Schemaverwaltung (Tabelle 13.10)

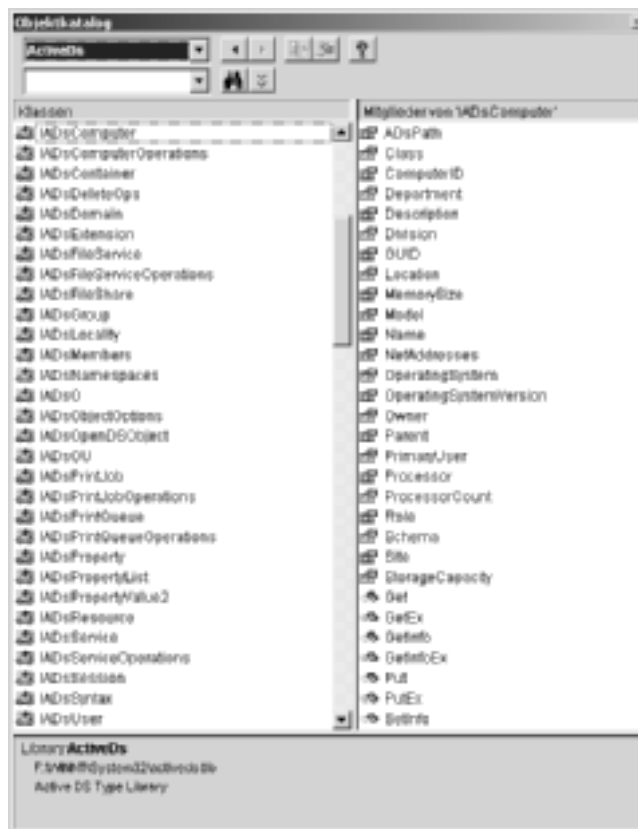


Abbildung 13.7:
Die wichtigsten
ADSI-Standard-
schnittstellen im
VB6-Objektkatalog

Active Directory Service Interface (ADSI)

Tabelle 13.5:
ADSI-Meta-
Schnittstellen

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 IADsContainer wird nur von Container-Klassen benutzt. IADsContainer realisiert Attribute und Methoden zur Verwaltung der enthaltenen Objekte.

Tabelle 13.6:
ADSI-Schnittstellen für persistente
Daten in
Verzeichnissen

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 13.7:
Zusatzschnittstellen für Methoden
auf persistenten
Verzeichnisobjekten

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 Fileservice; enthält Auflistungen der zugehörigen Sitzungen und Ressourcen

Das ADSI-Meta-Objektmodell

IADsPrintJob	Ein Druckauftrag in einer Druckerwarteschlange
IADsPrintJobOperations	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)

*Tabelle 13.8:
ADSI-Schnittstellen für nicht-persistente Daten in Verzeichnissen*

IADsOpenDSObject	Diese Schnittstelle dient der Impersonifizierung (d.h. dem Wechsel des Benutzerkontextes). <code>IADsOpenDSObject</code> stellt lediglich eine Methode zur Verfügung: <code>OpenDSObject()</code> . Bei <code>OpenDSObject()</code> 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.
IADsSecurityDescriptor	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).
IADsAccessControlEntry	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.
IADsAccessControlList	Liste aller <code>ControlEntries</code> , die eine ACL bilden.

*Tabelle 13.9:
ADSI-Schnittstellen, die der Sicherheit dienen*

IADsClass	Repräsentiert eine Klassendefinition in einem Verzeichnisdienstschema
IADsProperty	Repräsentiert eine Attributdefinition in einem Verzeichnisdienstschema
IADsSyntax	Repräsentiert eine Syntaxbeschreibung, die einem Attribut zugeordnet werden kann
IADsPropertyList IADsPropertyEntry IADsPropertyValue	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

*Tabelle 13.10:
ADSI-Schnittstellen zur Schemaverwaltung*

13.3.2 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 13.11 und 13.12). Alle Container-Objekte müssen zusätzlich `IADsContainer` implementieren (siehe dazu die Tabellen 13.13 und 13.14).

Tabelle 13.11:
Attribute der
Standardschnitt-
stelle `IADs`

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 13.12:
Methoden der
Standardschnitt-
stelle `IADs`

Methode	Erläuterung
<code>GetInfo()</code>	Einlesen aller Attribute eines Verzeichnisobjekts in den Cache
<code>GetInfoEx()</code>	Einlesen einzelner Attribute eines Verzeichnisobjekts in den Cache
<code>SetInfo()</code>	Schreiben des Cache in das Verzeichnisobjekt
<code>Get()</code>	Auslesen eines Attributs aus dem Cache; nur mehrwertige Attribute werden als <i>Array of Variant</i> zurückgegeben
<code>Put()</code>	Schreiben eines einwertigen Attributs in den Cache
<code>GetEx()</code>	Auslesen eines Attributs aus dem Cache. Im Gegensatz zu <code>GET()</code> werden alle Attribute als <i>Array of Variant</i> zurückgegeben.
<code>PutEx()</code>	Verändern eines mehrwertigen Attributs im Cache

Tabelle 13.13:
Attribute der Stan-
dardschnittstelle
`IADsContainer`

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. <code>Filter</code> erwartet als Parameter ein <i>Array of String</i> mit den Klassennamen.

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 RDN verlangen.
Create()	Erzeugen eines untergeordneten Objekts
Delete()	Löschen eines untergeordneten Objekts
CopyHere()	Kopieren eines ADSI-Objekts in diesen Container
MoveHere()	Verschieben eines ADSI-Objekts in diesen Container

Tabelle 13.14:
Methoden der Standardschnittstelle
IADsContainer

Einschränkungen mit CopyHere() und MoveHere()

CopyHere() und MoveHere() können nur innerhalb eines Verzeichnisdienstes 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 zwischen 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.



13.4 ADSI-Werkzeuge

Dieses Kapitel stellt nützliche Werkzeuge für das Active Directory Service Interface (ADSI) vor.

13.4.1 Active Directory Service Browser (ADB)

Um die Objekthierarchie eines Verzeichnisdienstes zu erkunden, ist der *Microsoft Active Directory Service Browser* (ADB) ein zweckmäßiges Werkzeug (siehe nächste Abbildung). Der ADB ist in den Installationspaketen zu ADSI enthalten und hat den Dateinamen ADSVW.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-Programmbeispiels, die nicht die volle Funktionalität des ADBs umfasst.

Ein Browser für
ADSI

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 Windows2000 Active Directory hat jedoch in seinem Schema mehr Einträge, so dass die Liste bei der Darstellung des Schemas abgeschnitten wird.

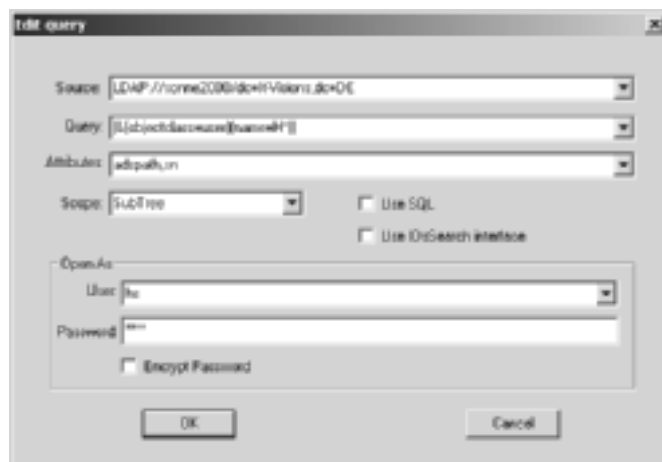


Der ADB ermöglicht sowohl das Browsing durch einen Verzeichnisbaum als auch die Ausführung von ADSI-Queries. Wählen Sie dazu FILE/NEW/QUERY.

Abbildung 13.8:
Ansicht eines Kon-
takt-Eintrags
in einem Active
Directory. Verzich-
ten muss man beim
ADB auf schöne
Icons. Dafür hat
man einen mächtigen
Thin-Client für
alle Verzeichnisse.



Abbildung 13.9:
Eingabe einer
ADSI-Query



13.4.2 ADSI Explorer

Verzeichnisdienste erforschen

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.

ADSI-Werkzeuge

Der *ADSI Explorer* ist ein kostenloses Werkzeug der Firma InTouchSoftware [ITS01]. Auf der Buch-CD finden Sie es unter [CD:/install/tools/ADSI/ADSI Explorer].

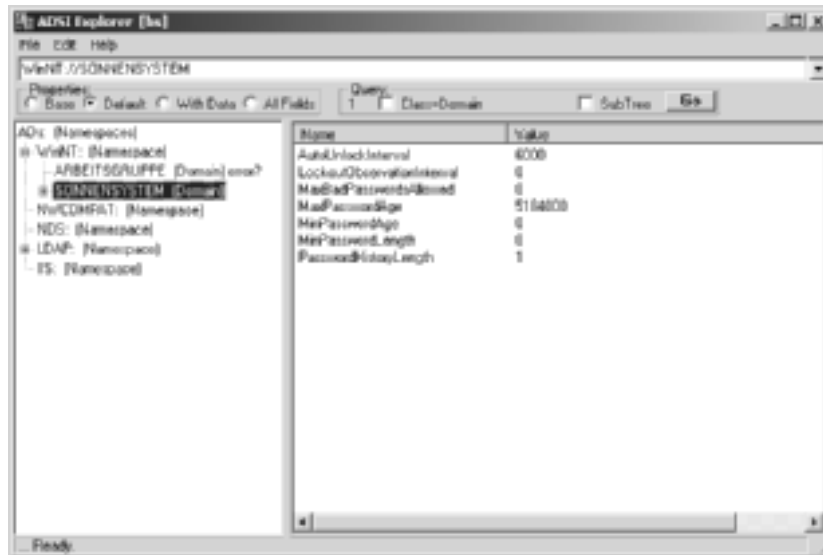


Abbildung 13.10:
Ansicht einer NT-
Domäne im ADSI
Explorer

13.4.3 ADSI Edit

ADSI Edit ist ein MMC Snap-In, das einen Low-Level-Zugriff auf das Active Directory ermöglicht. Verzeichnisobjekte können angezeigt, erzeugt, geändert, gelöscht und verschoben werden. *ADSI Edit* wird mit den auf der Windows2000 Server-CD enthaltenen Support Tools installiert.

Weiteres MMC-
Snap-In für AD

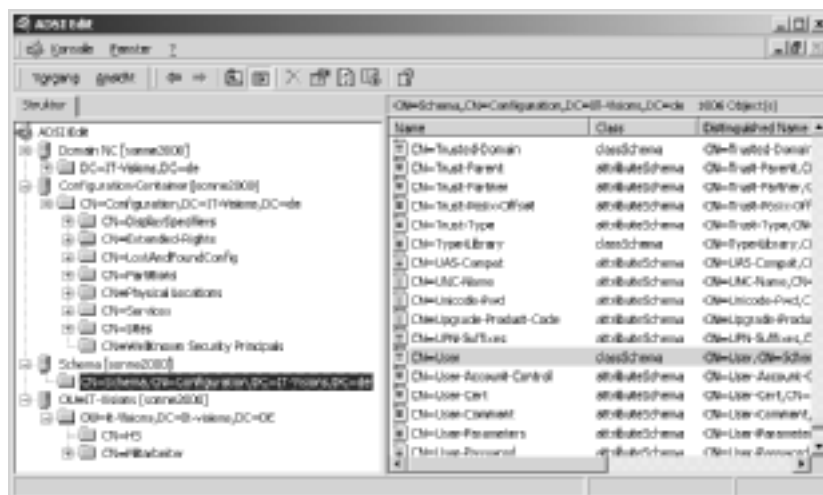


Abbildung 13.11:
Schemaansicht in
ADSI Edit

Active Directory Service Interface (ADSI)

Abbildung 13.12:
Flexibles Mapping
in ADSI Edit

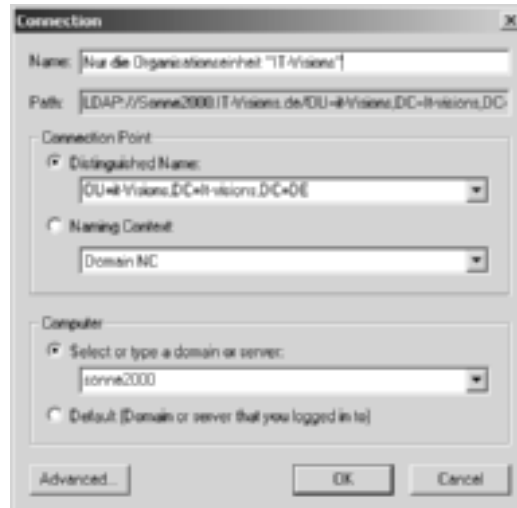
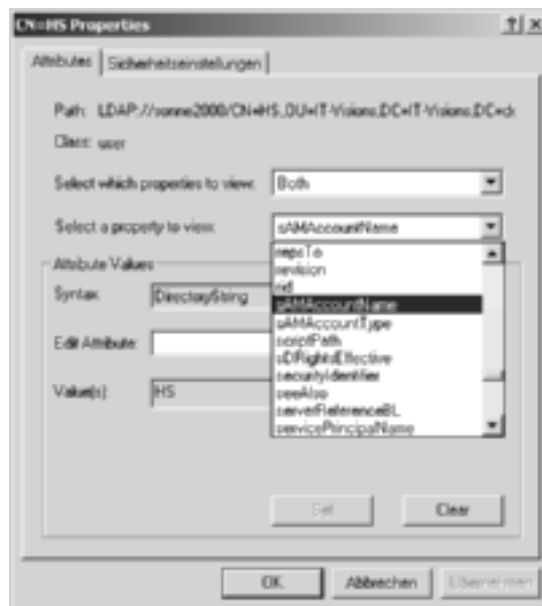


Abbildung 13.13:
Standardeigen-
schaftsfenster in
ADSI Edit



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 *DisplaySpecifiers* benötigt.

13.4.4 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/tools/adsi/schemabrowser/].



Abbildung 13.14:
ADSI Schema
Browser

13.5 Grundlagen der ADSI-Programmierung

Die in diesem Unterkapitel dargestellten Verfahren sind die Grundlage der Arbeit mit allen ADSI-Providern.

13.5.1 Objektidentifikation und Bindung

ADSI bedient sich COM-Moniker, um einzelne Einträge in verschiedenen Verzeichnisdiensten anzusprechen und einen Zeiger auf das Stellvertreter-Objekt zu erhalten. Der Moniker hat die Form

ADSI-Pfade

<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

DN und RDN

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.

Tabelle 13.15:
Beispiele für ADSI-
Pfade in verschiede-
nen Verzeichnis-
diensten

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	IIS://COMPUTERNAME/W3SVC/1



Bei einigen Providern kann der Klassenname des gewünschten Objekts durch ein Komma getrennt mit angegeben werden, falls Mehrdeutigkeiten zwischen den Unterobjekten bestehen (beispielsweise kann unter NT4 ein Computer genauso heißen wie ein Benutzer). Die Angabe des Klassennamens beschleunigt in einigen Fällen die Lokalisierung des Verzeichnisobjekts (z.B. beim WinNT-Provider bei der Frage, ob WinNT://name eine Domain oder ein Computer ist).

Bindung der Universalklasse an ein Verzeichnisobjekt

Bindung Voraussetzung für den Zugriff auf Objekte des Verzeichnisdienstes ist die Bindung eines ADSI-Objekts an ein Objekt des Verzeichnisdienstes. Die Bindung erfolgt nicht über die Instanziierung einer Klasse mit anschließendem Methodenaufruf, sondern über die bereits beschriebenen ADSI-Pfade (Moniker). In Visual Basic ist also die Funktion `GetObject()` zu verwenden.

```
Set objADS = GetObject(ADSIpfad)
```

Ausblick

Das Active Directory kennt noch einige besondere Formen des Bindens, z.B. das serverlose Binden oder das Binden über eine GUID.

Bindung über RDN

Ein konkretes Unterobjekt kann von einem Container aus auch über seinen relativen Namen (RDN) direkt angesprochen werden. Dazu dient die Methode `GetObject()` aus der `IADsContainer`-Schnittstelle. Allerdings ist dabei die Angabe des Klassennamens (hier: `user`) obligatorisch:

```
Set container = GetObject("WinNT://Mars")
Set objUser = container.GetObject("user", "A_vonKuensberg")
```

Der Zugriff auf das User-Objekt in einem Schritt würde dagegen lauten:

```
Set objUser = GetObject("WinNT://meinedomain/Mars/A_vonKuensberg")
```

Oder mit Angabe der Objektklasse, falls der Objektname nicht eindeutig ist:

```
Set objUser = GetObject("WinNT://Mars/Barbara,user")
```

Leider sind die Fehlermeldungen, die `GetObject()` bei erfolglosem Aufruf liefert, nicht immer aussagekräftig: Ein »Syntax Error« kann einerseits bedeuten, dass der ADSI-Pfad nicht korrekt aufgebaut ist, andererseits aber auch, dass das angesprochene Objekt nicht existiert.



Test auf Objektexistenz

Da ADSI keine eingebaute Funktion besitzt, um auf die Existenz eines Verzeichnisobjekts zu prüfen, kann im nächsten Listing abgedruckte Hilfsroutine `ExistsObject()` verwendet werden. Der Nachteil an dieser Try-and-Fail-Methode ist, dass die Entscheidung sehr lange dauern kann. Wird beispielsweise ein Rechner angesprochen, der nicht erreichbar ist, meldet `GetObject()` erst nach einigen Sekunden einen Fehler.

ExistsObject()

```
' ### Existiert eine COM-Instanz?  
Function existsObject(moniker) ' As Boolean  
Dim obj ' As Object  
On Error Resume Next  
Set obj = GetObject(moniker)  
If obj Is Nothing Then  
    ExistsObject = False  
Else  
    ExistsObject = True  
End If  
Err.Clear  
On Error GoTo 0  
End Function
```

Listing 13.1: Prüfen, ob eine COM-Instanz existiert
[CD:/code/komponenten/_Bibliotheken/ws_scriptlib.vbs]

Da ADSI keine eingebaute Funktion besitzt, um auf die Existenz eines Verzeichnisobjekts zu prüfen, kann die in Kapitel 3.18.7 vorgestellte Hilfsroutine `ExistsObject()` verwendet werden. Der Nachteil an dieser Try-and-Fail-Methode ist, dass die Entscheidung sehr lange dauern kann. Wird beispielsweise ein Rechner angesprochen, der nicht erreichbar ist, meldet `GetObject()` erst nach einigen Sekunden einen Fehler.

13.5.2 Methodenaufrufe und Attributzugriffe

**Unterschiede
beachten**

Bei ADSI gibt es einen wichtigen Unterschied zwischen dem Zugriff auf Methoden und dem Zugriff auf Attribute: Alle Methoden sind im direkten Zugriff (Punkt-Notation in Visual Basic), während auf Attribute zum Teil über Methoden der Meta-Schnittstelle *IADs* zugegriffen werden muss. Das liegt daran, dass ein Verzeichnisdienst keine eigenen Methoden bereitstellt. Die Methoden werden von ADSI-Schnittstellen zur Verfügung gestellt.

**Beispiel zum
Methodenaufruf**

Voraussetzung vor dem Zugriff auf eine Methode oder ein Attribut ist – natürlich – die Bindung einer Objektvariablen an ein Verzeichnisobjekt. Das folgende Beispiel zeigt den Aufruf von Methoden. So kann beispielsweise ein NT-Dienst wie der Scheduler mit ADSI gestartet und gestoppt werden, wie das nachfolgende WSH-Skript zeigt.

```
computer = Inputbox("Welcher Computer?")
Set servObj = GetObject("WinNT://" & computer & "/schedule")
servObj.Stop ' Dienst stoppen
servObj.Start ' Dienst starten
```

Listing 13.2: Bindung an den Scheduler-Dienst und Ausführung von zwei Methoden



In Sprachen, die frühe Bindung verwenden, kann es nötig sein, die Schnittstelle zu wechseln, da die Methoden auf ein Verzeichnisobjekt zum Teil durch verschiedene Schnittstellen bereitgestellt werden. Bei der Skriptprogrammierung bietet Ihnen die Standardschnittstelle *IADs* per *IDispatch*-Zugriff die volle Funktionalität (mit einigen wenigen Ausnahmen).

Attributzugriff

Direkter Zugriff

Auf ein Attribut kann nur direkt (über die Punkt-Notation) zugegriffen werden, sofern es in einer ADSI-Schnittstelle definiert ist.

```
Objektvariable.Attribut
```

Put(), Get()

Auf alle Verzeichnisattribute, für die es keine Entsprechung in ADSI-Standardschnittstellen gibt, müssen die *IADs*-Methoden *Put()* und *Get()* angewendet werden.

```
objMailbox.Put "Attributname", Wert
Variable = u.Get("Attributname")
```

ADSI unterscheidet zwischen zwei Arten von Attributen:

- Einwertige Attribute haben genau einen Wert.
- Mehrwertige Attribute können mehrere Werte gleichzeitig haben.

Ein Beispiel für ein mehrwertiges Attribut ist eine E-Mail-Adresse. Ein Benutzer kann neben der Adresse *Holger.Schwichtenberg@it-visions.de* auch noch Aliase haben, z.B. *hs@it-visions.de* und *hs@windows-scripting.de*.

**Mehrwertige
Attribute**

Anders als Datensätze in einer relationalen Datenbank besitzen Verzeichniseinträge optionale Attribute. In vielen Verzeichnisdiensten sind nicht-verwendete optionale Attribute nicht mit einem Nullwert belegt, sondern gelten als nicht-vorhanden. Ein Zugriff darauf führt zu einem Fehler (8000500d: »Die Active-Directory-Eigenschaft wurde nicht im Cache gefunden«). Dieser Fehler kann auch mit Attributen auftreten, die im Rahmen einer ADSI-Standardschnittstelle definiert sind, denn ADSI hat – wie bereits erwähnt – ein eher lockeres Verständnis vom Vertragscharakter einer Schnittstelle.

**Optionale
Attribute**

In diesem Umstand steckt eine heimtückische Falle, wenn Sie bei der Skriptentwicklung nicht beachten, dass in einer anderen Umgebung nicht notwendigerweise alle Attribute vorhanden sind, die es in Ihrer Testumgebung gibt. Es ist daher sinnvoll, den Attributzugriff fehlertolerant zu kapseln:



```
Function ADSIGet(obj, attribut)
ADSIGet = ""
On Error Resume Next
ADSIGet = obj.Get(attribut)
End Function
```

*Listing 13.3: Fehlertolerantes Auslesen eines Attributs in einem Verzeichnisobjekt
[ADSIGet.wsf]*

Mehrwertige Attribute

Mehrwertige Attribute werden über Arrays zugewiesen und benötigen die speziellen IADs-Methoden `GetEx()` und `PutEx()`. `PutEx()` kennt dabei vier verschiedene Modi (siehe folgende Tabelle).

**GetEx(),
PutEx()**

```
' -- Definition des Arrays
Dim mehrwert(2)
' -- Füllen des Arrays
mehrwert(0) = "H.Schwichtenberg@it-visions.de"
mehrwert(1) = "hs@windows-scripting.de"
mehrwert(2) = "hs@it-visions.de"
modus = 1
' -- Setzen des mehrwertigen Attributs
obj.PutEx modus, "AttributName", mehrwert
```

Listing 13.4: Pseudo-Code für den Zugriff auf ein mehrwertiges Attribut

*Tabelle 13.16:
Die vier Modi der
PutEx()-Methode.
Diese Konstanten
sind leider nicht in
der Typelibrary
definiert.*

Symbolische Konstante	Numerische Konstante	Erläuterung
ADS_PROPERTY_CLEAR	1	Löscht alle Werte
ADS_PROPERTY_UPDATE	2	Die übergebenen Werte ersetzen die alten Werte.
ADS_PROPERTY_APPEND	3	Die übergebenen Werte werden angefügt.
ADS_PROPERTY_DELETE	4	Die übergebenen Werte werden aus dem Attribut entfernt.

Sonderbehandlung für Arrays

Beim Auslesen mehrwertiger Attribute mit `Get()` ist eine Fallunterscheidung notwendig. Ein mehrwertiges Attribut, das nur einen Wert enthält, wird nicht als Array übergeben, sondern direkt als einzelner Wert. Die im folgenden Listing dokumentierte Hilfsfunktion `flat()` kann grundsätzlich eingesetzt werden, um einen mehrwertigen ADSI-Attributwert in einen String umzuwandeln. `GetEx()` liefert dagegen immer ein Array, auch dann, wenn das Attribut nur einwertig ist.

```
Function CSVadd(s1, s2) ' As String
If s1 <> "" Then s1 = s1 & ";"
s1 = s1 & s2
csvadd = s1
End Sub

Function flat(var) ' As String
On Error Resume Next
Dim i ' As Integer
If IsArray(var) Then ' Array flachklopfen
    flat = ""
    For i = LBound(var) To UBound(var)
        CSVadd flat, var(i)
    Next
Else ' War kein Array
    flat = var
End If
End Function
```

*Listing 13.5: Listing 13.2: Macht aus einem Array einen CSV-String.
[CD: /code/komponenten/_Bibliotheken /WS_ScriptLIB.vbs]*

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 `SetInfo()` (aus der IADs-Schnittstelle) abgeschlossen werden. Erst dann wird der Cache an

den zu Grunde liegenden Verzeichnisdienst übergeben. Damit wird auch die Transaktionssicherheit gewährleistet: Entweder werden alle Änderungen ausgeführt oder keine. Auch für das Einlesen der Attribute in den Cache gibt es eine Methode: `GetInfo()`. Das Skript sollte sie explizit aufrufen, wenn nicht sicher ist, ob die Werte im Cache noch aktuell sind. Mit `GetInfo()` können auch Änderungen verworfen werden, wenn zwischen den Änderungen und dem `GetInfo()` kein `SetInfo()` steht. Mit `GetInfoEx()` können vor einem ersten Attributzugriff gezielt einzelne Werte in den Cache gelesen werden, um zur Verringerung der Netzwerklast die Übertragung aller Attribute zu vermeiden.

```
Set UsrObj = GetObject("WinNT://Sonne/AnnikaRechmann")
' -- Deaktivierung
UsrObj.AccountDisabled = True
' -- Cache schreiben
UsrObj.SetInfo
say "Konto deaktiviert"
...
UsrObj.GetInfo
say "Zustand des Kontos: " & UseObj.AccountDisabled
```

Listing 13.6: Arbeit mit dem Property Cache

Probleme mit dem Datentyp

Bei bestimmten Konstellationen kann es zu Problemen hinsichtlich des Datentyps bei der Übergabe von Werten an `Put()` kommen. So übergibt VBScript eine Variable vom Subtyp *Variant* per Zeiger; der LDAP-Provider unterstützt aber diese Form der Übergabe nicht. Sie müssen mit einem Trick VBScript dazu zwingen, keinen Zeiger, sondern den Wert direkt zu übergeben.

**Notwendige
Typkonvertie-
rungen**

- Eine Möglichkeit ist, beim Aufruf von `Put()` die Variable explizit in den passenden Subtyp zu konvertieren.

```
u.Put "samAccountName", CStr(un)
objRecipient.Put "mailPreferenceOption", CInt(0)
```

- Eine andere Möglichkeit besteht darin, den Wert einfach in Klammern zu setzen. Wie in Kapitel 3 beschrieben, sieht VBScript dann einen Ausdruck, der ausgewertet wird, und ein Ausdruck wird immer als sein Wert übergeben, nicht als Zeiger.

```
u.Put "samAccountName", (un)
```

Natürlich ist die erste Möglichkeit die bessere.

13.5.3 Zugriff auf Container-Objekte

Der Zugriff auf die Container-Objekte und ihre Eigenschaften ist vollkommen identisch mit dem Zugriff auf Leaf-Objekte. Die Iteration durch ADSI-

Container-Objekte

Active Directory Service Interface (ADSI)

Container erfolgt in Visual Basic analog zur Iteration durch Collections mit der For...Each-Schleife. Sie bindet die Laufvariable obj nacheinander an die Objekte im Container:

```
Set container = GetObject("WinNT://Sonne")
For Each obj In container
    say obj.class & ":" & obj.name
Next
```

Listing 13.7: Iteration über einen Computer-Container im WinNT-Provider

Filter Die For...Each-Schleife durchläuft grundsätzlich alle in dem Container enthaltenen Objekte. Eine Einschränkung kann jedoch mit dem Filter-Attribut aus der IADsContainer-Schnittstelle gegeben sein. Das Filter-Attribut erwartet ein Array mit den Namen der Klassen, auf die die Objektmenge beschränkt werden soll.

```
Set container = GetObject("WinNT://Sonne")
container.Filter = Array("User", "Group")
For Each u In container
    say u.name
Next
```

Listing 13.8: Filterung eines Containerinhalts: Der Inhalt des Computer-Containers wird auf die Klassen User und Group eingeschränkt.



Leider trifft man auch in ADSI 2.5 noch auf einen bekannten Bug: ADSI hat Probleme mit der Grundschulmathematik und verzählt sich beim Auflisten eines Containers mit For...Each gelegentlich. Obwohl bereits alle Unterobjekte durchlaufen wurden, wird eine neue Iteration begonnen, die dann aber beim ersten Objektzugriff mit einem »Objektvariable nicht gesetzt« jäh beendet wird. Dagegen hilft nur eine tolerante Fehlerbehandlungsroutine.

Rekursion über einen Verzeichnisdienst-Container

**Rekursion mit
ADSI**

Eine interessante Aufgabe ist auch hier die rekursive Ausgabe eines Verzeichnisbaums. Bei der Skriptprogrammierung gibt es ein Problem bei der Feststellung, ob ein Objekt ein Container oder ein Blatt ist, denn ein Test auf die Existenz der IADsContainer-Schnittstelle kann in der typenlosen Welt von VBS nicht durchgeführt werden. Eine Möglichkeit wäre der Zugriff auf die Schemainformationen. Der hier dargestellte Weg ist etwas direkter: Es wird die Iteration über ein Verzeichnisobjekt versucht. Schlägt sie fehl, handelt es sich offensichtlich um ein Leaf-Objekt.

```
Sub ADSIRekList(oContainer, ebene)
    Dim obj ' As IADs
    Dim c ' As IADsContainer
    say Space(ebene * 2) & oContainer.Name
    On Error Resume Next
```

Grundlagen der ADSI-Programmierung

```
For Each obj In oContainer
If Err.Number <> 0 Then      ' Leaf-Objekt
    Err.clear
    Exit Sub
Else
    ADSIRekList obj, ebene + 1 ' Container
                                ' Rekursion !
End If
Next
End Sub
```

Listing 13.9: Hilfsroutine: Rekursiver Durchlauf durch einen Verzeichnisdienst-Container [ADSIRekList.wsf]

```
Dim obj ' As IADs
Set obj = GetObject("WinNT://Sonnensystem")
ADSIRekList obj, 0
```

Listing 13.10: Testfunktion für ADSIRekList() [teste_ADSIRekList.wsf]

Collections

Neben den Containern gibt es in ADSI auch noch einige konventionelle Collections, z.B. die Members-Collection innerhalb der Schnittstelle IADsGroup. Eine Collection unterscheidet sich dadurch von einem Container, dass sie nicht über einen ADSI-Pfad, sondern über ein Attribut eines Verzeichnisobjekts angesprochen wird.

Collections

13.5.4 Instanzenverwaltung

Die IADsContainer-Schnittstelle stellt die Methoden zur Verwaltung von Objekten bereit: Create() erzeugt neue Objekte, Delete() löscht bestehende Objekte. In beiden Fällen ist der Klassenname anzugeben.

Create()

```
Set NTDomain = GetObject("WinNT://sonne")
Set newuser = NTDomain.Create("user", "ASchuermann")
newuser.Fullname = "Astrid Schuermann"
newuser.Description = "Korrektorin dieses Buchs!"
newuser.SetInfo
```

Listing 13.11: Erzeugung eines neuen Benutzers im WinNT-Provider

Ein Objekt wird nicht durch einen Methodenaufruf auf sich selbst, sondern über die Ausführung von Delete() auf einem Container-Objekt unter Angabe von Klassennamen und RDN des zu löschenden Objekts entfernt. Der Aufruf von SetInfo() ist nicht nötig.

Delete()

```
Set NTDomain = GetObject("WinNT://Sonne")
NTDomain.Delete "user", "MarkusLeibold"
```

Listing 13.12: Löschen eines Benutzers aus einer WinNT-Domäne



Vor der Ausführung von `Create()` und `Delete()` sollten Sie auf die Existenz eines Objekts prüfen, um einen Laufzeitfehler zu vermeiden.

Rekursives Löschen

Komplexe Löschvorgänge

Üblicherweise können Container-Objekte erst dann gelöscht werden, wenn sie leer sind. Die Routine `ADSI_DeleteThis()` löscht in dem Fall, dass der übergebene ADSI-Pfad einen Container darstellt, zunächst rekursiv alle Unterobjekte. Die Routine ist aber auch eine Erleichterung für Leaf-Objekte, da ein Verzeichnisobjekt direkt über seinen Pfad gelöscht werden kann. Im Gegensatz zur Routine `ADSIRekList()` wird hier der Zugriff auf das Schema dazu verwendet, zwischen Leaf- und Containerobjekten zu unterscheiden.

```
Sub ADSI_DeleteThis (this)
    Dim objthis ' As IADs
    Dim objparent ' As IADsContainer
    Dim objSch ' As IADsClass
    Dim objchild ' As IADs
    ' -- Zugriff auf zu löschendes Objekt
    Set objthis = GetObject(this)
    Set objSch = GetObject(objthis.Schema) ' Schemazugriff
    ' -- Bei Containern Iteration über alle Unterobjekte...
    If objSch.container Then ' nur für Container
        For Each objchild In objthis
            ' -- Unterobjekt löschen
            ADSI_DeleteThis objchild.adspath ' Rekursion
        Next
    End If
    ' -- Löschen dieses Objekts ausführen...
    Set objparent = GetObject(objthis.Parent)
    objparent.Delete objthis.Class, objthis.Name
    say "Gelöscht: " & this
End Sub
```

Listing 13.13: Rekursives Löschen in einem Verzeichnisdienst

Für Container-Objekte wird häufig eine Routine benötigt, um alle Unterobjekte des Containers zu entfernen, den Container selbst aber nicht zu löschen.

```
Sub ADSI_Clear (this)
    Dim objthis ' As IADs
    Dim objchild ' As IADs
    ' -- Zugriff auf zu säubernden Container
    Set objthis = GetObject(this)
    For Each objchild In objthis
        ' -- Unterobjekt löschen
        ADSI_DeleteThis objchild.adspath
    Next
End Sub
```

Listing 13.14: Säubern eines Containers

13.5.5 ADSI-Sicherheit

Sicherheitsfragen spielen beim Zugriff auf Verzeichnisdienste via ADSI eine große Rolle. Es gilt zu verhindern, dass Unbefugte Verzeichnisobjekte lesen oder gar verändern. ADSI bietet zwei Sicherheitsmodi: den Standardmodus und den Impersonifizierungsmodus.

Sicherheitsfragen

Standardsicherheit

Im Standardmodus werden alle ADSI-Operationen unter dem Sicherheitskontext ausgeführt, unter dem der ADSI-Client läuft. Welcher Sicherheitskontext das ist, hängt von der Identitäts-Konfiguration des (Scripting) Hosts ab. In der Regel ist es jedoch der gerade angemeldete Benutzer.

Standardmodus

Impersonifizierungsmodus

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.

Impersonifizierung

Der ADSI-Client verwendet den ADSI-Impersonifizierungsmodus durch die ADSI-Standardschnittstelle `IADsOpenDSObject`, um im laufenden Programm den Benutzerkontext zu wechseln. Diese Schnittstelle wird beim Zugriff auf ein Namespace-Objekt – also auf den ADSI-Pfad ohne Distinguished Name – zurückgegeben. `IADsOpenDSObject` stellt nur eine Methode zur Verfügung: `OpenDSObject()`. Diese Methode erfüllt zunächst einmal die gleiche Funktion wie die VB-Funktion `GetObject()`. Zusätzlich kann ein Benutzerkonto angegeben werden, unter dem die folgenden ADSI-Zugriffe ausgeführt werden. Anzugeben sind ein Benutzername und ein Passwort. Der letzte Parameter `Flags` gibt im Wesentlichen die Authentifizierungsmethode an (vgl. folgende Tabelle). Der Regelfall ist `ADS_SECURE_AUTHENTICATION` (1). Beim Windows NT 4.0 Provider wird dann die *NT LAN Manager (NTLM)*-Authentifizierung benutzt. Bei Windows2000 mit Active Directory wird zunächst versucht, *Kerberos* zu benutzen, bevor auf NTLM zurückgegriffen wird.

`IADsOpenDSObject::OpenDSObject()`

```
Dim MyNamespace ' as IADsOpenDSObject
Set MyNamespace = GetObject("LDAP:") ' Namespace
Set objX = MyNamespace.OpenDSObject(DN, UserName, Password, Flags)
' Impersonifizierung
```

Alle folgenden Operationen auf dem ADSI-Objekt, auf das `objX` verweist, werden dann unter dem angegebenen Benutzerkonto ausgeführt. Der Nachteil dieser Methode besteht darin, dass `OpenDSObject()` das Passwort im Klartext erwartet. Das Kennwort muss also im Klartext im Quellcode abgelegt oder durch den Client selbst verschlüsselt abgelegt und dekodiert werden.

Passwort im Klartext

Tabelle 13.17:
Konstanten für
Sicherheitsflags bei
OpenDSObject().
Diese Konstanten
sind leider nicht in
der Typelibrary
definiert.

Symbolische Konstante	Wert	Erläuterung
ADS_SECURE_AUTHENTICATION	1	Beim Windows NT 4.0 Provider wird dann die NT LAN Manager (NTLM)-Authentifizierung benutzt. Bei Windows2000 mit ADS wird zunächst versucht, Kerberos zu benutzen, bevor auf NTLM zurückgegriffen wird.
ADS_USE_ENCRYPTION	2	Die Übertragung wird mit SSL verschlüsselt, sofern ein Server-Zertifikat vorhanden ist.
ADS_READONLY_SERVER	4	Es wird angezeigt, dass der Server das Ändern von Einträgen nicht unterstützen muss.
ADS_NO_AUTHENTICATION	16	Es wird eine anonyme Verbindung aufgebaut (vom WinNT-Provider nicht unterstützt).
ADS_FAST_BIND	32	Fast-Bind bedeutet, dass nur die Standardschnittstellen des Objekts, nicht aber die zusätzlichen Schnittstellen zur Verfügung stehen. Dieser Modus erhöht die Zugriffsgeschwindigkeit.
ADS_USE_SIGNING	64	Stellt Datenintegrität sicher
ADS_USE_SEALING	128	Verwendet Kerberos zur Verschlüsselung (nur Windows2000)

Benutzerpfade unter NT

Bei OpenDSObject() können Benutzernamen unter WinNT wie folgt angegeben werden:

- BENUTZERNAME
- DOMAIN\BENUTZERNAME
- COMPUTER\BENUTZERNAME

Benutzerpfade im Active Directory

Mit Windows2000 ADS via LDAP gibt es darüber hinaus folgende Möglichkeiten:

- Distinguished Name, z.B. CN=HS,OU=IT-VISIONS,DC=IT-VISIONS,DC=DE
- User Principal Name (UPN), z.B. HS@IT-VISIONS.DE. Dies ist der Name, der sich aus dem Windows2000-Anmeldenamen und der Domain zusammensetzt. Auch wenn der Aufbau daran erinnert, ist dies nicht notwendigerweise die E-Mail-Adresse des Benutzers. Der UPN ist in dem Verzeichnisattribut userPrincipalName gespeichert.

Zugriffsrechte auf Objektebene

Access Control Lists

In Verzeichnisdiensten können üblicherweise Blätter und/oder Knoten durch *Access Control Lists (ACLs)* geschützt werden. ADSI unterstützt diese Funktion durch die Schnittstellen `IADsSecurityDescriptor`, `IADsAccessControlEntry` und `IADsAccessControlList` – allerdings nur für Windows2000. NT4

verfügt ohnehin nicht über eine derart fein granulierte Sicherheit, aber ADSI hat auch Probleme mit den ACLs im Exchange Server. Sie müssen dafür auf die Zusatzkomponente *ADsSecurity* (vgl. Kapitel 13.11) zurückgreifen.

13.5.6 Schemazugriff

Das Schema eines Verzeichnisdienstes kann auch per ADSI ausgelesen werden. Der Aufbau des Schemas ist von Verzeichnisdienst zu Verzeichnisdienst verschieden. So definiert WinNT Klassen durch Instanzen der Klasse *class* und Attribute durch Instanzen der Klasse *property*. Bei Active Directory heißen diese Schemaklassen *classSchema* und *attributeSchema*.

Zugriff auf das transparente Schema

Zum polymorphen Zugriff auf diese unterschiedlichen Klassen definiert ADSI die Standardschnittstellen *IADsClass*, *IADsProperty* und *IADsSyntax*. Sofern ein Verzeichnisdienst Schemamodifikationen zulässt, können diese Schnittstellen auch dazu verwendet werden, um neue Klassen und Attribute zu erstellen.

ADSI-Meta-Schnittstellen

Das nächste Listing zeigt eine Funktion, die für ein beliebiges Objekt ausgibt, welche Attribute es besitzt. Dabei wird zwischen Pflichtattributen und optionalen Attributen unterschieden. Für Container-Objekte wird mit Hilfe des Attributs *Containment* zusätzlich ausgegeben, welche Unterklassen sie enthalten können. Die Funktion *SchemaInfo()* erwartet als Parameter einen gültigen ADSI-Pfad, also auch die Namespace-ID.

SchemaInfo

Die Routine *SchemaInfo()* gibt zu Attributen auch die Werte aus. Bitte beachten Sie jedoch die notwendige Fehlertoleranz mit *On Error Resume Next*: Einige Attribute lassen sich nicht auslesen, es kommt beim Zugriff mit *Get()* zu einer Fehlermeldung. Auch ist für einen Sonderfall im Active Directory eine besondere Variante der *Flat()*-Funktion notwendig, da es im Active Directory mehrwertige Attribute gibt, deren einzelne Werte wieder Objekte sind, die GUIDs repräsentieren. *AdsFlat()* wiederum benötigt eine Umwandlungsroutine für eine GUID aus einem binären Format in einen String (Routine *binGUIDtoSTR()*).

Notwendige Fehlertoleranz

```
Function SchemaInfo(adsipfad)
Dim obj ' As IADs
Dim Cl ' As IADs
Dim Name ' As Variant
Dim Wert ' As Variant
' -- Bindung an Verzeichnisobjekt
Set obj = GetObject(adsipfad)
' -- Zugriff auf Schemaobjekt
Set Cl = GetObject(obj.Schema)
say "Schema-Informationen für: " & adsipfad
say "-----"
say "Klasse: " & Cl.Name
say "GUID: " & Cl.guid
say "-----"
```

Active Directory Service Interface (ADSI)

```
' ----- Unterklassen und Unterobjekte
If Cl.container Then
    say "--- Unterobjekte:"
    For Each Name In Cl.Containment
        say Name
    Next
Else
    say "--- Klasse ist ein Leaf-Objekt."
End If
' ----- Pflicht-Attribute
say "--- Pflichtattribute:"
For Each Name In Cl.MandatoryProperties
    Wert = ""
    On Error Resume Next
    Wert = obj.Get(Name)
    On Error GoTo 0
    Wert = ADSflat(Wert)
    If Wert <> "" Then
        say Name & " = " & Wert
    Else
        say Name
    End If
Next

Dim oBinVal 'As ActiveDs.DNWithBinary
' ----- Optionale Attribute
say "--- Optionale Attribute:"
For Each Name In Cl.OptionalProperties
    Wert = ""
    On Error Resume Next
    Wert = obj.Get(Name)
    On Error GoTo 0
    Wert = ADSflat(Wert)
    If Wert <> "" Then
        say Name & " = " & Wert
    Else
        say Name
    End If
Next
End Function

Listing 13.15: Zeigt die Schemainformationen zu einem Verzeichnisobjekt an [Schema-Info.wsfl]

Function ADSflat(var)
'On Error Resume Next
Dim i 'As Integer
' -- Mehrfachattribut?
```

Grundlagen der ADSI-Programmierung

```
If IsArray(var) Then ' Array flachklopfen
    ADSflat = ""
    For i = LBound(var) To UBound(var)
        If ADSflat <> "" Then ADSflat = ADSflat & ";"
        ' -- Prüfen auf ActiveDs.DNWithBinary
        If IsObject(var(i)) Then
            ADSflat = ADSflat & var(i).DNString & " == " & _
            binGUIDtoSTR(var(i).BinaryValue)
        Else
            ADSflat = ADSflat & var(i)
        End If
    Next
Else ' War kein Array
    ADSflat = var
End If
End Function
```

Listing 13.16: Besondere Variante von flat()

```
Function binGUIDtoSTR(guid)
    Dim i ' As Integer
    For i = (LBound(guid) + 1) To (UBound(guid) + 1)
        binGUIDtoSTR = binGUIDtoSTR & Hex(AscB(MidB(guid, i, 1)) \ 16) _
        & Hex(AscB(MidB(guid, i, 1)) Mod 16)
    Next
End Function
```

Listing 13.17: Umwandlung einer binären GUID in einen String

```
SchemaInfo "WinNT://mars"
SchemaInfo "WinNT://mars/w3svc"
SchemaInfo "LDAP://sonne2000/OU=IT-Visions,dc=it-visions,dc=de"
SchemaInfo "LDAP://sonne2000/dc=it-visions,dc=de"
SchemaInfo "LDAP://<WKGUID=a9d1ca15768811d1aded00c04fd8d5cd,dc=it-visions,dc=de>"
```

Listing 13.18: Test der Routine SchemaInfo() [teste_SchemaInfo.wsfl]

Die Buch-CD [CD:/install/tools/adsi/schemabrowser/] enthält auch das in Visual Basic 6.0 geschriebene Tool *ADSI Schema Browser* (SCHEMABROWSER.EXE). Der Quellcode wird an dieser Stelle nicht erläutert, weil er im Kern der Routine *SchemaInfo()* entspricht und darüber hinaus im Wesentlichen nur Routinen für die Steuerung des GUIs enthält.



13.5.7 ADSI-Queries

OLE DB-Provider für Active Directory

ADSI unterstützt neben der schrittweisen Iteration durch Container auch den Mengenzugriff auf Verzeichniseinträge über OLE DB. Der *OLE DB-Provider für den Active Directory Service* gestattet es, per ActiveX Data Objects (ADO) auf Verzeichnisinformationen zuzugreifen. Der OLE DB-Provider wandelt dabei das hierarchische Objektmodell eines Verzeichnisdienstes in eine flache relationale Tabellenstruktur um. Diese Zugriffsart hat zwei Vorteile:

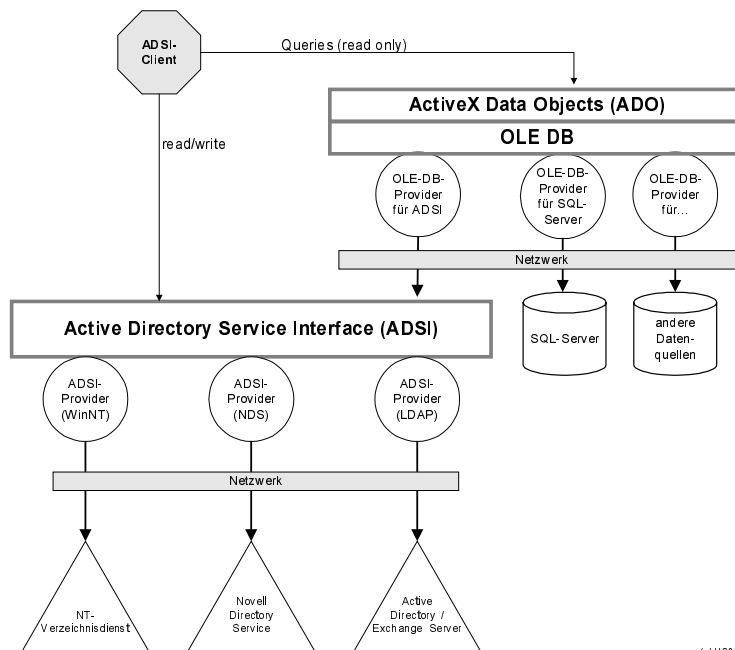
- Die Menge der Verzeichniseinträge kann bereits auf der Serverseite eingeschränkt werden.
- Die Abfrage kann Einträge aus mehreren Ebenen im Verzeichnisbaum zurückgeben.



Da ein Verzeichnisdienst eine semi-strukturierte Datenbank ist, in der nicht alle Objekte die gleichen Attribute besitzen, bleiben bei der Umwandlung in eine relationale Tabelle zwangsläufig einige Felder der Tabelle leer.

Dieser OLE DB-Provider basiert auf der ADSI-Komponente, ohne die er nicht verwendbar ist. Der Provider wird durch das ADSI-Setup installiert, nicht durch das MDAC-Setup.

Abbildung 13.15:
Architektur für
den Zugriff auf
Verzeichnisdienste
via ADO



(c) H500

Der Provider unterliegt derzeit folgenden Beschränkungen:

Einschränkungen

- ▶ Es ist nur ein Lesezugriff möglich. Es hieß zwar aus Microsoft-Kreisen, mit Windows2000 würde ein Schreibzugriff möglich, diese Funktionalität ist jedoch in Windows2000 noch nicht verfügbar.
- ▶ Es können derzeit nur LDAP-fähige Verzeichnisdienste abgefragt werden. Zur Unterstützung von OLE DB muss ein ADSI-Provider die IDirectorySearch-Schnittstelle unterstützen. Das leistet derzeit nur der LDAP-Provider.

Nicht alle Versionen von ADSI und ADO sind kompatibel miteinander. Verwenden Sie bei ADSI 2.0 auch Version 2.0 von ADO. ADSI 2.5 benötigt mindestens ADO 2.1!

Syntax für Suchanfragen

Der Ablauf ist wie bei ADO üblich: Der Benutzer führt eine Abfrage aus und erhält ein Recordset-Objekt zurück. ADSI-Queries auf LDAP-Verzeichnissen sind in zwei Syntaxformen möglich: in der LDAP-Query-Syntax nach [RFC1960] und [RFC2254] und in einer SQL-ähnlichen Syntax.



Syntax

LDAP-Query-Syntax

Die allgemeine Vorschrift lautet:

LDAP-Syntax

```
"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.
Beispiel: LDAP://sonne2000/dc=It-Visions,dc=DE
- ▶ **Filter** eine Bedingung in umgekehrt polnischer Notation (UPN oder Postfix-Notation). Diese Notation zeichnet sich dadurch aus, dass die Operatoren am Anfang stehen.
Beispiel: (&(objectclass=user)(name=h*))
- ▶ **Attribute** eine durch Komma getrennte Attributliste der gewünschten Verzeichnisattribute, die in die Tabelle aufgenommen werden sollen. Diese Angabe ist nicht optional, der Sternoperator (»*«) wie bei SQL ist nicht erlaubt.
Beispiel: AdsPath,Name,SamAccountname
- ▶ **Scope** eine der in der folgenden Tabelle genannten Konstanten

Tabelle 13.18:
Suchtiefen bei
ADSI-Queries

Konstante (LDAP-Syntax)	Konstante (ADO)	Wert	Erläuterung
BASE	ADS_SCOPE_BASE	0	Es wird nur der angegebene Eintrag gesucht. Die Ergebnismenge umfasst keinen oder einen Datensatz.
ONELEVEL	ADS_SCOPE_ONELEVEL	1	Es wird in den Einträgen gesucht, die dem angegebenen Eintrag untergeordnet sind.
SUBTREE	ADS_SCOPE_SUBTREE	2	Es werden alle darunter liegenden Ebenen durchsucht.

SQL-Query-Syntax**SQL-Syntax** Zunächst die Vorschrift:

```
"SELECT Attribute FROM 'Start' WHERE Bedingung order by
SortierAttribute"
```

Dabei ist:

- ▶ Attribute eine durch Komma getrennte Attributliste
- ▶ Start ein LDAP-Pfad inkl. LDAP:// in einfachen Anführungszeichen
- ▶ Bedingung eine Bedingung in SQL-Syntax
- ▶ SortierAttribute eine durch Komma getrennte Attributliste



Die Syntax entspricht also der SQL-Syntax mit der Ausnahme, dass an die Stelle des Tabellennamens der LDAP-Pfad rückt. Der Vorteil der LDAP-Syntax liegt in der Möglichkeit, die Suchtiefe zu definieren; der Vorteil der SQL-Syntax ist die Sortiermöglichkeit. Der Nachteil der SQL-Syntax kann allerdings durch die Übergabe eines Werts an das durch den OLE DB-Provider für die Command-Klasse definierte Attribut »search scope« ausgeglichen werden.

```
objCMD.Properties("searchscope") = ADS_SCOPE_SUBTREE
```

Tabelle 13.19:
Beispiele für ADSI-
Queries

	LDAP-Syntax	SQL-Syntax
Exchange Server: Alle Postfächer unterhalb des Empfänger-Containers	<LDAP://ldapsrvr/ O=ITV/ OU=ESSEN/ CN=recipients>; (objectClass= organizationalperson); AdsPath,Name;subtree")	Select AdsPath, Name FROM 'LDAP://ldapsrvr/ O=ITV/OU=ESSEN/ CN=recipients' where objectClass= 'organizationalperson'

	LDAP-Syntax	SQL-Syntax
Active Directory: Alle Benutzer, deren Name mit h beginnt	<LDAP://sonne2000/ dc=It-Visions,dc=DE>; ((&(objectclass=user) (name=h*)); adspath, SamAccountname; subtree	Select adspath, SamAccountname from 'LDAP://sonne2000/ dc=It-Visions, dc=DE' where objectclass='user' and name = 'h*'

Tabelle 13.19:
Beispiele für ADSI-
Queries
(Forts.)

Eine Abfrage, die nur aus der Bedingung `class=*` besteht, funktioniert nicht. Um alle Verzeichnisobjekte zurückzuliefern, muss der Sternoperator auf ein anderes Attribut angewendet werden.



Beispiel

Das folgende Listing zeigt die Hilfsroutine `adoads_i_execADOquery` (`strLDAPQuery`), die eine beliebige ADO/ADSI-Query ausführt. Auffällig ist, dass der Connection String nur aus der Angabe des Providers besteht. Im Gegensatz zu anderen Datenverbindungen ist hier kein Servername anzugeben. Der Servername wird erst im Rahmen der ADSI-Query mitgeteilt, die hier in LDAP-Syntax angegeben ist. Nach dem Ausführen der Abfrage mit `Execute()` steht ein `Recordset`-Objekt zur Verfügung.

Beispiel

```
' ### Ausführung einer ADSI/ADO-Query
Sub adoads_i_execADOquery(strLDAPQuery)
Dim objCon ' As ADODB.Connection
' -- ADO-Connection-Objekt erzeugen
Set objCon = CreateObject("ADODB.Connection")
' -- Verbindung zu dem OLE DB-Provider für ADSI
CONSTING = "Provider=ADSDSOObject"
objCon.Open CONSTING ' , "administrator", ""
' -- Selektion aller User, deren Verzeichnisname mit H beginnt
' --- Ausführen der Abfrage
say "Starting Query..." & strLDAPQuery
Set rs = objCon.Execute(CStr(strLDAPQuery))
' --- Ausgabe der Ergebnisse
list_table rs
' -- Schließen der Objekte
rs.Close
objCon.Close
End Sub
```

Listing 13.19: Anfrage an ein Active Directory via ADO

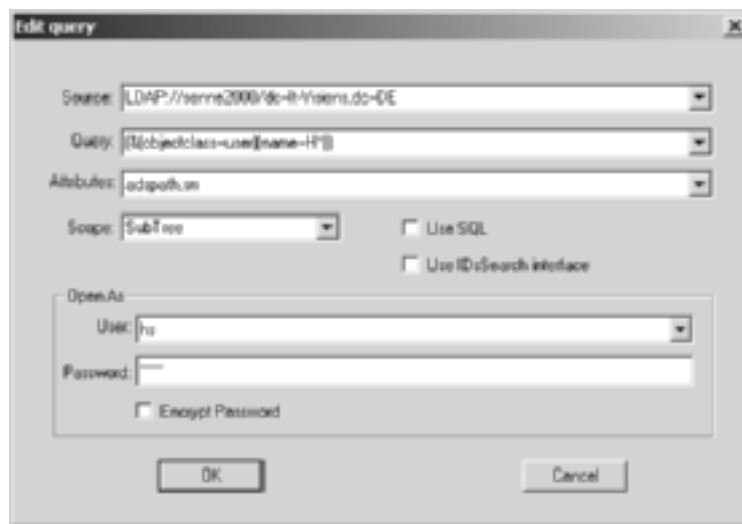
Das folgende Skript testet die Hilfsroutine mit Active Directory und Exchange Server.

Active Directory Service Interface (ADSI)

```
' - Beispiel Active Directory
adoadsi_execADOquery "<LDAP://sonne2000/dc=it-visions,dc=de>;
(&(objectclass=user)(cn=H*));cn,givenname,sn,samaccountname,adspath;
subtree"
' - Beispiel Exchange 5.5
adoadsi_execADOquery "<LDAP://Sonne/cn=recipients,ou=Sonnensystem,
o=Universum>;(&(objectclass=organizationalperson)(cn=H*))>;adspath,cn,
uid,mail; subtree"
```

Listing 13.20: Testet Anfrage an ein Active Directory bzw. an Exchange via ADO [adoadsi_test.wsf]

Abbildung 13.16:
ADSI-Abfragen
können auch im
Active Directory
Browser ausgeführt
werden.



13.6 ADS Namespaces Container

Namespaces Der *ADS Namespaces Container* steht über allen ADSI-Providern und ist daher im ADSI-Kern implementiert (siehe ADSI-Meta-Objektmodell, Kapitel 13.3) Der *Namespaces Container* wird – wie fast alle anderen Objekte in ADSI auch – als eine persistente Instanz angesehen. Daher erfolgt die Aktivierung mit `GetObject()`, nicht mit `CreateObject()`.

```
Set Obj = GetObject("Ads:")
```



Auch wenn die Microsoft-Dokumentation in diesem Zusammenhang immer vom Active Directory spricht und den Moniker »ADS« verwendet, ist dieser Container auf allen Clients verfügbar, die ADSI installiert haben.

ADS Namespaces Container

Der Namespaces-Container enthält Namespace-Objekte für die verfügbaren Namespaces. Jedes Namespace-Objekt enthält die Root-Objekte aller erreichbaren Verzeichnisbäume (also z.B. NT-Domänen oder IIS-Server). Namespace-Objekte unterstützen neben den Schnittstellen IADs und IADsContainer auch die Schnittstelle IADsOpenDSObject, die der Impersonifizierung (vgl. Kapitel 13.5) dient. Die Namespace-Objekte sind in der Registry eingetragen: Sie finden im COM-Viewer z.B. »WinNT Namespace Object« und »LDAP Namespace Object«.

Namespace-Objekte

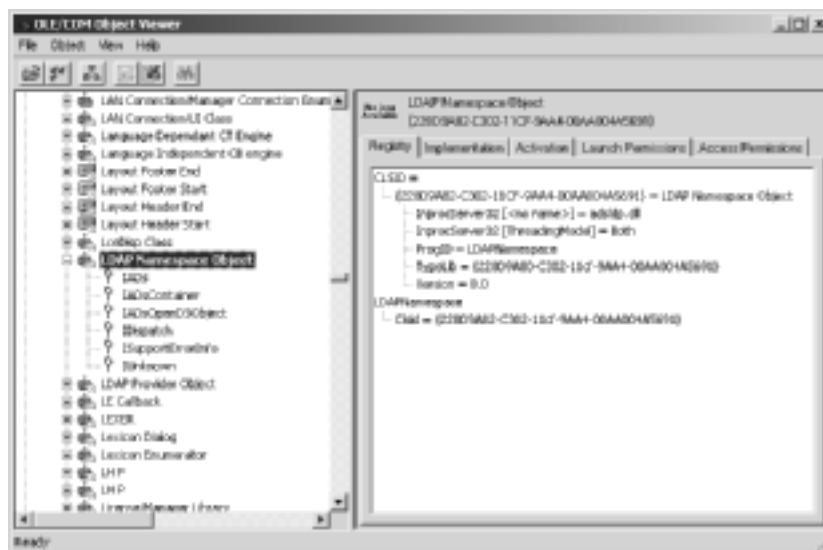


Abbildung 13.17:
Das Namespace-Objekt für LDAP im COM-Viewer

```
' == Liste der Namespaces
Sub ADSI_Namespaces()
Dim objNSCont ' as IADsNamespaces
Dim objNS ' as IADs
Dim objRoot ' as IADs
Set objNSCont = GetObject("ADS:")
' --- Schleife für die Namespaces
say "verfügbare Namespaces:"
For Each objNS In objNSCont
say "-" & objNS.Name
' --- Liste der Wurzelcontainer
For Each objRoot In objNS
On Error Resume Next
say " - " & objRoot.Name
Next
Next
End Sub
```

Listing 13.21: Liste der Namespaces mit den Wurzeln der Verzeichnisbäume
[ADSI_Namespaces.wsf]

Active Directory Service Interface (ADSI)

Das Skript liefert in der Testumgebung folgende Ausgabe:

Verfügbare Namespaces:

- WinNT:
 - ARBEITSGRUPPE
 - SONNENSYSTEM
- NWCOMPAT:
- NDS:
- LDAP:
 - DC=IT-Visions
- IIS:
 - MARS
 - SONNE2000

Listing 13.22: Ausgabe des Skript ADSI_Namespaces.wsf

Der WinNT-Namespace enthält die verfügbaren Domänen und Arbeitsgruppen, der IIS-Namespace die verfügbaren Webserver, LDAP die Domain des Active Directory. Ein Novell-Verzeichnisdienst wurde nicht gefunden.



Das Abschalten der Fehlerüberprüfung mit `On Error Resume Next` in Listing 5.58 ist notwendig, da ein leerer Namespace beim `For...Each`-Zugriff einen Fehler liefert. Ein `Count`-Attribut, mit dem vorher geprüft werden könnte, ob es Inhalte gibt, steht auf einem Namespace-Objekt nicht zur Verfügung.

13.7 ADSI-Provider für Windows NT 4.0 (WinNT)

NT4-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.

Features

Mit dem WinNT-Provider können folgende Objekte im NT-Verzeichnis verwaltet werden: 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 und Dateien, die in Benutzung sind.

Der WinNT-Provider dient auch dem Zugriff auf die lokalen Benutzer einer Windows2000 Professional Workstation bzw. eines Windows2000 Servers ohne Active Directory. Das Schema des WinNT-Verzeichnisdienstes wurde dabei nicht geändert. Der WinNT-Provider kann auch auf das Active Directory zugreifen.



13.7.1 WinNT-Objektmodell

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.

Wo kommt das Schema her?

Domain-Objekte enthalten Computer-, User- und Group-Objekte. Ein Computer-Objekt enthält User-, Group-, Service-, Fileservice-, 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.

Objekthierarchie

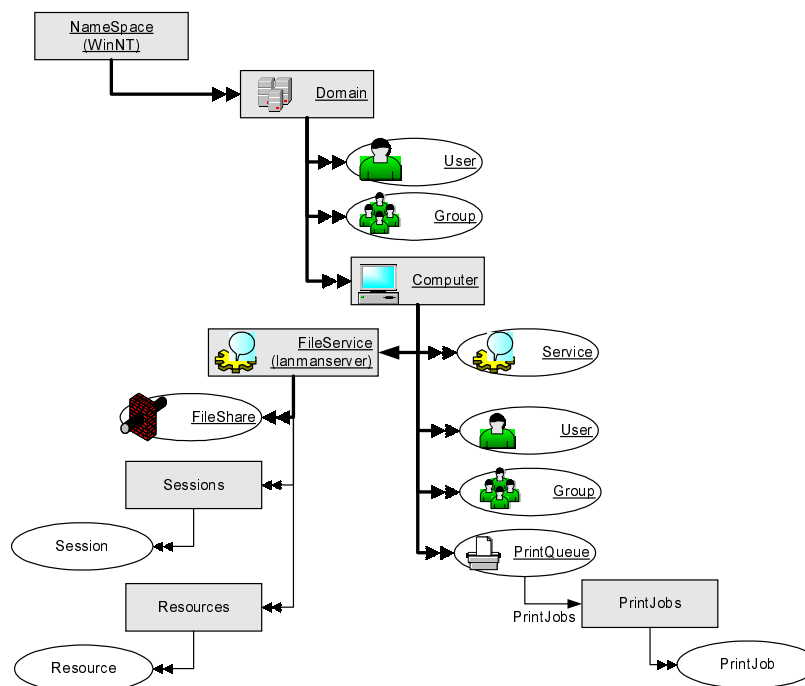


Abbildung 13.18:
Objekte im
WinNT-Namespace

Active Directory Service Interface (ADSI)

Service-Objekte sind in den meisten Fällen Leaf-Objekte, ein Fileservice ist jedoch auch selbst ein Container.

Klassen und Schnittstellen

Implementierte Standardschnittstellen

Pflichtgemäß implementieren alle Leaf-Klassen die Standardschnittstelle IADs und alle Container-Objekte die Standardschnittstellen IADs und IADsContainer. Darüber hinaus implementiert der WinNT-Provider für alle Verzeichnisobjekte weitere ADSI-Standardschnittstellen, die den Zugriff komfortabler machen und zusätzliche Methoden anbieten.

Tabelle 13.20:
Weitere Schnittstellen der Objekte im WinNT-Provider

Klasse	Typ	Implementierte Zusatzschnittstellen
Domain	Container	IADsDomain
Computer	Container	IADsComputer IADsComputerOperations
File Service	Container	IADsFileservice IADsFileserviceOperations
Service	Leaf	IADsService IADsServiceOperations
User	Leaf	IADsUser
Group	Leaf	IADsGroup
PrintQueue	Leaf	IADsPrintQueue IADsPrintQueueOperations
PrintJob	Leaf	IADsPrintJob IADsPrintJobOperations
Service	Leaf	IADsService IADsServiceOperations
FileShare	Leaf	IADsFileShare
Resource	Leaf	IADsResource
Session	Leaf	IADsSession

13.7.2 Identifikation und Bindung

Objektidentifikation

Der WinNT-Provider benutzt die Namespace-ID »WinNT«, wobei Groß- und Kleinschreibung exakt zu beachten sind. Der providerspezifische Teil entspricht der Form /obersteEbene/ebene1/ebene2.

Tabelle 13.21:
ADSI-Pfade im WinNT-Provider

WinNT-Objektyp	ADSI-Pfad
WinNT-Root	WinNT://
Domäne	WinNT://DOMAINNAME
Computer	WinNT://DOMAINNAME/COMPUTERNAME ODER WinNT://COMPUTERNAME

WinNT-Objekttyp	ADSI-Pfad
Domänen-Benutzer (entsprechend für Gruppen)	WINNT://DOMAINNAME/BENUTZERNAME ODER WINNT://PDCNAME/BENUTZERNAME
Domänen-Benutzer (entsprechend für Gruppen)	WINNT://COMPUTERNAME/BENUTZERNAME
NT-Dienste	WINNT://COMPUTERNAME/DIENSTNAME
Druckerwarteschlangen	WINNT://COMPUTERNAME/DRUCKERNAME
Verzeichnisfreigabe	WINNT://COMPUTERNAME/LANMANSERVER/FREIGABE NAME
WinNT-Schema	WINNT://COMPUTERNAME/SCHEMA ODER WINNT://DOMAINNAME/SCHEMA

Tabelle 13.21:
ADSI-Pfade im
WinNT-Provider
(Forts.)

Viele Aktionen, die auf einem Domain-Objekt möglich sind, sind auch auf einem Computer-Objekt möglich. In Skripten, die keine Typdeklaration verwenden, müssen Sie nur den ADSI-Pfad ändern. Haben Sie in einer typisierten Umgebung jedoch die Objektvariable auf `IADsDomain` deklariert, müssen Sie den Typ anpassen, da Sie sonst die Meldung »Typen unverträglich« erhalten.



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

Angabe des
Klassennamens

Die Angabe des Klassennamens erhöht außerdem die Geschwindigkeit, mit der ein Objekt gefunden wird, zum Teil erheblich. Dies ist besonders bei Domänen und Computern wichtig, da der WinNT-Provider hier stets darauf angewiesen ist, die Objekte im Netz zu lokalisieren.



13.7.3 Verwaltung von NT-Domänen

Die Domain-Klasse repräsentiert eine Windows NT-Domain. Ein Domain-Objekt kann nicht via ADSI erzeugt werden (um eine Domain zu erzeugen, muss ein Rechner mit Windows NT Server und der Option Domain Controller installiert werden!). ADSI kann Attribute einer Domain auslesen und verändern.

Domain-Klasse

Tabelle 13.22:
Attribute von
IADSDomain

Attribut	Erläuterung
MinPasswordLength	Mindestanzahl der Stellen in einem Passwort
MinPasswordAge	Zeitraum, nach dem das Passwort frühestens geändert werden kann (0 = sofortige Änderung erlaubt). Das Attribut erwartet den Wert in Sekunden. Die Tage müssen also mit 86400 multipliziert werden!
MaxPasswordAge	Anzahl der Sekunden, nach denen das Passwort spätestens geändert werden muss (-1 = Kennwort läuft nie ab)
MaxBadPasswords Allowed	Anzahl der fehlerhaften Kennworteingaben, nach denen eine Sperre erfolgen soll
PasswordHistory Length	Anzahl der Kennwörter, die aufbewahrt werden sollen, um zu verhindern, dass der Benutzer bereits verwendete Passwörter erneut benutzt. Das Maximum ist 24.
AutoUnlockInterval	Dauer der Sperre in Sekunden (-1 bedeutet, dass das Konto so lange gesperrt bleibt, bis ein Administrator es wieder aktiviert). Der Benutzermanager zeigt hier nur Minutenwerte an. Die Anzeige wird ggf. auf volle Minuten aufgerundet.
LockoutObservation Interval	Anzahl der Sekunden, nach denen ein Konto zurückgesetzt wird. Mit diesem Attribut kann bestimmt werden, wie groß die Zeitspanne zwischen zwei Fehlversuchen sein kann, damit diese als zusammenhängend gesehen werden. Der Benutzermanager zeigt hier nur Minutenwerte an. Die Anzeige wird ggf. auf volle Minuten aufgerundet.

Kontenrichtlinien ändern

Richtlinien Das folgende Skript setzt die Kontenrichtlinien einer Domain so, dass die Benutzer ihre Passwörter jeweils zwischen dem 30. und dem 60. Tag wechseln müssen.

```
Set oDomain = GetObject("WinNT://Sonnensystem")
' -- Gewünschte Zeitspannen
iMinTage = 30 ' Tage
iMaxTage = 60 ' Tage
iUnlock = 100 ' Minuten
Observation = 50 ' Minuten
' -- Umrechnung auf Sekunden
iMinTage = iMinTage * 60 * 60 * 24
iMaxTage = iMaxTage * 60 * 60 * 24
iUnlock = 100 * 60
Observation = 50 * 60
' -- Wertzuweisung
```

ADSI-Provider für Windows NT 4.0 (WinNT)

```
oDomain.MinPasswordAge = iMinTage
oDomain.MaxPasswordAge = iMaxTage
oDomain.AutoUnlockInterval = iUnlock
oDomain.LockoutObservationInterval = Observation
' Schreiben des Caches
oDomain.SetInfo
say "Kontorichtlinien geändert!"
```

*Listing 13.23: Zugriff auf Kontorichtlinien via Domain-Objekt
[ADSINT_Kontorichtlinien.wsf]*

Auflisten aller Domänen

Es ist möglich, alle Domänen eines NT-Netzwerks aufzulisten. Dazu ist ein Zugriff auf das WinNT-Namespace-Objekt nötig.

```
Dim oDS ' As ActiveDs.IADsOpenDSObject
' -- Zugriff auf Namespace
Set oDS = GetObject("WinNT:")
' -- Liste ausgeben
For Each oDomain In oDS
    say oDomain.Name
Next
```

Listing 13.24: Liste aller erreichbaren Windows-Domänen [ADSINT_AlleDomaenen.wsf]

Auflistung aller Unterobjekte einer Domäne

Das folgende Skript listet alle Unterobjekte einer Domain mit ihrem Klassennamen auf.

```
' -- Zugriff auf Domain-Objekt
Set oDomain = GetObject("WinNT://Sonnensystem")
' -- kein Filter, alle auflisten
For Each obj In oDomain
    say obj.Name & " (" & obj.Class & ")"
Next
```

*Listing 13.25: Liste aller Unterobjekte eines Domain-Objekts mit den Klassennamen
[ADSINT_AlleDomainUnterobjekte.wsf]*

13.7.4 Die WinNT-Klasse »Computer«

Ein Computer-Objekt implementiert unter WinNT zwar neben der Schnittstelle IADsComputer auch die Schnittstelle IADsComputerOperations, die darin definierten Methoden ShutDown() und Status() jedoch nicht.

Computer-Klasse

Liste aller Computer in einer Domain

Die Auflistung der Computer in einer Domain ist über den Zugriff auf das Domain-Objekt mit dem Filter auf die Computer-Klasse möglich. Viele Attri-

Alle Computer

bute von IADsComputer (z.B. Model, Site) sind unter dem WinNT-Provider nicht verfügbar.

```
' -- Zugriff auf Domain-Objekt
Set oDomain = GetObject("WinNT://Sonnensystem")
' -- Filter auf Computer
oDomain.Filter = Array("computer")
' -- Auflisten
For Each oComputer In oDomain
    say oComputer.Name & _
        " mit " & oComputer.OperatingSystem & _
        " " & oComputer.OperatingSystemVersion
Next
```

Listing 13.26: Liste der Computer in einer Domäne [ADSINT_Computerliste.wsf]

13.7.5 Benutzerkonten verwalten mit der »User«-Klasse

Benutzerkonten

Das User-Objekt repräsentiert ein Windows-NT-Benutzerkonto in einer Domäne oder ein lokales Benutzerkonto auf einem einzelnen NT-Computer (Server oder Workstation).



Sie können mit ADSI keine Benutzer auf einem Windows 95/98/ME-System verwalten, die innerhalb der Systemsteuerung eingerichtet werden.

Tabelle 13.23:
Attribute der
WinNT-Klasse
»User«

Attribut	Erläuterung
AccountDisabled	Der Wert <i>True</i> bedeutet, dass das Konto deaktiviert ist. Dieses Attribut wird durch ADSI bereitgestellt und besitzt keine Entsprechung im WinNT-Verzeichnisdienst.
AccountExpirationDate	Datum und Uhrzeit, zu der das Benutzerkonto ungültig wird
BadPasswordAttempts	Anzahl der Fehlversuche bei der Passworteingabe
Description	Textliche Beschreibung des Benutzers
FullName	Vollständiger Name
HomeDirDrive	Laufwerksbuchstabe für das Homeverzeichnis
HomeDirectory	Homeverzeichnis
LastLogin	Datum und Uhrzeit der letzten Anmeldung
LastLogoff	Datum und Uhrzeit der letzten Abmeldung
LoginHours	Stunden, zu denen sich der Benutzer anmelden darf
LoginScript	Name des Login-Skripts

Attribut	Erläuterung
LoginWorkstation	Arbeitsstationen, an denen sich dieser Benutzer anmelden darf
MaxLogins	maximale Anzahl der gleichzeitigen Anmeldungen
MaxPasswordLength	maximale Länge des Passworts
MaxStorage	Speicher, den der Benutzer maximal belegen darf. Diese Funktion ist unter NT 4.0 noch nicht verfügbar. Das Attribut steht daher immer auf unendlich (Wert -1).
MinPasswordLength	Mindestlänge für das Passwort
ObjectSID	Security Identifier des Benutzerkontos
Parameters	String, in dem Anwendungen Daten zum Benutzer abspeichern können. Hier wird z.B. die Callback-Nummer für den RAS-Zugang abgelegt.
PasswordAge	Alter des aktuellen Passworts
PasswordExpired	<i>True</i> bedeutet, dass das Passwort abgelaufen ist.
Profile	Profilverzeichnis
UserFlags	Hier können die aus dem Benutzermanager bekannten Einstellungen KENNWORT LÄUFT NIE AB (Flagwert: &H10000) und BENUTZER KANN PASSWORT NICHT ÄNDERN (Hexadezimalwert &H40) gesetzt werden. Die beiden Einstellungen können durch ein bitweises ODER miteinander verknüpft werden.

Tabelle 13.23:
Attribute der
WinNT-Klasse
»User«
(Forts.)

Methode	Erläuterung
ChangePassword()	Setzen des Passworts unter Angabe des alten Passworts
SetPassword()	Setzen des Passworts ohne Angabe des alten Passworts

Tabelle 13.24:
Auf der WinNT-
Klasse User ausführ-
bare Methoden

Liste der Benutzer einer Domäne

Das nachfolgende Beispiel listet alle Benutzer einer Domain auf. Dabei werden auch der vollständige Name und die Beschreibung ausgegeben.

```
Set oDomain = GetObject("WinNT://Sonnensystem")
oDomain.Filter = Array("user")
say "-- Benutzerliste:"
For Each oUser In oDomain
    say oUser.Name & ", " & oUser.FullName & ", " & _
        oUser.Description
Next
```

Listing 13.27: Liste aller Benutzer einer NT-Domäne [ADSINT_BenutzerListe.wsf]

Anlegen eines neuen Benutzerkontos

Benutzer anlegen Vor dem Anlegen eines neuen NT-Benutzerkontos muss zunächst die Bindung an den übergeordneten Domain- oder an einen Computer-Container hergestellt werden. Bei der Methode `Create()` sind der Klassenname `user` und als zweiter Parameter der gewünschte Benutzername anzugeben. Der Benutzername ist der Relative Distinguished Name eines NT-Benutzers. Erst mit dem Aufruf von `SetInfo()` wird der Benutzer tatsächlich angelegt. Die `User`-Klasse verlangt keine Pflichtattribute.

```
' -- Zugriff auf Domain-Objekt
Set oDomain = GetObject("WinNT://Sonnensystem")
' -- Benutzer anlegen
Set oUser = oDomain.Create("user", "FoxMulder")
' -- Setzen von Eigenschaften
oUser.FullName = "Fox Mulder (FBI-Agent)"
oUser.Description = "Nur ein Test-User"
oUser.HomeDirectory = "e:\homes\username"
oUser.AccountExpirationDate = Now()
oUser.LoginScript = "test.bat"
' -- Cache schreiben
oUser.SetInfo
say "Benutzer angelegt!"
```

Listing 13.28: Hinzufügen eines neuen Benutzers [ADSINT_NeuerBenutzer.wsfl]

An dieser Stelle zeigt sich eine Tücke von ADSI, die auch in anderen Verzeichnisdiensten auftritt. Per Programmcode lassen sich Konfigurationen erzeugen, die von den grafischen Benutzeroberflächen nicht behandelt werden können. Mit ADSI ist es möglich, ein NT-Benutzerkonto mit einer Länge von mehr als 20 Zeichen anzulegen. Der Verzeichnisdienst lässt das offensichtlich zu, der NT-Benutzermanager verweigert aber jeglichen Zugriff auf das Konto. Um solche Probleme wieder zu bereinigen, bedarf es eines weiteren ADSI-Programms.

Die Zeile `oUser.AccountExpirationDate = Now()` bewirkt, dass der Benutzer sein Passwort bei der nächsten Anmeldung ändern muss.

Ändern des Passworts

Das Passwort für einen Benutzer kann erst gesetzt werden, nachdem das Anlegen mit `SetInfo()` vollzogen wurde. Grundsätzlich gibt es zwei Möglichkeiten, ein Passwort mit ADSI zu setzen:

ChangePassword()

- Mit der Methode `ChangePassword()` unter Angabe des bisherigen Passworts

```
Set oU = GetObject("WinNT://Sonnensystem/BirgitKilian")
oU.ChangePassword "alteskennwort", "neueskennwort"
```

- Bei `SetPassword()` ist die Angabe des bisherigen Passworts nicht nötig.

SetPassword()

```
Set oU = GetObject("WinNT://Sonnensystem/BirgitKilian")
oU.SetPassword "neueskennwort"
```

`ChangePassword()` sollte angewendet werden, wenn sichergestellt werden soll, dass nur der betreffende Benutzer selbst das Passwort ändert. Die Methode lässt sich nur ausführen, wenn die Kontorichtlinien dies erlauben (wenn Sie das Skript ausgeführt haben, das die minimale Passwortdauer auf 10 Tage setzt, dann kann `ChangePassword()` erst nach 10 Tagen zum ersten Mal ausgeführt werden!).

`SetPassword()` und `ChangePassword()` können nur auf Benutzerobjekte angewendet werden, die bereits persistent im Verzeichnis existieren. Nach dem Neuanlegen eines Benutzerkontos mit `Create()` muss also erst der Cache mit `SetInfo()` in das Verzeichnis geschrieben werden. Erst dann können `SetPassword()` und `ChangePassword()` verwendet werden. Die Methoden machen die Änderungen automatisch persistent, so dass für die Kennwortänderung kein `SetInfo()` nötig ist.



Umbenennen eines Benutzers

Der WinNT-Verzeichnisdienst erlaubt die Umbenennung eines Benutzerkontos nach dem Anlegen, da für die eindeutige Identifizierung nicht der Kontoname, sondern der *Security Identifier (SID)* des Kontos maßgeblich ist. Das Konto verliert also nicht seine Gruppenzuordnungen oder Rechte. Das folgende Skript gibt zur Kontrolle den SID und die GUID des Kontos vor und nach der Umbenennung aus. Der SID ist eine Zahlenreihe, die als Array zurückgeliefert wird und auf die vor der Ausgabe `flat()` angewandt werden sollte.

Namensänderung

Die Methode zur Umbenennung heißt in ADSI `MoveHere()` und ist eine der Methoden der Standardschnittstelle `IADsContainer`. Es ist nicht möglich, ein Benutzerkonto im WinNT-Verzeichnisdienst zu verschieben, da es nur einen Container für Benutzer geben kann. Eine Verschiebung zwischen Domänen ist ebenfalls nicht möglich.

Verschieben nicht möglich

```
Const strDomain = "Sonnensystem"
Const strAlterName = "FoxMulder"
Const strNeuerName = "DanaScully"
' -- SID und GUID vorher
Set oUser = GetObject("WinNT://" & strDomain & "/" & strAlterName)
say flat(oUser.Get("objectSID")) & "," & oUser.Guid
' -- Zugriff auf Domain !
Set oDomain = GetObject("WinNT://" & strDomain)
' -- MoveHere auf Domain ausführen
oDomain.MoveHere "WinNT://" & strDomain & "/" & strAlterName, _
strNeuerName
```

Active Directory Service Interface (ADSI)

```
say "Benutzer umbenannt!"
' -- SID und GUID nachher
Set oUser = GetObject("WinNT://" & strDomain & "/" & strNeuerName)
say flat(oUser.Get("objectSID")) & "," & oUser.Guid
```

Listing 13.29: Umbenennung eines Benutzers [ADSINT_BenutzerUmbenennung.wsf]



Achtung vor einer verwirrenden Fehlermeldung: Der Fehler »Der Netzwerkpfad wurde nicht gefunden.« kann auch bedeuten, dass der für das Benutzerkonto vorgesehene neue Name bereits existiert.

Benutzerkonto löschen

Delete() Ein Benutzer wird gelöscht durch den Aufruf der `Delete()`-Methode des Containers, in dem er enthalten ist. Das Beispiel zeigt das Löschen eines Domänenbenutzers. Bei der `Delete()`-Methode ist – wie beim Erzeugen – der Klassenname `user` anzugeben, um Verwechslungen mit eventuell gleichnamigen `Group`-Objekten zu vermeiden. Der Aufruf von `SetInfo()` ist nicht notwendig, `Delete()` wird sofort ausgeführt!

```
Set DomainObj = GetObject("WinNT://Sonnensystem")
DomainObj.Delete "user", "FoxMulder"
say "Benutzer gelöscht!"
```

Listing 13.30: Löschen eines Benutzerkontos [ADSINT_BenutzerLoeschen.wsf]

Deaktivieren eines Benutzerkontos

AccountDisabled Das nachfolgende Beispiel zeigt, wie mit Hilfe des Attributs `AccountDisabled` ein Benutzer deaktiviert werden kann, so dass er sich nicht mehr am Netz anmelden kann. Die Umkehrung der Aktion ist mit der Zuweisung von `False` möglich.

```
' -- Zugriff auf User-Objekt
Set oUser = GetObject("WinNT://Sonnensystem/FoxMulder")
' -- Deaktivierung
oUser.AccountDisabled = True ' = False zum reaktivieren!
' -- Cache schreiben
oUser.SetInfo
```

Listing 13.31: Deaktivieren eines Benutzerkontos [ADSINT_BenutzerDeaktivieren.wsf]

Benutzerrechte

NTRights.Exe Benutzerrechte (Benutzerprivilegien) wie »Lokale Anmeldung« und »Herunterfahren« sind leider bislang durch keine COM-Komponente abgedeckt. Eine Lösung bietet nur das NT4- bzw. Windows2000-Resource Kit mit dem Kommandozeilentool `NTRIGHTS.EXE`. Die Liste der verfügbaren Rechte zeigt das Tool beim Aufruf ohne Parameter an.

```
NTRights.Exe
Grants/Revokes NT-Rights to a user/group
usage: -u xxx User/Group
        -m \\xxx machine to perform the operation on (default local
machine)
        -e xxxxx Add xxxxx to the event log
        -r xxx revokes the xxx right
        +r xxx grants the xxx right
```

13.7.6 Terminalserver-Einstellungen

ADSI unterstützt nicht das Auslesen oder Setzen der Benutzereinstellungen für die Windows Terminal Services. Dies wird auch für das Windows2000-Active Directory nicht unterstützt.



13.7.7 Gruppenverwaltung mit der Group-Klasse

Die Group-Klasse repräsentiert eine in einer Domain oder auf einem Computer angelegte Benutzergruppe.

Attribut	Erläuterungen
ObjectType	2 = globale Gruppe, 4 = lokale Gruppe
ObjectSID	Security Identifier der Gruppe (wird vom System vergeben)
Description	textliche Beschreibung der Gruppe

Tabelle 13.25:
Attribute einer
WinNT-Gruppe

Methode	Erläuterungen
isMember()	Prüft, ob der angegebene Benutzer Mitglied der Gruppe ist. Zurückgeliefert wird <i>True</i> (ist vorhanden) oder <i>False</i> (ist nicht vorhanden).
Add()	Fügt einen Benutzer zu der Gruppe hinzu
Remove()	Löscht einen Benutzer aus der Gruppe

Tabelle 13.26:
Auf einer WinNT-
Gruppe ausführbare
Methoden

Gruppenzugehörigkeit anzeigen

Jedes User-Objekt hat ein Attribut namens *Groups*, das eine Liste (Collection) der Gruppen darstellt, denen der Benutzer angehört.

```
' -- Zugriff auf User-Objekt
Set oUser = GetObject("WinNT://Sonnensystem/FoxMulder")
' -- Iteration über alle Elemente der Group-Liste
For Each oGroup In oUser.Groups
    say oGroup.Name
Next
```

Listing 13.32: Gruppenzuordnung eines Benutzerkontos
[ADSINT_Gruppenzugehoerigkeit.wsf]



Aus der anderen Richtung können über die `Members`-Collection eines `Group`-Objekts alle enthaltenen Unterobjekte aufgelistet werden. Da eine Gruppe nicht nur Benutzer, sondern auch selbst wieder Gruppen enthalten kann, darf im Fall einer Typdeklaration die Laufvariable nicht auf `IADsUser` gesetzt sein, sondern muss neutral auf die Schnittstelle `IADs` deklariert werden.

```
Dim obj ' as ActiveDs.IADs
' -- Zugriff auf Group-Objekt
Set oGroup = GetObject("WinNT://Sonnensystem/Administratoren")
' -- Iteration über alle enthaltenen User
For Each obj In oGroup.Members
    say obj.Name & " (" & obj.Class & ") "
Next
```

Listing 13.33: Liste der in einer Gruppe enthaltenen Benutzer und Gruppen [ADSINT_Gruppenzusammensetzung.wsf]

Erstellen einer neuen Gruppe

**Benutzergruppen
anlegen und
füllen**

Das Einrichten einer Gruppe erfolgt analog zur Erstellung eines `User`-Objekts. Beachten Sie aber den bei `Create()` anzugebenden Klassennamen `group`. `GroupType` ist ein Pflichtattribut des WinNT-Verzeichnisdienstes, das aber automatisch auf den Wert 2 (globale Gruppe) gesetzt wird, wenn der ADSI-Client keinen Wert angibt.

Das Hinzufügen eines Benutzers zu einer Gruppe geht nicht vom `User`-Objekt, sondern vom `Group`-Objekt aus. Im `Group`-Objekt muss die Methode `Add()` aufgerufen werden. Als einziger Parameter wird der vollständige ADS-Pfad des `User`-Objekts angegeben, das in die Gruppe aufgenommen werden soll. Wenn der Benutzer bereits Mitglied der Gruppe ist, gibt ADSI den Fehler »Der angegebene Kontenname ist bereits ein Mitglied der Gruppe.« zurück.

```
' -- Gruppe erstellen
Set oDomain = GetObject("WinNT://Sonnensystem")
Set oGroup = oDomain.Create("group", "FBIAgenten")
oGroup.Put "GroupType", 4 ' 4 = Lokale Gruppe
oGroup.SetInfo
' -- Gruppe füllen
oGroup.Add "WinNT://Sonnensystem/FoxMulder"
oGroup.Add "WinNT://Sonnensystem/DanaScully"
' -- Gruppenmitglieder auflisten
For Each oUser In oGroup.Members
    say oUser.Name
Next
```

Listing 13.34: Erstellen und Füllen einer neuen Gruppe [ADSINT_NeueGruppe.wsf]

**Entfernen von
Gruppen-
mitgliedern**

Um ein unerwünschtes Gruppenmitglied loszuwerden, reicht ein Einzeiler, sofern die Bindung an die Gruppe erfolgt ist:

```
oGroup.Remove "WinNT://Sonnensystem/DanaScully"
```

13.7.8 Verwaltung von NT-Diensten mit der »Service«-Klasse

Die Service-Klasse repräsentiert einen Windows-NT-Dienst. NT-Dienste laufen im Hintergrund und sind auch aktiv, ohne dass ein Benutzer an der Konsole angemeldet ist. Ein Dienst ist ein Unterobjekt eines Computer-Containers.

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 Sessions, Resources und FileShares. Der Fileservice in Windows NT heißt *lanmanserver*. Der Zugriff auf den NT-Fileservice erfolgt folgendermaßen: WinNT://computername/lanmanserver. Ein Fileservice-Objekt ist im Gegensatz zu einem Service-Objekt ein Container: Ein Fileservice kann FileShares enthalten. Außerdem enthält ein Fileservice-Objekt zwei Collections:

- ▶ Sessions verweist auf die offenen Benutzersitzungen.
- ▶ Resources enthält die geöffneten Ressourcen.

Windows-NT-Dienst verwalten

Fileservice-Objekt

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
ServiceAccountName	Name des NT-Benutzers, unter dem sich dieser Dienst am System anmeldet. Der Wert »LocalSystem« repräsentiert den Systemaccount.
Dependencies	Name der Dienste, von denen dieser Dienst abhängig ist
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.
Description	Nur bei Fileservices: textliche Beschreibung
MaxUserCount	Nur bei Fileservices: maximale Anzahl der Benutzer; -1 = unbestimmt

Tabelle 13.27:
Attribute der
WinNT-Klassen
Service bzw.
Fileservice

Tabelle 13.28:
Methoden der
WinNT-Klassen
Service und
Fileservice

Methoden	Erläuterung
Start()	Starten des Dienstes
Stop()	Stoppen des Dienstes
Pause()	Anhalten des Dienstes
Continue()	Fortsetzen des Dienstes
SetPassword()	Setzen des Passworts des Dienstkontos

Liste aller Dienste

Das folgende Skript gibt die installierten Dienste mit ihrem Startverhalten und dem aktuellen Zustand aus. Status und Starttype liefern Konstanten zurück

```
Set oComputer = GetObject("WinNT://Sonne")
' -- Filter setzen
oComputer.Filter = Array("Service")
' -- alle NT-Dienste auflisten
For Each oService In oComputer
    say oService.Name & " (" & oService.Class & ") Status: " & _
oService.Status & " Start: " & oService.StartType
Next
```

Listing 13.35: Ausgabe aller NT-Dienste mit Status und Starttype
[ADSINT_Services.wsf]

Das Starten und Stoppen eines Dienstes ist mit den durch die Service-Klasse implementierten Methoden aus IADsService schnell erledigt.

```
' -- Bindung an Webserver-Dienst
Set oService = GetObject("WinNT://Sonne/w3svc")
' -- Dienst stoppen
oService.stop
Msgbox "Dienst gestoppt!"
' -- Dienst starten
oService.start
Msgbox "Dienst wieder gestartet!"
```

Listing 13.36: Starten und Stoppen des Internet Information Server-Dienstes
[ADSINT_W3SVCRestart.wsf]

13.7.9 Verzeichnisfreigaben verwalten

Verzeichnisfrei- gabe verwalten

Ein Fileshare-Objekt repräsentiert eine Verzeichnisfreigabe. Der Zugriff auf diese Objekte erfolgt ausschließlich über den Fileservice *ntlanmanserver*. Eine Freigabe ist direkt über einen ADSI-Pfad der Form WINNT://COMPUTERNAME/LANMANSERVER/FREIGABENAME erreichbar.

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 13.29:
Attribute der
WinNT-Klasse
FileShare

Mit ADSI können zwar die Freigaben aufgelistet sowie Freigaben erzeugt und gelöscht werden, aber die Rechte können nicht gesetzt werden – auch nicht mit der *ADsSecurity-Komponente* aus dem ADSI Resource Kit. Das Erzeugen von Verzeichnisfreigaben und die Vergabe von Rechten ist möglich mit der Windows Management Instrumentation (siehe Kapitel 14).



```
Set oFS = GetObject("WinNT://Sonne/lanmanserver")
For Each oShare In oFS
    say oShare.Name & " zeigt auf " & oShare.Path
Next
```

Listing 13.37: Liste aller Freigaben [ADSINT_ListeFreigaben.wsf]

```
Set oFS = GetObject("WinNT://Sonne/lanmanserver")
' -- Erstellen der Freigabe
Set OShare = oFS.Create("fileshare", "test")
OShare.Path = "f:\\"
OShare.SetInfo
Say "Freigabe wurde erstellt!"
' -- Löschen der Freigabe
oFS.Delete "fileshare", "test"
Say "Freigabe wurde wieder gelöscht!"
```

Listing 13.38: Anlegen und Löschen einer Freigabe [ADSINT_NeueFreigabe.wsf]

13.7.10 Benutzersitzungen und geöffnete Ressourcen

Der Service *LanManServer* eröffnet über die Collections *Sessions* und *Resources* aus der *IADsFileserviceOperations*- Schnittstelle den Zugriff auf Benutzersitzungen und geöffnete Ressourcen. Die in den Collections enthaltenen *Session*- bzw. *Resource*-Objekte besitzen keinen ADSI-Pfad, da sie keine persistenten Verzeichnisobjekte sind. Sie sind daher nicht über `GetObject()` direkt ansprechbar.

Sitzungen und
Ressourcen

Active Directory Service Interface (ADSI)

```
' -- Bindung an Fileservice
Set oFSop = GetObject("WinNT://mars/lanmanserver")
' -- Sitzungen
For Each oSession In oFSop.Sessions
    say oSession.User & " ist verbunden von " & _
    oSession.Computer & " seit " & _
    oSession.ConnectTime & " Sekunden"
Next
' -- Ressourcen
For Each oResource In oFSop.Resources
    say oResource.User & " verwendet " & _
    oResource.Path
Next
```

*Listing 13.39: Ausgabe der Benutzersitzungen und offenen Ressourcen
[ADSINT_SessionsUndResources.wsf]*



Es ist mit ADSI nicht möglich, Benutzersitzungen zu trennen und geöffnete Ressourcen zu schließen. In Tests kam es zu Problemen mit den Session-Objekten, die zum Teil mit der Fehlermeldung »Datei nicht gefunden« den Dienst verweigerten.

13.7.11 Druckersteuerung

ADSI definiert vier COM-Schnittstellen für den Druckerzugriff:

IADSPrintQueue	► IADSPrintQueue ermöglicht den Zugriff auf die persistenten Eigenschaften eines Druckers und der zugehörigen Druckerwarteschlange.
IADSPrintQueue-Operations	► IADSPrintQueueOperations definiert den nicht-persistenten Status einer Druckerwarteschlange, einen Zeiger auf die Collection (verkettete Liste) der Druckaufträge sowie Methoden, um die Warteschlange zu beeinflussen.
IADSPrintJob	► IADSPrintJob liefert Informationen über einen Druckauftrag, die diesem bei seiner Erzeugung mitgegeben wurden.
IADSPrintJobOperations	► IADSPrintJobOperations liefert im Gegensatz zu IADSPrintJob dynamische Informationen über den Druckauftrag (Position in der Warteschlange, Status des Druckauftrags, gedruckte Seiten). Außerdem definiert diese Schnittstelle Methoden, um einen einzelnen Druckauftrag zu manipulieren.

Implementierung im WinNT-Provider

PrintQueue und PrintJob	Im ADSI-Provider für Windows NT sind diese vier Schnittstellen in zwei Objekten implementiert. Ein PrintQueue-Objekt implementiert IADSPrintQueue und IADSPrintQueueOperations, wohingegen ein PrintJob-Objekt die Schnittstellen IADSPrintJob und IADSPrintJobOperations realisiert. In der

Objekthierarchie ist ein `PrintQueue`-Objekt einem `Computer`-Objekt untergeordnet. Das `PrintQueue`-Objekt wiederum enthält einen Verweis auf eine Collection von `PrintJob`-Objekten.

Voraussetzung für alle ADSI-Operationen ist ein Zeiger auf das passende `PrintQueue`-Objekt. Jede Warteschlange ist über einen ADSI-Pfad in der Form `WINNT://RECHNERNAME/DRUCKERNAME` eindeutig adressierbar. Lästig ist in diesem Zusammenhang, dass die `PrintJobs`-Collection nur sequenziell durchlaufen werden kann und nicht wie üblich über ein `Item`-Attribut verfügt; ein direkter Zugriff auf einen bestimmten Druckauftrag ist also nicht möglich.

Identifikation und Zugriff

Auskunft über den aktuellen Zustand des Druckers gibt das Attribut `Status`. Windows kennt 25 verschiedene Zustände für Drucker (z.B. `PRINTING`, `PAUSED`, `PAPER_OUT`, `PAPER_JAM`, `OUTPUT_BIN_FULL`), wobei mehrere Zustände gleichzeitig gelten können. Jeder Zustand repräsentiert daher ein Bit in dem vier Byte umfassenden `Status`-Attribut; der Gesamtzustand ergibt sich durch die bitweise Oder-Verknüpfung der Einzelzustände.

Druckerstatus

13.8 Windows2000 Active Directory-Administration mit ADSI

Das Active Directory ist der Verzeichnisdienst, den Microsoft erstmalig mit Windows2000 ausgeliefert 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 ohne Schema-Erweiterungen umfasst das Schema 142 Klassen und 863 Attribute. Im Vergleich dazu hat Windows NT4 nur 15 Klassen und 77 Attribute. Das Zugriffsprotokoll für das Active Directory ist LDAP. Sie müssen in ADSI daher den LDAP-Provider verwenden.

Active Directory

Durch die Installation eines Exchange Server 2000, der auf dem Active Directory aufsetzt, steigt die Zahl der Klassen auf 299 und die Zahl der Attribute auf 1705 (Angaben bestimmt mit Exchange 2000 RC1).

Exchange 2000

Ein Active Directory hat eine komplexe Struktur. Grundbaustein eines Active Directory 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 Domain 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

Sonderrollen	AD kennt einige wenige Sonderrollen, die nur ein Server haben kann. Diese werden <i>Flexible Single Master Operations (FSMO)</i> genannt. In der deutschen MMC findet man den Begriff »Betriebsmaster«.
Bäume in Wälder zusammenfassen	Domänen können zu so genannten <i>Trees</i> (Bäumen) und diese wiederum zu <i>Forests</i> (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.
	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: ▶ 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 gegenüber dem NT4-Verzeichnis 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 <code>OrganizationalUnit</code> und <code>Container</code>. ▶ fein granulierte Zugriffsrechte auf jeder Ebene und auch für jedes Attribut, so dass eine Delegation von administrativen Aufgaben möglich wird ▶ ein erweiterbares, objektorientiertes Schema

13.8.1 ADS-Objektmodell

Oberstes Element jedes Active Directory ist das Objekt `rootDSE`. Der Verzeichnisdienst teilt sich darunter in drei Partitionen:

- Der **DefaultNamingContext** ist die Sicht auf den Verzeichnisdienst, wie ihn das MMC Snap-In ACTIVE DIRECTORY-BENUTZER UND COMPUTER bietet.
- Im **ConfigurationNamingContext** werden Einstellungen zu dem Active Directory zuarbeitenden Diensten, den Sites und dem Inter-Site-Transport, den erweiterten Rechten, den *Display Specifiers* (ein *Display Specifier* legt fest, mit welchen Eigenschaftsfenstern eine Klasse in der MMC angezeigt wird) und den *Well Known Security Principals* (z.B. *Everybody*, *System*) abgelegt.
- Der **SchemaNamingContext** enthält das Schema des ADs.

DefaultNaming-Context

**SchemaNaming-Context
Configuration
NamingContext**

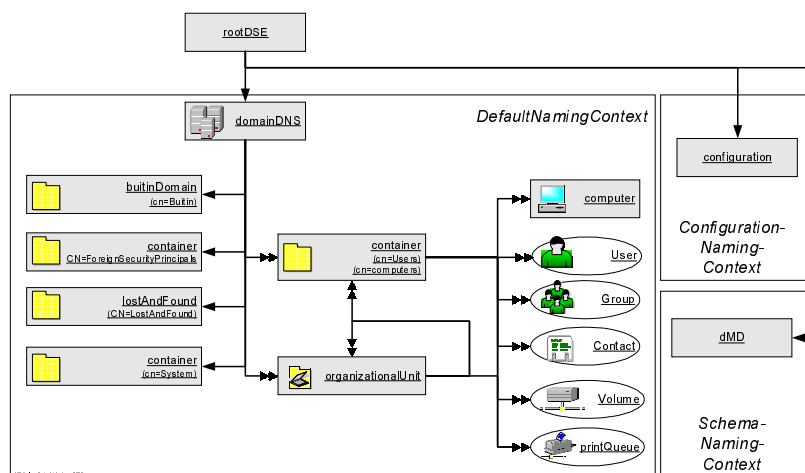


Abbildung 13.19:
Ein kleiner Aus-
schnitt aus dem
komplexen Objekt-
modell des Active
Directory Service

Mit dem folgenden Skript ermitteln Sie die ADSI-Pfade der Partitionen in einem Active Directory:

```
Dim root ' As IADs
Set root = GetObject("LDAP://Sonne2000/RootDSE")
say "Servername: " & root.Get("Servername")
say "defaultNamingContext: " & root.Get("defaultNamingContext")
say "ConfigurationNamingContext: " & _
root.Get("ConfigurationNamingContext")
say "SchemaNamingContext: " & root.Get("SchemaNamingContext")
```

Listing 13.40: Ausgabe der Partitionen [ADSIADS_Partitionen.wsfl]

Das Skript liefert in der Testumgebung die folgende Ausgabe:

```
Servername: CN=SONNE2000,CN=Servers,CN=Standardname-des-ersten-
Standorts,CN=Sites,CN=Configuration,DC=IT-Visions,DC=de
defaultNamingContext: DC=IT-Visions,DC=de
ConfigurationNamingContext: CN=Configuration,DC=IT-Visions,DC=de
SchemaNamingContext: CN=Schema,CN=Configuration,DC=IT-Visions,DC=de
```

Listing 13.41: Ausgabe des obigen Listing

Klassen im Active Directory**Ausgewählte Klassen**

Alle vordefinierten Klassen im Active Directory sind Container-Klassen und unterstützen daher IADs und IADsContainer. Die nachfolgende Tabelle zeigt eine Auswahl der wichtigsten der 142 AD-Klassen mit ihren Schlüsselattributen (fast immer cn) und den unterstützten ADSI-Schnittstellen.

Tabelle 13.30:
Wichtigste Klassen
im Active Directory

Verzeichniseintrag	Klassenname (LDAP-Name)	Schlüsselattribut	Unterstützte Zusatzschnittstellen
Benutzer	user	cn	IADsUser
Gruppe	group	cn	IADsGroup
Container	container	cn	(keine)
Organisationseinheit	organizationalUnit	ou	IADsOU
Domänenbestandteil	domainDNS	dc	(keine)
Verzeichnisfreigabe	Volume	cn	(keine)
Computer	Computer	cn	IADsComputer
Drucker	printQueue	cn	IADsPrintQueue IADsPrintQueueOperations
Kontakt	Contact	cn	IADsUser
AD Schema	dmD	cn	(keine)
Attribut im Schema	attributeSchema	cn	(keine)
Klasse im Schema	classSchema	cn	(keine)
Konfigurationscontainer	configuration	cn	(keine)

13.8.2 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 Servername wird der so genannte *Locator Service* verwendet. Beim serverlosen Binden sucht der Active Directory Locator Service mit Hilfe des Domain Name Service (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 DN wird der DefaultNamingContext in der aktuellen Domäne angesprochen.

Bindung über GUIDs

Bei der Adressierung über einen Textpfad besteht die Gefahr, dass Verzeichnisobjekte umbenannt wurden. Active Directory ermöglicht daher die Bindung über eine GUID, die für ein Verzeichnisobjekt unveränderlich ist. Die GUID muss natürlich für ein Objekt bekannt sein.

```
LDAP://sonne2000/<GUID=228D9A87C30211CF9AA400AA004A5691>
```

GUID-Bindung

Für die Standardcontainer in einem Active Directory gibt es eine besondere Unterstützung. Für diese so genannten *Well Known Objects* besteht eine vordefinierte GUID, die in jedem Active Directory gleich ist.

Well-Known-Objects

```
LDAP://<WKGUID=a9d1ca15768811d1aded00c04fd8d5cd,dc=it-visions,dc=de>
```

Bitte beachten Sie, dass hierbei die Ansprache über WKGUID= erfolgt und die dahinter angegebene GUID nicht die wirkliche GUID des Objekts ist. Auch die Standard-Container erhalten bei der Installation eines Active Directory eine individuelle GUID; die WKGUID ist ein allgemein gültiger Alias.

Well-Known-Object	GUID
CN=DELETED OBJECTS	18E2EA80684F11D2B9AA00C04F79F805
CN=INFRASTRUCTURE	2FBAC1870ADE11D297C400C04FD8D5CD
CN=LOSTANDFOUND	AB8153B7768811D1ADED00C04FD8D5CD
CN=SYSTEM	AB1D30F3768811D1ADED00C04FD8D5CD
OU=DOMAIN CONTROLLERS	A361B2FFFFD211D1AA4B00C04FD7D83A
CN=COMPUTERS	AA312825768811D1ADED00C04FD8D5CD
CN=USERS	A9D1CA15768811D1ADED00C04FD8D5CD

Tabelle 13.31:
Liste der Well-Known-Objects

Das folgende Listing zeigt die Ermittlung der WKGUIDs per Programmcode. Diese Automatisierungslösung erscheint auf den ersten Blick überflüssig. Auf den zweiten Blick ist diese knifflige Aufgabe jedoch ein gutes Anschauungsobjekt, da hier der ADS-Datentyp `DNWithBinary` verwendet wird. Die WKGUIDs sind in dem mehrwertigen Attribut `WELLKNOWN OBJECTS` gespeichert, wobei jeder einzelne Wert wiederum ein Objekt des Typs `ActiveDs.DNWithBinary` ist.

Ermittlung der WKGUID

Die von diesem Objekt in binärer Form zurückgegebene GUID muss mit `binGUIDtoSTR()` vor der Ausgabe in einen String umgewandelt werden.

```
' ### Umwandlung einer binären GUID in einen String
Function binGUIDtoSTR(guid)
Dim i
For i = (LBound(guid) + 1) To (UBound(guid) + 1)
    binGUIDtoSTR = binGUIDtoSTR & Hex(AscB(MidB(guid, i, 1)) \ 16) &
    Hex(AscB(MidB(guid, i, 1)) Mod 16)
End Function
```

Active Directory Service Interface (ADSI)

```
Next
End Function
!!!
Sub ADSIADS_GetWellKnownObjects()
Dim binarray
Dim obj ' As IADs
Dim strGUID ' As String
Dim oBinVal 'As ActiveDs.DNWithBinary
Set obj = GetObject("LDAP://sonne2000/dc=it-visions,dc=de")
' -- Auslesen der Liste der WellKnownObjects
binarray = obj.Get("wellKnownObjects")
For Each oBinVal In binarray
    ' -- Umwandlung in String
    strGUID = binGUIDtoSTR(oBinVal.BinaryValue)
    say oBinVal.DNString & " = " & strGUID
Next
End Sub
```

*Listing 13.42: Listing 5.136: Ermittlung der WKGUIDs der Well Known Objects
[ADSIADS_GetWellKnownObjects.wsf]*

13.8.3 Benutzerverwaltung

Die Benutzerverwaltung gehört zu den wichtigsten Aufgaben in einem Active Directory. Dabei werden hier insbesondere die Unterschiede zum WinNT-Provider hervorgehoben.

Liste der Benutzerkonten

SAMAccount- Name

Die Ausgabe der Benutzerliste unterscheidet sich von dem Listing aus dem Kapitel zum WinNT-Provider nur hinsichtlich des ADSI-Pfads. Beachten Sie jedoch, dass dieses Skript nur die User-Objekte erfasst, die sich in einem bestimmten Container, hier im Standardcontainer `Users`, befinden. Unter WinNT gab es immer nur einen Container pro Domain, im Active Directory können beliebig viele Container in einer Hierarchie erzeugt werden. Im AD gibt es natürlich zusätzliche Attribute: Das Attribut `SAMAccountName` ist der Benutzeranmeldename für alte NT 3.51- und NT4-Clients. `Department` und `telephonenumber` sind zwei der vielfältigen neuen Attribute, die das Active Directory bietet. Da beides optionale Attribute sind, sollten Sie die Hilfsroutine `ADSIGet()` verwenden.

```
' ### Fehlertolerantes Auslesen eines Attributs in einem
Verzeichnisobjekt
Function ADSIGet(obj, attribut)
ADSIGet = ""
On Error Resume Next
ADSIGet = obj.Get(attribut)
End Function
```


Windows2000 Active Directory-Administration mit ADSI

```
Dim c ' As IADsContainer
Dim u ' As IADsUser
Set c = GetObject("LDAP://Sonne2000/cn=users,dc=it-visions,dc=de")
c.Filter = Array("User")
For Each u In c
    say u.AdsPath & " : " & u.Get("SAMAccountname")
    say ADSIGet(u, "Department") & ", " & ADSIGet(u, "telephonenumber")
Next
```

*Listing 13.43: Listing 5.137: Benutzer in einem Container auflisten
[ADSIADS_BenutzerListe.wsfl]*

Um mit einer Anweisung mehrere Container zu durchsuchen, können Sie ADSI-Queries verwenden.

**Benutzerliste via
ADSI-Query**

```
Dim strLDAPQuery ' As String
Dim objcon ' As ADODB.Connection
' -- ADO-Connection-Objekt erzeugen
Set objcon = CreateObject("ADODB.Connection")
' -- Verbindung zu dem OLE DB-Provider für ADSI
ConString = "Provider=ADSDSOObject"
objcon.Open ConString, "administrator", ""
' -- Selektion aller User im ganzen Active Directory
strLDAPQuery = "<LDAP://sonne2000/dc=it-visions,dc=de>;  
(objectclass=user);samaccountname,adspath;subtree"  
' --- Ausführen der Abfrage
Set rs = objcon.Execute(strLDAPQuery)
' --- Ausgabe der Ergebnisse
While Not rs.EOF
    say rs("adspath") & ";" & rs("samaccountname")
    rs.MoveNext
Wend
' -- Schließen der Objekte
rs.Close
objcon.Close
```

Listing 13.44: Liste aller Benutzer in einem Active Directory [ADSIADS_ADOQuery.wsfl]

Mit dem nachfolgenden Query-String können Sie alle Objekte eines Active Directory auflisten.

```
strLDAPQuery = "<LDAP://sonne2000>;adspath;subtree"
```



Benutzer anlegen

Neuen Benutzer im AD anlegen

Ähnlich, aber dennoch nicht identisch zu NT4 ist das Anlegen eines Benutzers unter Active Directory. Neben dem Verzeichnisnamen benötigt jeder AD-Benutzer als Pflichtattribut einen `SAMAccountName`. Da bei LDAP anders als bei NT4 der Attributname Teil des RDNs ist, muss dem neuen Benutzernamen in der `Create()`-Methode getrennt durch ein Gleichheitszeichen der Attributname vorangestellt werden, der der Identifizierung der Instanzen dieser Klasse dient (hier: `cn`).

```
Function ADSIADS_createUser (container, un)
Dim c ' As IADsContainer
Dim u ' As IADsUser
' -- Bindung an Container
Set c = GetObject(container)
' -- Erzeugung des neuen Benutzers
Set u = c.Create("user", "cn=" & un)
' -- Attribute setzen
u.Put "samAccountName", CStr(un)
u.SetInfo
' -- Konto aktivieren
u.AccountDisabled = False
u.SetInfo
say "Benutzer wurde angelegt:" & Chr(13) & u.AdsPath
Set w2k_createUser = u
End Function
```

*Listing 13.45: Anlegen eines neuen Benutzers unter Windows2000
[ADSIADS_createUser.wsf]*



Bitte beachten Sie, dass zur Umgehung von Datentypproblemen bei `Put()` die übergebenen Werte mit Typkonvertierungsfunktionen »behandelt« werden müssen (vgl. Ausführungen in Kapitel 3).

Der Container ist natürlich in LDAP-Form zu spezifizieren. Ein Aufruf der Routine könnte folgendermaßen aussehen:

```
ADSIADS_createUser "LDAP://sonne2000/cn=users,dc=it-visions,dc=de",  
"ThomasGartner"
```

Terminalserver-Einstellungen

Die Einstellungen der Registerkarte »Terminaldienstprofile« zu einem Benutzer-Objekt im Active Directory können nicht via ADSI gelesen oder geändert werden.



Abbildung 13.20:
Registerkarte »Terminaldienstprofile«

13.8.4 Organisationseinheiten verwalten

Eine hervorstechende Eigenschaft des Active Directory ist es, beliebige Organisationsstrukturen in Form von Verzeichnis-Containern nachzubilden. Ein solcher Container heißt im AD `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 `Create()`. Mit `Locality` wird ein Attribut gesetzt, das in der Schnittstelle `IADsOU` definiert ist, während `ManagedBy` verzeichnisdienstspezifisch ist. `ManagedBy` erwartet einen DN eines Benutzers oder einer Gruppe.

Anlegen von OUs

```
Dim OUListe ' Array der anzulegenden OUs
Dim objCon ' As IADsContainer
Dim objOU ' As IADsOU
Const CONTAINER = "LDAP://sonne2000/ou=it-visions,dc=it-visions,dc=de"
Const MANAGER = "CN=HS,OU=IT-Visions,DC=IT-Visions,DC=de"
OUListe = Array("Einkauf", "Vertrieb", "Entwicklung", "Finanzen")
' -- Bindung an Container
Set objCon = GetObject(CONTAINER)
For Each ou In OUListe
    ' -- Erzeugung der neuen OU
    Set objOU = objCon.create("organizationalunit", "ou=" & ou)
    ' -- Attribute setzen
    objOU.LocalityName = "Essen"
    objOU.Description = "www.IT-Visions.de"
    objOU.Put "ManagedBy", MANAGER
```

Active Directory Service Interface (ADSI)

```
objOU.SetInfo  
say "OU wurde angelegt:" & Chr(13) & objOU.ADsPath  
Next
```

*Listing 13.46: Anlegen von Organisationseinheiten im Active Directory
[ADSIADS_NeueOUs.wsf]*

13.9 Exchange Server-Administration mit ADSI

- MAPI, DAPI** In der Vergangenheit war die Programmierung des Microsoft Exchange Servers durch die Schnittstellen MAPI (Messaging Application Programming Interface) und DAPI (Directory Application Programming Interface) eine große Herausforderung. Durch die Unterstützung von LDAP kann der Exchange Server aber inzwischen auch via ADSI programmiert werden.
- LDAPv3** Exchange unterstützt LDAP seit Version 5.0, dort jedoch zunächst nur lesend (LDAPv1). Der volle Schreib-/Lesezugriff via LDAPv3 besteht erst seit Version 5.5. Über den ADSI-Provider für LDAP ist es möglich, viele administrative Aufgaben auf einem Exchange Server zu automatisieren. Mit ADSI können alle Einträge im Exchange-Verzeichnisdienst verändert werden. Ein Zugriff auf den Informationsspeicher ist jedoch nicht möglich.
- Features** Sie können via ADSI beispielsweise
- ▶ Postfächer, benutzerdefinierte Empfänger und E-Mail-Verteiler anlegen und löschen
 - ▶ E-Mail-Verteiler verändern
 - ▶ zahlreiche Konfigurationseinstellungen einsehen und ändern



Was Sie mit ADSI allerdings nicht können ist das Anlegen und Löschen von öffentlichen Ordnern. Ein Löschen in ADSI entfernt den Ordner zwar aus dem Verzeichnisbaum, nicht jedoch aus der Datenbank. Dazu müssen Sie CDO verwenden.

13.9.1 LDAP-Konfiguration

Um auf einen Exchange Server mit ADSI über LDAP zuzugreifen, ist lediglich eine Installation der ADSI-Komponenten auf dem jeweiligen Client-PC notwendig. Eine Installation auf dem Server ist nicht erforderlich. Das LDAP-Protokoll auf dem Exchange Server muss jedoch aktiviert und konfiguriert werden. Sie müssen dabei zwei Dinge beachten:

- LDAP-Aktivierung** ▶ Aktivieren Sie LDAP in der Standort-Konfiguration (KONFIGURATION/PROTOKOLLE/LDAP).

- Wenn Sie Suchanfragen benutzen wollen, dann achten Sie darauf, dass die Suchoptionen Ihren Bedürfnissen angepasst sind. In der Regel ist der Wert »Maximale Anzahl der Suchresultate« zu niedrig eingestellt.

Einstellungen

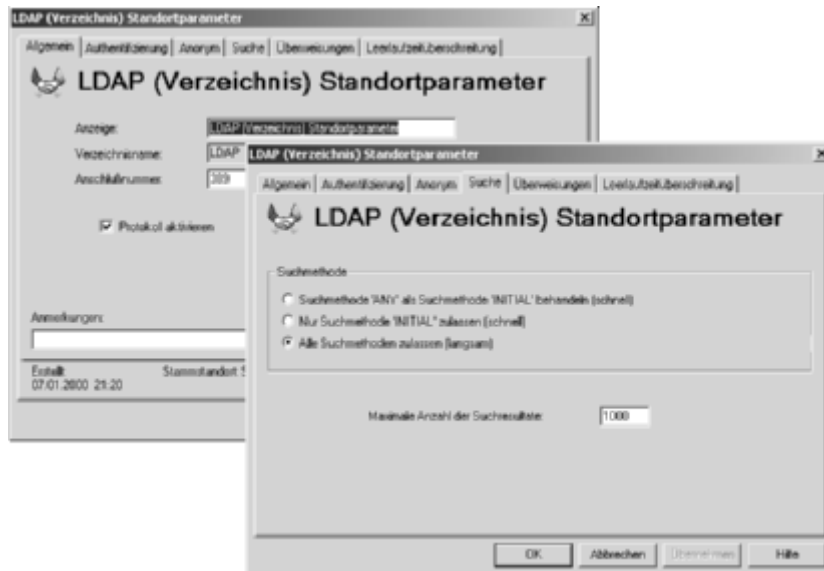


Abbildung 13.21:
LDAP-Konfiguration
im Exchange
Server

Eine falsche Konfiguration kann zu Fehlern mit höchst merkwürdigen Fehlermeldungen führen, wie »Die Namensbegrenzung der lokalen Netzwerkkarte wurde überschritten«. Wer diese Fehlermeldung sieht, muss nicht nach dem Hardware-Service rufen. Hinter dieser kuriosen Meldung verbirgt sich ein zu geringer Eintrag in der Option »Maximale Anzahl der Suchresultate« in der Site-Konfiguration im Exchange Server.



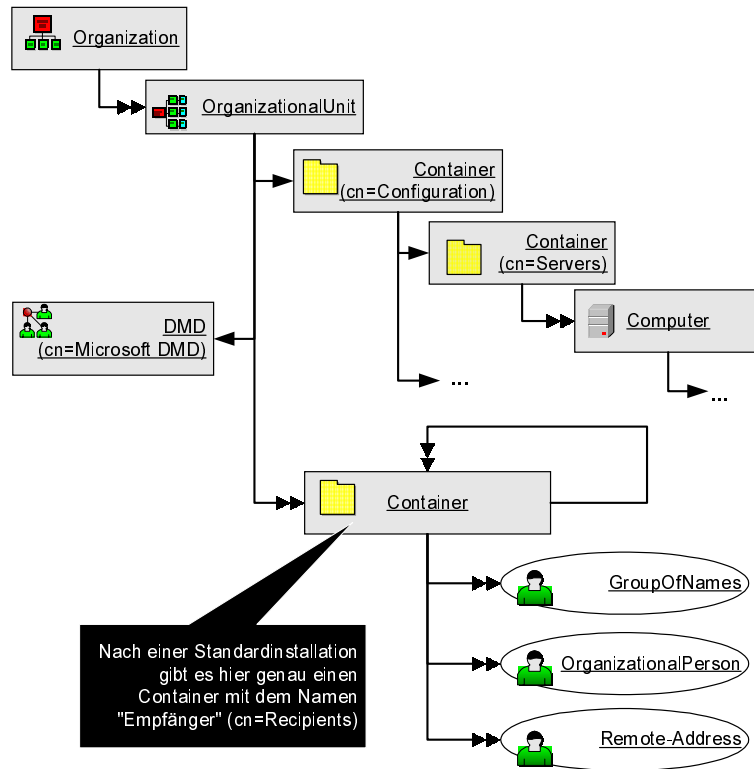
13.9.2 Exchange-Objektmodell

Die folgende Abbildung zeigt die wesentlichen Teile des Objektmodells des Exchange-Verzeichnisdienstes. An der Spitze steht eine Instanz der Klasse Organization, die eine oder mehrere OrganizationalUnit-Objekte enthält. Jede OrganizationalUnit enthält zwei Instanzen der Klasse Container, eine mit dem Namen Configuration und eine mit dem Namen Recipients. Der Configuration-Container enthält zahlreiche Leaf-Objekte und weitere Container (z.B. Servers und Protocols). Der Recipients-Container enthält die Exchange-Empfänger, die in einer Container-Hierarchie angeordnet sein können. Außerdem enthält die OrganizationalUnit (OU) ein DMD-Objekt mit dem Namen Microsoft DMD. DMD steht für *Directory Management Domain* und enthält das Schema des Exchange Servers.

Exchange- Objektmodell

Active Directory Service Interface (ADSI)

Abbildung 13.22:
Kernbereich
des Objektmodells
des Exchange
Servers 5.5



Klassen und Schnittstellen

Exchange-Klassen

Die nachfolgende Tabelle zeigt die wichtigsten Klassennamen im Exchange-Verzeichnisdienst und deren Bedeutung.

Keine Exchange-Klasse implementiert über IADs bzw. IADsContainer hinausgehende ADSI-Standschnittstellen.

Tabelle 13.32:
Klassen im
Exchange-
Verzeichnis

Klassenname (LDAP-Name)	Typ	Bedeutung
organizationalPerson	Leaf	Postfach (Mailbox)
remote-address	Leaf	Benutzerdefinierter Empfänger
groupOfNames	Leaf	Verteilerliste
container	Container	Exchange-Container
Organization	Container	Exchange-Organisation
OrganizationalUnit	Container	Exchange-Organisationseinheit
Computer	Container	Ein Computer mit installiertem Exchange Server

Hinweis: Verwechseln Sie nicht die Exchange-Klasse `Container` mit `IADsContainer`. `Container` ist eine Exchange-Klasse, die durch `IADsContainer` repräsentiert wird. `IADsContainer` wird aber auch als Meta-Schnittstelle für andere Exchange-Klassen (z.B. `Organization`, `OrganizationalUnit`, `Protocol-Cfg-Shared-Site`) verwendet.

Namensverwirrung

Beim Zugriff auf die Attribute eines Exchange-Objekts muss beachtet werden, dass Exchange vier verschiedene Bezeichnungen für ein und dasselbe Attribut kennt:

- ▶ den im Exchange-Admin-GUI (ADMIN.EXE) angezeigten Namen
- ▶ den internen *Admin Display Name*
- ▶ den eigentlichen Verzeichnisnamen (*Common Name*)
- ▶ den *LDAP-Namen*

Die Situation wird dadurch kompliziert, dass Exchange intern für die Attribute einen anderen Verzeichnisnamen (den so genannten *Common Name*) benutzt, der weder dem *Admin Display Name* noch dem LDAP-Namen entspricht. Der *Common Name* wird vom ADSI-Nutzer benötigt, um das Schema per Programm auszulesen. Ein ADSI-Client kann das Schema wie jeden Ast im Exchange-Baum auflisten. Zum Zugriff via LDAP benötigen Sie den *LDAP-Namen*!

Mehrere verschiedene Attributnamen



Die Namen sind zum Teil ähnlich (z.B. ist `Company` sowohl *Display-Name*, als auch *Common Name* und *LDAP-Name*), zum Teil aber auch ganz verschieden: So hat der *Common Name* `Locality-Name` den *Display-Namen* `City` und als *LDAP-Namen* ein schlichtes `L`. Ein zweites Beispiel: Hinter dem einfachen *Display-Namen* `Office` steckt das Verzeichnisattribut `Physical-Delivery-Office-Name`, das in LDAP ohne Bindestriche geschrieben werden muss, also `PhysicalDeliveryOfficeName`.

Ähnlichkeiten und Unterschiede

Den Zusammenhang zwischen den Namen zeigt Ihnen das Exchange-Schema. Die folgende Tabelle stellt die Namen für die wichtigsten Attribute gegenüber.

Bildschirm-anzeige im deutschen Admin-GUI	Admin Display Name (im Schema)	Exchange Common Name (Verzeichnisname)	LDAP-Name
Verzeichnisname	Directory Name	Common-Name	Rdn
- (wird nicht angezeigt)	Obj-Dist-Name	Obj-Dist-Name	Distinguished Name
Anzeige	Display Name	Display-Name	Cn

Tabelle 13.33:
Unterschiedliche Bezeichnungen für ein und dasselbe Attribut im Exchange Server

Tabelle 13.33:
Unterschiedliche
Bezeichnungen für
ein und dasselbe
Attribut im
Exchange Server
(Forts.)

Bildschirm- anzeige im deutschen Admin-GUI	Admin Display Name (im Schema)	Exchange Common Name (Verzeichnisname)	LDAP-Name
Alias	Alias Name	Mail-Nickname	Uid
Vorname	First Name	Given-Name	GivenName
Nachname	Last Name	Surname	Sn
Firma	Company	Company	Company
Adresse	Address	Address	PostalAddress
PLZ	Postal Code	Postal-Code	PostalCode
Ort	City	Locality-Name	L
Staat	Country	Text-Country	Co
Abteilung	Department	Department	Department
Büro	Office	Physical-Delivery- Office-Name	PhysicalDelivery OfficeName
Telefon	Phone Number	Telephone-Office1	TelephoneNumber
Antwort- E-Mail- Adresse	E-Mail Addresses	Proxy-Addresses	Mail
(Weitere) E-Mail- Adressen	E-Mail Addresses	Proxy-Addresses	OtherMailbox
Primäres Windows NT- Konto	Primary Windows NT Account	Assoc-NT-Account	Assoc-NT-Account

13.9.3 Identifikation und Bindung

Schlüsselattribute

Fast alle Exchange-Klassen verwenden als Schlüsselattribut **CN**. Nur die **Organization (O)** und die **OrganizationalUnit (OU)** bilden eine Ausnahme. Jeder Container und jedes Empfänger-Objekt wird jedoch über den **CN** identifiziert. Sofern ein Empfänger nicht direkt im Standardempfänger-Container, sondern tiefer in einer Container-Hierarchie liegt, muss der Distinguished Name entsprechend viele **CN**-Einträge aufweisen.

Aufbau des ADSI-Pfads

Der Distinguished Name (DN) eines Exchange-Verzeichniseintrags spiegelt die Exchange-Organisationsstruktur wider: Der Name der Organisation fließt nach dem Eintrag **O=** ein, die Exchange-Site nach **OU=** (*Organizational Unit*). Der Empfänger-Container wird durch **CN=** (*Common Name*) spezifiziert. Sofern Empfänger-Container in Untercontainer aufgeteilt sind, muss der komplette Pfad in Form weiterer **CN**-Einträge spezifiziert werden – ebenso alle Empfänger und Elemente des Konfigurationsbaums. Anhand

des ADSI-Pfads lässt sich also die Art des Eintrags nicht erkennen. Dazu müssen Sie das `Class`-Attribut aus der `IADs`-Schnittstelle verwenden.

Eintragsart	Beispiel Top-Down-Syntax (auch: Big Endian)	Beispiel Bottom-Up-Syntax (auch: Little Endian)
Root-Element	LDAP://SONNE	LDAP://SONNE
Container	LDAP://SONNE/ O=UNIVERSUM/ OU=SONNENSYSTEM/ CN=RECIPIENTS/ CN=MITARBEITER	LDAP://SONNE/ CN=MITARBEITER, CN=RECIPIENTS, OU=SONNENSYSTEM, O=UNIVERSUM
Empfänger	LDAP://SONNE/ O=UNIVERSUM/ OU=SONNENSYSTEM/ CN=RECIPIENTS/ CN=MITARBEITER/ CN=FOX MULDER	LDAP://SONNE/ CN=FOX MULDER, CN=MITARBEITER, CN=RECIPIENTS, OU=SONNENSYSTEM, O=UNIVERSUM
Konfiguration des POP3-Protokolls	LDAP://SONNE/ O=UNIVERSUM/ OU=SONNENSYSTEM/ CN=CONFIGURATION/ CN=PROTOCOLS/ CN=POP3	LDAP://SONNE/ CN=POP3, CN=PROTOCOLS, CN=CONFIGURATION, OU=SONNENSYSTEM, O=UNIVERSUM

*Tabelle 13.34:
Beispiele für ADSI-
Pfade im Exchange
Server. Die ADSI-
Pfade sind jeweils
in eine Zeile zu
schreiben.*

13.9.4 Das Exchange-Schema

Die Objekthierarchie des Exchange Servers wird durch das Exchange Administratorprogramm (`ADMIN.EXE`) anschaulich dargestellt. Sehr hilfreich für die ADSI-Programmierung des Exchange Servers ist, dass `ADMIN.EXE` auch eine Ansicht des Exchange-Schemas bietet. Für die Schemaansicht ist es notwendig, das Administratorprogramm im so genannten Raw-Modus zu starten (Kommandozeilenoption `/R`).

**Raw-Modus in
Admin.exe**

```
admin.exe /r
```

Der Raw-Modus liefert zwei wertvolle Ansichten:

- Im Basiseigenschaftsfenster erhalten Sie Auskunft über den exakten Distinguished Name (DN) eines Eintrags.
- Durch Auswahl der Option `VOLLSTÄNDIGES VERZEICHNIS` im Menü `ANSICHT` zeigt sich das Exchange-Schema als zusätzlicher Ast in der Baumdarstellung.

**Basiseigenschafts-
fenster**

Schema-Ansicht

Active Directory Service Interface (ADSI)

Abbildung 13.23:
Basiseigenschafts-
fenster einer
Mailbox

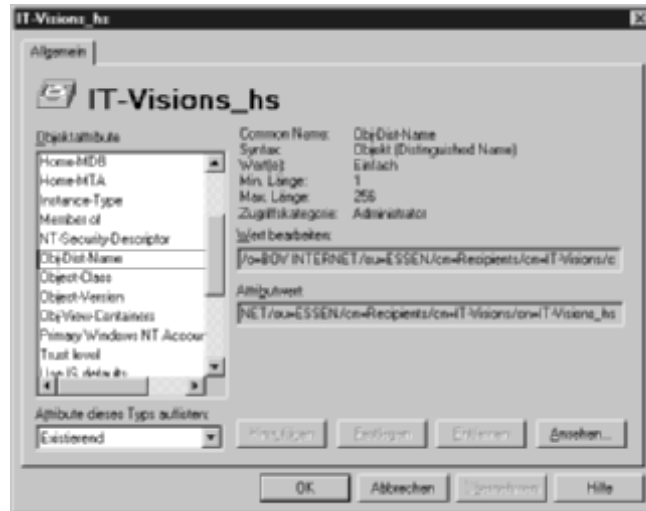
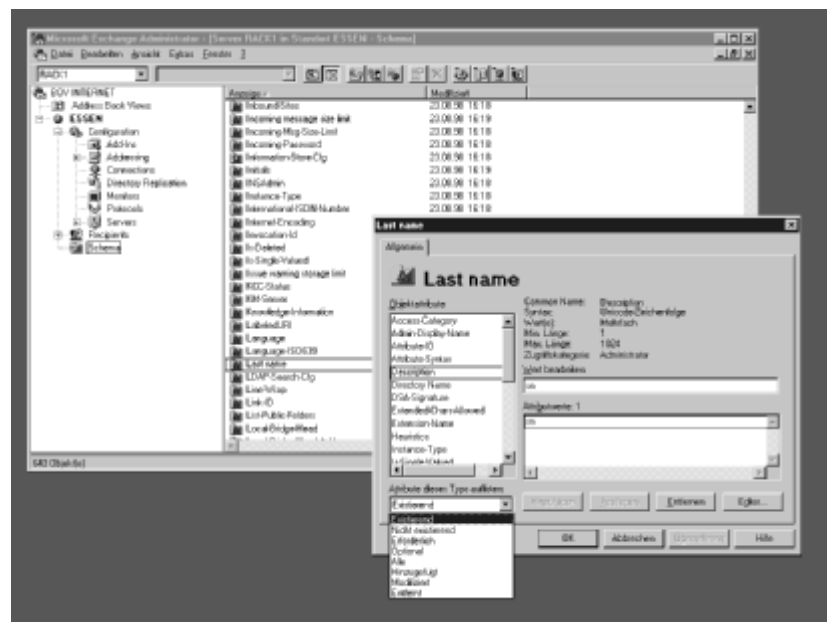


Abbildung 13.24:
Schema-Ansicht im
Raw-Modus des
Exchange-
Admin-GUIs



Auf der Suche nach dem LDAP-Namen eines Attributs

Häufig wird es darum gehen, Eigenschaften von Objekten zu setzen, die aus dem GUI bekannt sind. Der Weg von der Bildschirmanzeige zum LDAP-Namen ist jedoch schwierig. Von der Bildschirmanzeige zum korrekten LDAP-Namen eines Attributs sind zwei Schritte notwendig:

- Zunächst muss in den **Basiseigenschaften** (Menü DATEI/BASISEIGENSCHAFTEN) des gewünschten Verzeichniseintrags der *Admin Display Name* für das Attribut herausgesucht werden. Dieser weicht sowohl bei der englischen als auch bei der deutschen Version der ADMIN.EXE von der auf dem Bildschirm dargestellten Bezeichnung ab.
- Nun kann man den *LDAP-Namen* des Attributs im Exchange-Schema nachschlagen. Dort sucht man den im ersten Schritt ermittelten *Admin Display Name*. Erneut über DATEI/BASISEIGENSCHAFTEN gelangt man in das Eigenschaftsfenster des Attributs (man befindet sich dann also in den Eigenschaften der Eigenschaften). Dabei ist die unter DESCRIPTION angegebene Bezeichnung der korrekte *LDAP-Name*.

Ermittlung des Admin Display Name

Die Description enthält den LDAP-Namen

Da das Exchange Directory nicht erweiterbar ist, können keine Ergänzungen oder Veränderungen am Schema vorgenommen werden. Lediglich die fünfzehn vorhandenen Custom Attributes (zehn in Exchange 5.0) können beliebig belegt werden.

Erweiterbarkeit

Zugriffsmöglichkeiten

Ob ADSI via LDAP ein Attribut überhaupt betrachten und verändern kann, bestimmen die Eigenschaften *Heuristic* und *Access-Category* im Schema für jedes einzelne Attribut. Sie finden diese Information in der gleichen Ansicht, in der Sie auch den LDAP-Namen finden können (siehe oben). Beispielsweise bedeutet der *Heuristic*-Wert 12 für das Attribut *Assoc-NT-Account*, dass es via LDAP zwar gesetzt, aber nicht ausgelesen werden kann.

Heuristic und Access-Category

Bit	Erläuterung
Bit 0	Replikation zwischen Standorten (0=Ja, 1=Nein)
Bit 1	Sichtbar via LDAP (0=Nein, 1=Ja)
Bit 2	Zugriff für authentifizierte Clients (0=Nein, 1=Ja)
Bit 3	0=Attribut kann via ADSI gelesen und gesetzt werden 1=Attribut kann via ADSI nur gesetzt werden
Bit 4	Attribut ist sichtbar im Admin-GUI (0=Nein, 1=Ja)

Tabelle 13.35:
Flag-Werte für
Heuristic

Wert	Erläuterung
0	Nur das System darf das Attribut modifizieren.
1	Benutzer mit Admin-Rechten können das Attribut modifizieren.
2	Benutzer mit User-Rechten können das Attribut modifizieren.
3	Benutzer mit Änderungs-Rechten können das Attribut modifizieren.

Tabelle 13.36:
Werte für *Access-Category*

13.9.5 Empfänger auflisten

Empfängerliste Das nachfolgende Beispiel zeigt, wie ein Zugriff auf einen Empfänger-Container im Exchange-Verzeichnisdienst möglich ist.

```
Set container = GetObject("LDAP://Sonne/cn=recipients,
ou=Sonnensystem,o=Universum")
container.Filter = Array("organizationalPerson")
For Each r In container
    say r.cn & ", " & r.mail & ", " & r.AdsPath
Next
```

Listing 13.47: Liste der Postfächer in einem Container [ADSIEX_EmpfaengerListe.wsfl]

Nachdem mit `GetObject()` und dem entsprechenden ADSI-Pfad ein Zugriff auf den Exchange-Container erfolgt ist, wird mit dem `Filter`-Attribut die Ergebnismenge auf die Einträge der Klasse `organizationalPerson` beschränkt. Danach erfolgt in einer Schleife die Ausgabe des Anzeigenamens und der E-Mail-Adresse sowie des vollständigen ADS-Pfads des einzelnen Eintrags.

ADSI-Queries

ADO/ADSI-Anfragen Der ADSI-Provider für LDAP unterstützt auch ADSI/ADO-Queries, mit denen die Menge der zurückgelieferten Einträge eingeschränkt werden kann. Ein Beispiel für eine Query auf einem Exchange Server finden Sie in Kapitel 13.5.7.

13.9.6 Erzeugung neuer Empfängereinträge

Neue Einträge Die Erzeugung neuer Einträge im Exchange-Verzeichnis hält im Vergleich zur Erzeugung neuer Einträge mit dem WinNT-ADSI-Provider einige Herausforderungen bereit. Der Exchange Server verlangt Attributeintragungen, die das Exchange-Administratorprogramm implizit vornimmt, die bei der manuellen Programmierung aber explizit erfolgen müssen. Bei der Erstellung von Postfächern kommt ADSI 2.5 nicht mit eigenen Mitteln aus, sondern benötigt die Hilfe von Zusatzkomponenten.



Es ist ratsam, im Raw-Modus die Attributwerte von per ADMIN.EXE angelegten Empfängern genau zu betrachten. Daran erkennen Sie, welche Attribute ein Exchange-Empfänger auf Ihrem Server benötigt.

Erzeugung neuer Postfächer

Herausforderung Sicherheits-einstellungen Ein besonderes Hindernis sind bei der Erzeugung neuer Postfächer die Einträge für Zugriffsrechte bei Exchange-Postfächern:

- Für die Belegung der Eigenschaft *Primäres Windows NT-Konto* (Attribut *Assoc-NT-Account*) wird der *Security Identifier (SID)* eines NT-Benutzerkontos benötigt. Eine entsprechende Methode steht erst ab ADSI-Version 2.5 zur Verfügung und verlangt auch dort eine Umwandlungsroutine.

- Für die Berechtigungseinstellungen (Attribut NT-Security-Descriptor) wird eine Access Control List (ACL) benötigt, die ADSI auch in Version 2.5 nicht an ein Exchange-Verzeichnisobjekt zuweisen kann.

Die Realisierung mit API-Aufrufen ist grundsätzlich möglich, aber für das Scripting nicht verwendbar. Nachdem Microsoft Probleme lange Zeit einfach ignoriert hat und die Entwicklergemeinde sich mit eigenen Komponenten beholfen hat, die die entsprechenden API-Funktionsaufrufe kapselten, bietet der Hersteller des Exchange Servers inzwischen direkt zwei alternative Zusatzkomponenten:

- Die *ADsSecurity-Komponente* aus dem ADSI Resource Kit
- Die *AcctCrt-Komponente* aus dem Exchange Server SDK

Beide Komponenten werden in diesem Buch behandelt. *ADsSecurity* wird in Kapitel 13.11 ausführlich besprochen. Die *AcctCrt-Komponente* vereinfacht das komplette Handling von NT-Konten in Zusammenhang mit Exchange-Mailboxen: Konten können erstellt und gelöscht werden, der SID eines Kontos kann ermittelt und ein einfacher Security Descriptor erzeugt werden. Einfach heißt in diesem Zusammenhang allerdings, dass nur Security Descriptors erzeugt werden können, wenn es genau einen Postfachbesitzer gibt. Weitere Benutzer und andere Rollen (z.B. SENDEN ALS, EMPFANGEN ALS) sind nicht möglich.

AcctCrt

Dagegen fokussiert die *ADsSecurity-Komponente* allein auf die Ermittlung von SIDs und die Erstellung von komplexen Security Descriptors. Die in dieser Komponente implementierten Klassen bilden die komplexe Realität ab, in der eine *Access Control List (ACL)* aus mehreren *Access Control Entries (ACE)* besteht, wobei jedem ACE ein Benutzer und bestimmte Rechte zugewiesen sind. ACL und ACE werden mit den Standard-ADSI-Klassen erzeugt. *ADsSecurity* erledigt lediglich das Auslesen und Zuweisen einer ACL an das *SecurityDescriptor*-Attribut einer Mailbox. Die *ADsSecurity-Komponente* bietet die größere Flexibilität und soll hier vorgestellt werden.

ADsSecurity

Beispiel (Teil 1)

Das nachfolgende Skript zeigt zunächst den einfacheren Teil: das Erzeugen des neuen Eintrags und das Setzen der wichtigsten Attribute. Voraussetzung ist eine korrekte Belegung aller Attributwerte.

**Erster Teil
des Beispiels**

```
' --- Environment NT-Domain
strNTdomain = "Sonnensystem"
' --- Environment Exchange-Server
StrServerName = "Sonne"
strO = "Universum"
StrOU = "Sonnensystem"
strMTA = "cn=Microsoft MTA,cn=" + StrServerName +
",cn=Servers,cn=Configuration,ou=" + StrOU + ",o=" + strO
strMDB = "cn=Microsoft Private MDB,cn=" + StrServerName +
",cn=Servers,cn=Configuration,ou=" + StrOU + ",o=" + strO
```

Active Directory Service Interface (ADSI)

```
strExadmin_user = "cn=Administrator, cn=SONNENSYSYSTEM"
strExadmin_pwd = "ds9"
strContainer = "LDAP://" + StrServerName + "/cn=Recipients,ou=" + _
StrOU + ",o=" + strO
' --- Angaben zur Mailbox
strDisplayName = "Joerg Bolender"
strFirstname = "Joerg"
strLastName = "Bolender"
strAlias = "JBolender"
strSMTPMail = "Joerg@Bolender.net"
strDirectoryName = strAlias
strCountry = "US"
strTel = "0815/4711"
' -- Angaben zum zugehörigen NT-Konto
strNTUserName = "JBolender"
strNTUserPassword = "MudiaArt"
' ----- NT-Benutzer anlegen
Set DOM = GetObject("WinNT://" & strNTdomain)
Set usr = DOM.Create("user", strNTUserName)
usr.SetInfo
'usr.SetPassword strNTUserPassword
' ----- Mailbox anlegen
' --- Container öffnen
Set objNS = GetObject("LDAP:")
Set objContainer = objNS.OpenDSObject(strContainer, strExadmin_user, _
strExadmin_pwd, 0)
' --- Mailbox erzeugen mit Create
Set objRecipient = objContainer.Create("organizationalPerson", "cn=" + _
CStr(strDirectoryName))
' --- Individuelle Attribute setzen
objRecipient.Put "givenName", CStr(strFirstname)
objRecipient.Put "sn", CStr(strLastName)
objRecipient.Put "cn", CStr(strDisplayName)
objRecipient.Put "uid", CStr(strAlias)
objRecipient.Put "mail", CStr(strSMTPMail)
objRecipient.Put "TelephoneNumber", CStr(strTel)

objRecipient.Put "rfc822Mailbox", CStr(strSMTPMail)
objRecipient.Put "textEncodedORaddress", CStr("c=" & strCountry & ";a="
;p=" + strO + ";o=" + StrOU + ";s=" + strAlias + ";")
' --- Sonstige notwendige Attribute
objRecipient.Put "Home-MTA", CStr(strMTA)
objRecipient.Put "Home-MDB", CStr(strMDB)
objRecipient.Put "mailPreferenceOption", 0
objRecipient.Put "MDB-Use-Defaults", True
objRecipient.Put "MAPI-Recipient", True
objRecipient.Put "Replication-Sensitivity", 20
```

```
objRecipient.SetInfo  
say "Mailbox " & strAlias & " wurde angelegt..."
```

Listing 13.48: Anlegen eines Exchange-Postfachs [ADSIEX_NeuesPostfach.wsf]

Das Attribut `mail` ist die Standardadresse des Benutzers. Weitere E-Mail-Adressen werden über das mehrwertige Attribut `otherMailbox` zugewiesen.



Sicherheitseinstellungen (Teil 2 des Beispiels)

Ein so angelegtes Postfach kann jedoch noch nicht benutzt werden. Die beiden folgenden Listings zeigen, wie mit der Komponente *ADsSecurity* die Postfachrechte gesetzt werden können. Zunächst muss der SID des NT-Benutzers mit Hilfe der Klasse *ADsSID* an das Attribut `Assoc-NT-Account` zugewiesen werden. Dann müssen die Berechtigungen mit der Klasse *ADsSecurity* um einen Eintrag für den primären NT-Benutzer erweitert werden.

Zweiter Teil des Beispiels

```
' -- --- Setzen des Primären Windows NT-Kontos  
Dim objSid ' -- As New ADsSID  
Set objSid = CreateObject("ADsSID")  
Const ADS_SID_HEXSTRING = 1  
Const ADS_SID_WINNT_PATH = 5  
objSid.SetAs ADS_SID_WINNT_PATH, "WinNT://" & strNTdomain & "/" & _  
strNTUserName & ",user"  
sidHex = objSid.GetAs(ADS_SID_HEXSTRING)  
objRecipient.Put "Assoc-NT-Account", sidHex  
objRecipient.SetInfo
```

Listing 13.49: Zuweisen der Sicherheitseinstellung an ein neues Postfach [ADSIEX_NeuesPostfach.wsf]

```
' -- --- Setzen der Berechtigungen  
Dim objSD ' -- As IADsSecurityDescriptor  
Dim objACL ' -- As IADsAccessControlList  
Dim objACE ' -- As ActiveDs.AccessControlEntry  
Dim objSec ' -- As ADsSecurity  
Set objSec = CreateObject("ADsSecurity")  
Set objACE = CreateObject("AccessControlEntry")  
Const ADS_RIGHT_EXCH_MODIFY_USER_ATT = 2  
Const ADS_RIGHT_EXCH_MAIL_SEND_AS = 8  
Const ADS_RIGHT_EXCH_MAIL_RECEIVE_AS = 16  
' -- --- SecDescriptor auslesen  
Set objSD = objSec.GetSecurityDescriptor(objRecipient.ADsPath)  
' -- --- in ACL umwandeln  
Set objACL = objSD.DiscretionaryAcl  
' -- --- Eintrag erzeugen  
objACE.Trustee = strNTdomain & "\" & strNTUserName  
objACE.AccessMask = ADS_RIGHT_EXCH_MODIFY_USER_ATT Or
```

Active Directory Service Interface (ADSI)

```
ADS_RIGHT_EXCH_MAIL_SEND_AS Or ADS_RIGHT_EXCH_MAIL_RECEIVE_AS
objACE.AceType = ADS_ACETYPE_ACCESS_ALLOWED
' -- --- Eintrag anfügen an ACL
objACL.AddAce objACE
' -- --- SecDescriptor aus ACL erzeugen
objSD.DiscretionaryAcl = objACL
' -- --- Neuen SecDescriptor zuweisen
objSec.SetSecurityDescriptor objSD
```

*Listing 13.50: Zuweisen der Berechtigungen an ein neues Postfach
[ADSIEX_NeuesPostfach.vbs]*

Postfach löschen

Mit Hilfe der IADsContainer-Methode Delete() kann ein Eintrag im Exchange-Verzeichnis schnell gelöscht werden.

13.9.7 Erzeugung von E-Mail-Verteilern

GroupOfNames verwalten

Die Erzeugung eines E-Mail-Verteilers und das Füllen mit Empfängern ist einfach. Zuerst ist eine Instanz der Klasse groupOfNames zu erzeugen. Dabei sollten die Attribute cn (Anzeigename), uid (Alias) und mail (SMTP-E-Mail-Adresse) gesetzt werden. Nach dem Schreiben des Property Caches mit SetInfo() kann der Methode objDL.Add(adspath) ein Empfänger der Liste hinzugefügt werden, wobei objDL der Zeiger auf das Verteilerobjekt und adspath der ADSI-Pfad des anzufügenden Empfängers ist.

ADSIEX_ NeueListe()

Die nachfolgend abgedruckte Hilfsroutine ADSIEX_NeueListe() erzeugt einen Verteiler auf Basis einer ADSI-Query und ermöglicht so auf einfache Weise, eine Menge von Empfängern, die ein bestimmtes Kriterium erfüllen, zu einem Verteiler zusammenzufassen.

```
Sub ADSIEX_NeueListe(vtcontainer, vname, vtemail, querycontainer, _
queryselect)
Dim objCON ' As IADsContainer
Dim objDL ' As IADs
Dim rs ' As ADODB.Recordset
Dim Query ' As String
' -- Neuen Verteiler anlegen
Set objCON = GetObject(vtcontainer)
Set oDL = objCON.Create("groupOfNames", "cn=" & vname)
oDL.Put "cn", CStr(vname) ' DisplayName
oDL.Put "uid", CStr(vname) ' Alias
oDL.Put "mail", CStr(vtemail) ' SMTP-Adresse
oDL.SetInfo
' -- Verbindung zu dem OLE DB-Provider für ADSI
Set objCON = CreateObject("ADODB.Connection")
CONSTRING = "Provider=ADSDSOObject"
objCON.Open CONSTRING, "administrator", ""
' -- Abfrage ausführen
```


Exchange Server-Administration mit ADSI

```
Query = "<" & querycontainer & ">"; & queryselect & _  
";adspath,cn,mail;subtree"  
Set rs = objCON.Execute(Query)  
' --- Iteration über alle: In den Verteiler aufnehmen  
While Not rs.EOF  
    oDL.Add rs("adspath").Value  
    say rs("adspath").Value  
    rs.MoveNext  
Wend  
rs.Close  
End Sub
```

Listing 13.51: Anlegen eines neuen E-Mail-Verteilers auf Basis einer ADO/ADSI-Query

```
vtcontainer = "LDAP://kom.wi-inf.uni-essen.de/  
cn=test,ou=pik,o=unighessenfb5"  
vtname = "Alle_Postfachinhaber"  
vtemail = "alle@kom.wi-inf.uni-essen.de"  
querycontainer = "LDAP://kom.wi-inf.uni-essen.de/  
cn=recipients,ou=pik,o=unighessenfb5"  
queryselect = "(&(objectclass=organizationalperson)(cn=*))"  
ADSIEX_NeueListe vtcontainer, vtname, vtemail, querycontainer, _  
queryselect
```

Listing 13.52: Anlegen eines neuen Verteilers für alle Postfächer unterhalb eines bestimmten Containers [ADSIEX_testNeueListe.wsf]

Andere Operationen

Ein Empfänger kann über `objDL.Remove(adspath)` aus einem Verteiler entfernt werden. Ein Verteiler wird – wie andere Verzeichniseinträge auch – über ein `Delete()` auf dem übergeordneten Container gelöscht.

**Empfänger
entfernen**

13.9.8 Zugriff auf Konfigurationsinformationen

Auch Konfigurationsinformationen des Exchange Servers können ausgelesen und verändert werden.

```
Sub ADSIEX_ConfigContainer()  
' -- Zugriff auf Konfigurations-Container  
Set container = _  
GetObject("LDAP://Sonne/cn=Configuration,ou=ESSEN, o=IT-Visions")  
' -- Alle Einträge ausgeben  
For Each c In container  
    say c.Class, c.adspath  
Next  
Set container = Nothing  
End Sub
```

Listing 13.53: Auslesen der Konfigurationsobjekte in einem Exchange Server [ADSIEX_ConfigContainer.wsf]

13.10 Internet Information Server-Administration mit ADSI

IIS Metabase Auch die Konfiguration des Internet Information Servers (IIS) ist in einem Verzeichnisdienst, der *IIS Metabase* (in der deutschen Dokumentation Metabasis genannt), abgespeichert. Es gibt einen ADSI-Provider für IIS ab Version 4.0. Microsoft nennt die ADSI-Objekte in der Dokumentation auch *IISAdmin-Objekte*. Die im IIS Resource Kit mitgelieferten WSH-Skripte beruhen auf ADSI. Im Gegensatz zu anderen ADSI-Providern enthält die ADSIIS.DLL auch eine Typbibliothek. Diese fällt jedoch sehr spärlich aus und enthält nur einige wenige Klassendefinitionen. Die Ausführungen in diesem Kapitel gelten sowohl für den IIS 4.0 als auch für den IIS 5.0 und den IIS 6.0.

Tabelle 13.37:
Schnellinfo ADSI-
Provider für IIS

Name und Abkürzung	IISAdmin-Objekte (ADSI-Provider für IIS)
DLL	adsiiis.dll
Komponentenname	IISOLE
ProgID	Keine. Die Klassen sind nur über ADSI-Moniker nutzbar.
Helpstring	Active DS IIS Namespace Provider
Hersteller	Microsoft
Besprochene Version	6.0.2462.1
NT4	Installation Version 4.0 durch Installation des Internet Information Server 4.0 bzw. Client Tools
Windows2000	Version 5.0 enthalten, wenn IIS durch Setup installiert wurde
Whistler (Windows XP / Windows.NET)	Version 6.0 enthalten, wenn IIS durch Setup installiert wurde
Dokumentation im MSDN	Plattform SDK Web Services Internet Information Server SDK Advanced Programmatic Administration Using IIS Admin Base Objects MSDN: IISRef.chm::/asp/abgu16b8.htm]

13.10.1 IIS-Objektmodell

Schema der IIS-Metabase Die Objekthierarchie des IIS ist komplex, hier soll daher nur ein Ausschnitt betrachtet werden. Jedes *IISComputer*-Objekt enthält jeweils genau ein Unterobjekt für die verschiedenen Dienste:

- ▶ ein Objekt mit Namen *W3Svc* aus der Klasse *IISWebService*
- ▶ ein Objekt mit Namen *MSFTPSvc* aus der Klasse *IISFTPService*
- ▶ ein Objekt mit Namen *NNTPSvc* aus der Klasse *IISNntp*
- ▶ ein Objekt mit Namen *SmtPsvc* aus der Klasse *SmtPService*

Das W3Svc-Objekt enthält jeweils genau ein Objekt der Klasse IISWebInfo (Name Info) und IISFilters (Name Filters) sowie beliebig viele Objekte der Klasse IISWebServer. Das MSFTPSvc-Objekt enthält genau ein Objekt der Klasse IISFTPService (Name Info) und beliebig viele Objekte der Klasse IISFTPService.

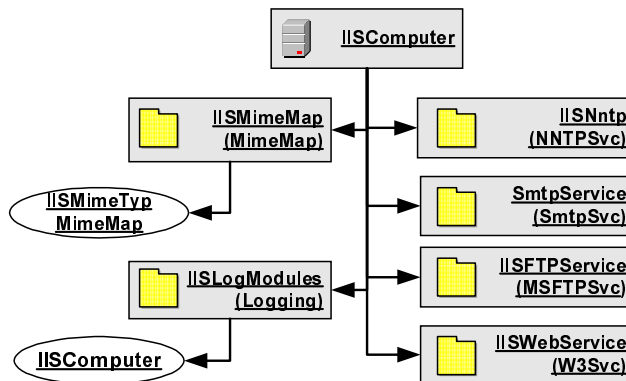


Abbildung 13.25:
Oberste Ebene des
IIS-Metabase-
Objektmodells

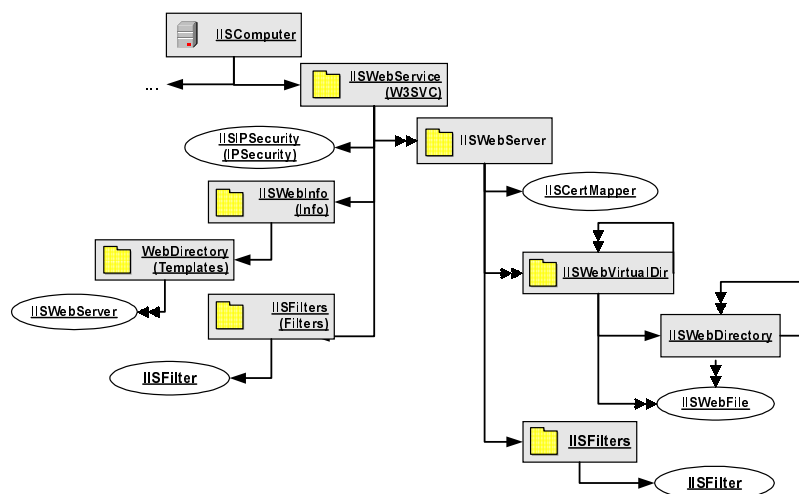


Abbildung 13.26:
Der W3Svc-Zweig
des IIS-Metabase-
Objektmodells

Im IIS-Schema gibt es einen Fehler: So darf ein IISWebService-Objekt laut Schema nur Unterobjekte vom Typ IISWebInfo und IISWebServer sowie vom allgemeinen Typ IISObject enthalten. Tatsächlich enthält ein IISWebService-Objekt aber auch immer ein Objekt vom Typ IISFilters.



Erweiterbarkeit Das Schema der IIS-Metabase ist erweiterbar. Sie können eigene Klassen mit Attributen anlegen. Informationen dazu finden Sie in der MSDN Library [MSL00]

13.10.2 Identifikation und Bindung

ADSI-Pfade im IIS Der Webservice wird über den ADSI-Pfad IIS://COMPUTERNAME/W3SVC erreicht. Die einzelnen Webserver werden dann leider nicht über den Namen adressiert, der in der Objekthierarchie der Management Console erscheint, sondern über eine fortlaufende interne Nummer (*Server-ID*), z.B. IIS://COMPUTERNAME/W3SVC/2. IIS-Administratoren bekommen es mit dieser unglücklichen Identifizierung zu tun, wenn sie das Logfile eines virtuellen Webserverns suchen müssen. Der Zugriff erfolgt wie üblich mit `GetObject()`.

```
Set objvirtweb = GetObject("IIS://Sonne2000/w3svc/1")
say objvirtweb.servercomment
```

Listing 13.54: Ausgabe des Namens des ersten virtuellen Webserverns

**Keine Klassen-
definitionen**

In typisierten Umgebungen können Objektvariablen leider nur allgemein als IADs deklariert werden; eine Typbibliothek mit dezidierten Klassendefinitionen steht nicht zur Verfügung.

```
Dim objWWWService As IADs
Dim objVirtWeb As IADs
```

13.10.3 Arbeit mit virtuellen Webservern

Die Beispiele in diesem Buch beschränken sich auf die automatisierte Verwaltung von Webservern. Mit diesem Wissen werden Sie in der Lage sein, sich in die Administration der anderen Dienste des IIS (FTP, SMTP und NNTP) schnell einzuarbeiten.

Liste der konfigurierten virtuellen Webserver

**Virtuelle
Webserver
auflisten**

Das nachfolgende Listing zeigt, wie alle virtuellen Webserver aufgelistet werden können, indem mit `For...Each` der `W3Svc`-Container durchlaufen wird. Da dort auch andere Objekte enthalten sein können, wird der Klassenname auf `IISWebServer` überprüft.

```
Dim objvirtweb ' As IADs
Dim objWWWService ' as IADS
Dim state(7) ' As String ' Serverzustände
Dim Port ' As String
Dim Binding 'As String
Dim IP ' As String
' -- Konstantendefinition für Serverzustand
Const MD_SERVER_STATE_STARTING = 1 'starting
```

Internet Information Server-Administration mit ADSI

```

Const MD_SERVER_STATE_STARTED = 2 'Server started
Const MD_SERVER_STATE_STOPPING = 3 'Server stopping
Const MD_SERVER_STATE_STOPPED = 4 'Server stopped
Const MD_SERVER_STATE_PAUSING = 5 'Server pausing
Const MD_SERVER_STATE_PAUSED = 6 'Server paused
Const MD_SERVER_STATE_CONTINUING = 7 'Server continuing
state(MD_SERVER_STATE_STARTING) = "starting"
state(MD_SERVER_STATE_STARTED) = "started"
state(MD_SERVER_STATE_STOPPING) = "stopping"
state(MD_SERVER_STATE_STOPPED) = "stopped"
state(MD_SERVER_STATE_PAUSING) = "pausing"
state(MD_SERVER_STATE_PAUSED) = "paused"
state(MD_SERVER_STATE_CONTINUING) = "continuing"
' -- Bindung an Webservice
Set objWWWService = GetObject("IIS://Sonne2000/w3svc")
' -- Iteration über alle virtuellen Webserver
For Each objvirtweb In objWWWService
    If objvirtweb.Class = "IISWebServer" Then
        ' -- Binding-String IP:Port:Hostname auflösen
        Binding = objvirtweb.ServerBindings(0)(0)
        IP = Left(Binding, InStr(Binding, ":") - 1)
        Binding = Right(Binding, Len(Binding) - InStr(Binding, ":"))
        Port = Left(Binding, InStr(Binding, ":") - 1)
        ' -- Ausgabe
        say objvirtweb.name & ":" & objvirtweb.servercomment & ":" & _
state(objvirtweb.ServerState) & ":" & IP & ":" & Port
    End If
Next

```

Listing 13.55: Liste aller virtuellen Webserver eines IIS [ADSIIS_liste.wsfl]

Zustände eines Webservers

Ein virtueller Webserver kann sieben verschiedene Zustände einnehmen, die als Zahlenwerte zurückgeliefert werden (vgl. folgende Tabelle). Im obigen Listing erfolgt die Umsetzung in einen Text mit Hilfe eines String-Arrays (state).

Zustände

Symbolische Konstante	Wert
MD_SERVER_STATE_STARTING	1
MD_SERVER_STATE_STARTED	2
MD_SERVER_STATE_STOPPING	3
MD_SERVER_STATE_STOPPED	4
MD_SERVER_STATE_PAUSING	5
MD_SERVER_STATE_PAUSED	6
MD_SERVER_STATE_CONTINUING	7

Tabelle 13.38:
Mögliche Zustände
eines virtuellen
Webservers im IIS

Webserver anlegen

Neuer virtueller Webserver

Das Anlegen eines neuen Webservers ist etwas aufwendiger als der Achtzeiler beim Anlegen eines neuen NT-Benutzers. Zunächst muss die nächsthöhere verfügbare eindeutige Server-ID durch eine Iteration über alle bestehenden virtuellen Webserver ermittelt werden. Danach kann mit einem `Create()` innerhalb des übergeordneten `WebService-Containers` ein neuer Webserver angelegt werden. Das Attribut `ServerComment` nimmt den Anzeigenamen des Webservers auf, der in der Microsoft Management Console (MMC) erscheint. `ServerBinding` erwartet die IP-Adresse nebst Portnummer, der der Webserver zugeordnet werden soll.

```
' -- Bindung an Webservice
Set objWWWService = GetObject("IIS://Sonne2000/w3svc")
For Each objVirtWeb In objWWWService
    If objVirtWeb.Class = "IISWebServer" Then
        max = objVirtWeb.Name
    End If
Next
Name = max + 1
Set objNewWeb = objWWWService.Create("IISWebServer", Name)
objNewWeb.servercomment = ServerName
objNewWeb.KeyType = "IISWebServer"
Binding(0) = IP & ":" & Port & ":"
objNewWeb.ServerBindings = Binding
objNewWeb.SetInfo
```

Listing 13.56: Anlegen eines virtuellen Webservers: Es fehlt allerdings die Erzeugung des Root-Verzeichnisses. [ADSIIS_neuesWeb.wsf]

Virtuellen Pfad hinzufügen

Auf diesem Stand ist der Webserver aber weder funktionsfähig noch in der MMC administrierbar. Es fehlt die Zuordnung zu einem Verzeichnis auf der Festplatte. Jedes `WebServer-Objekt` muss mindestens ein Unterobjekt vom Typ `IISWebVirtualDir` mit dem Namen `Root` enthalten.

```
Set objRootDir = objNewWeb.Create("IISWebVirtualDir", "ROOT")
objRootDir.Path = RootDir
objRootDir.AccessScript = True
objRootDir.SetInfo
```

Listing 13.57: Erzeugung eines Root-Verzeichnisses für den virtuellen Webserver [ADSIIS_neuesWeb.wsf]

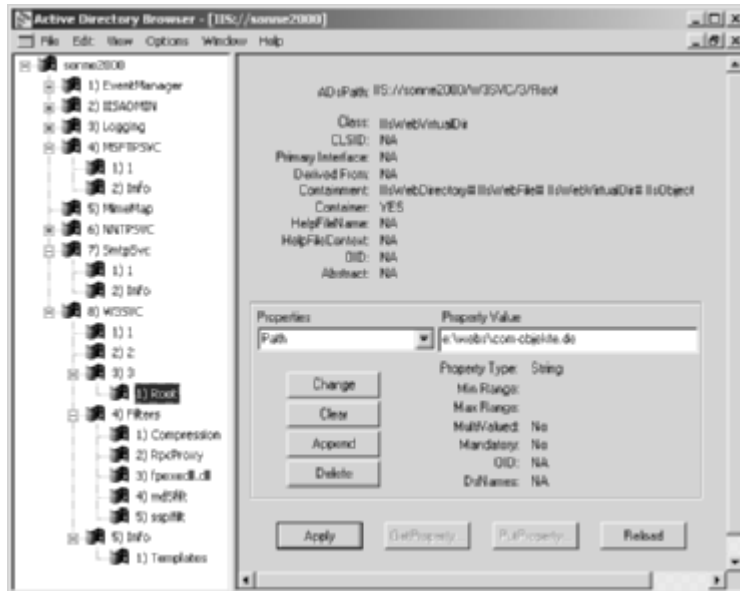


Abbildung 13.27:
Der Pfad für das
Root-Verzeichnis des
Webserver mit der
ID »3« im Active
Directory Browser

Illegales wird möglich

Wird das Anlegen des Root-Verzeichnisses vergessen oder schlägt es fehl, erscheint in der Management Console zwar ein neuer Eintrag für den angelegten virtuellen Webserver, es ist dort jedoch weder möglich, seine Eigenschaft zu verändern, noch ihn wieder zu löschen. Der ungültige Webservereintrag muss wieder per ADSI gelöscht werden.



Löschen und Zustandsänderungen

Das Entfernen eines virtuellen Webserver ist in einer Zeile erledigt:

```
objWWWService.Delete "IISWebVirtualDir", objWebSvr.Name
```

Delete()

Auch die Zustandsänderungen sind Einzeiler: `objVirtWeb.Start()`, `objVirtWeb.Stop()`, `objVirtWeb.Pause()` oder `objVirtWeb.Continue()`.

Zustands-
änderungen

Bei `Pause()` und `Stop()` sollte man vorher prüfen, ob der Webserver wirklich läuft. Wenn es nämlich nichts anzuhalten gibt, weil der Webserver schon inaktiv ist, dann braucht ADSI unerträglich lange, bis die Fehlermeldung erscheint.

Werkzeug SimpleIISAdmin

Auf der Buch-CD befindet sich der *SimpleIISAdmin*, ein einfaches Werkzeug, um virtuelle Webserver zu starten, zu stoppen, anzulegen und zu löschen (siehe folgende Abbildung). Der SimpleIISAdmin ist in Visual Basic 6.0

SimpleIISAdmin

Active Directory Service Interface (ADSI)

geschrieben und liegt im kompletten Quelltext auf der CD vor [CD:/code/komponenten /ADSI_IIS/SimpleIISAdmin].

Abbildung 13.28:
SimpleIISAdmin



Info-Objekt

Properties des WWW-Dienstes

Eine besondere Stellung nimmt das Info-Objekt ein. Dieses Objekt der Klasse IISWebInfo enthält zusätzliche Attribute des W3Svc-Objekts und ist gleichzeitig Container für die Master Properties des WWW-Dienstes. Diese Master Properties können im Eigenschaftsfenster des COMPUTER-Astes in der Management Console eingestellt und an alle untergeordneten Webserver vererbt werden. Die IIS Metabase enthält dazu unterhalb von IISWebInfo ein Objekt mit dem Namen Templates (Klasse IISWebDirectory), darunter wiederum liegen Objekte mit den Namen PublicWebSite und SecureWebSite. Darin ist jeweils ein Objekt Root vom Typ IISWebDirectory enthalten, das die Standardvorgaben für das Root-Verzeichnis enthält. Gleiches gilt für das Info-Objekt unterhalb des MSFTPSvc-Objekts.



Verwechseln Sie die Master Properties nicht mit der Default-FTP bzw. WWW-Site.

13.11 Zusatzkomponenten für ADSI

Es gibt inzwischen sechs Zusatzkomponenten von Microsoft, die die Funktionalität von ADSI erweitern.

ADSI Resource Kit

- Zum einen sind dies fünf Komponenten aus dem ADSI Resource Kit, das wiederum Teil des ADSI SDKs ist. Dies sind sehr kleine Komponenten, die nur aus einer einzigen Klasse bestehen.

- Zum anderen ist es die sehr mächtige Komponente *AdsiTools* (Teil der Windows2000 Support Tools).

AdsiTools

Alle sechs Komponenten sind bislang nicht in der MSDN Library dokumentiert. Das ADSI SDK enthält jedoch Beispiele und eine Datei RTK.HTM mit kurzen Erläuterungen.

13.11.1 ADSI-Versionsabfrage mit ADsVersion

Die Komponente ADsVERSION mit genau einer Klasse gleichen Namens liefert die Versionsnummer des installierten Active Directory Service Interface (ADSI). Auch die Abfrage entfernter Systeme ist möglich. ADsVERSION stellt fünf für sich selbst sprechende Methoden bereit `GetVersion()`, `GetMajorVersion()`, `GetMinorVersion()`, `GetLocale()` und `Connect("Computername")`.

**ADSI-Version
ermitteln**

Name und Abkürzung	ADsVersion-Komponente
Name der Komponentendatei	ADsVersion.dll
Interner Name der Typbibliothek	ADsVersion
Abweichender Komponentenname in der Registry	Die ProgID besteht abweichend von der üblichen Schreibweise Komponente. Klasse nur aus dem Begriff ADsVersion.
Helpstring der Typbibliothek	ADsVersion 1.0 Type Library
Hersteller	Microsoft
Lizenzierung	Kostenloses Add-on; die Komponente ist im ADSI SDK 2.5 enthalten
Besprochene Version	1.0.0.1
Alle Windows-Versionen	Nicht enthalten, Installation durch <code>regsvr32 ADSVERSION.DLL</code>
Dokumentation	Kurzbeschreibung in der Datei RTK.HTM im ADSI SDK

Tabelle 13.39:
Schnellinfo
ADsVersion-
Komponente

```
Set vs = CreateObject("ADsVersion")
' -- Verbindungsaufbau
vs.Connect "\\sonne2000"
' -- Versionsausgabe
say "Genaue Version: " & vs.GetVersion
say "Sprache: " & vs.GetLocale
```

Listing 13.58: Ermittlung der ADSI-Version auf einem entfernten System. Ohne die Methode `Connect()` arbeitet die Routine lokal. [ADSIEXTRAS_version.wsf]

Wichtig: ADsVersion liefert die internen Versionsnummern, die mit den DLL-Versionsnummern übereinstimmen. ADSI 2.5 unter Windows2000 ist intern Version 5.0.



13.11.2 Detaillierte Fehlermeldungen mit ADsError

ADSI-Fehlerinformationen

Die Klasse `ADsError` stellt nur eine Methode bereit. `GetErrorMessage(1ngFehlernummer)` liefert die Beschreibung zu einer gegebenen Fehlernummer. Diese Beschreibung ist meistens ausführlicher als die Beschreibung, die Visual Basic in seinem `Err`-Objekt speichert.

Tabelle 13.40:
Schnellinfo
ADsError-Komponente

Name und Abkürzung	ADsError-Komponente
Name der Komponentendatei	ADsError.DLL
Interner Name der Typbibliothek	ADsErrorLib
Abweichender Komponentenname in der Registry	Die ProgID besteht abweichend von der üblichen Schreibweise Komponente.Klasse nur aus dem Begriff ADsError.
Helpstring der Typbibliothek	ADsError 1.0 Type Library
Hersteller	Microsoft
Lizenzierung	Kostenloses Add-on; die Komponente ist im ADSI SDK 2.5 enthalten
Besprochene Version	1.0.0.1
Alle Windows-Versionen	Nicht enthalten, Installation durch <code>regsvr32 ADsError.DLL</code>
Dokumentation	Kurzbeschreibung in der Datei <code>RTK.HTM</code> im ADSI SDK

```
Set adsErr = CreateObject("ADsError")
On Error Resume Next
Set rec = GetObject("LDAP://DC=nichtda, DC=nirgendwo")
' -- Fehlerbehandlung

If Err <> 0 Then
    say "Fehler: " & Err.Description
    say "ADSI-Fehler: " & adsErr.GetErrorMessage(Err.Number)
End If
```

Listing 13.59: Verwendung von `ADsError` [ADSIEXTRAS_error.wsf]

Die Ausgabe sieht dann etwa so aus:

```
VB-Fehler : Automatisierungsfehler
Eine Referenzauswertung wurde vom Server zurückgesendet.
ADSI-Fehler: Eine Referenzauswertung wurde vom Server zurückgesendet.
-- Extended Error:LDAP Provider : 0000202B: RefErr: DSID-031006A4,
data 0, 1 access points
ref 1: 'nichtda.nirgendwo'
```

Listing 13.60: Ausgabe zu obigem Script

13.11.3 GetObject() durch die Hintertür mit ADsFactory

Die Komponente *ADsFactory* stellt die Klasse *ADsFactory* bereit, die den Zugriff auf Objekte via Moniker aus Umgebungen ermöglicht, die zwar die Instanziierung von Komponenten mit `CreateObject()`, nicht aber den Zugriff auf bestehende Instanzen mit `GetObject()` unterstützen. Eine solche Umgebung ist z.B. der Internet Explorer. Umgebungen, die `GetObject()` verwenden können, brauchen die *ADsFactory-Komponente* nicht. *ADsFactory* kapselt die Funktionalität von `GetObject()` in einer Klasse, die mit `CreateObject()` erzeugt werden kann.

**Umgehung der
GetObject()-
Blockade**

Name und Abkürzung	ADsFactory
Name der Komponentendatei	ADSFACTR.DLL
Interner Name der Typbibliothek	ADSFACTRLib
Abweichender Komponentenname in der Registry	Die ProgID besteht abweichend von der üblichen Schreibweise Komponente.Klasse nur aus dem Begriff ADsFactory.
Helpstring der Typbibliothek	ADSRAS 1.0 Type Library
Hersteller	Microsoft
Lizenzierung	Kostenloses Add-on; die Komponente ist im ADSI SDK 2.5 enthalten
Besprochene Version	1.0.0.1
Alle Windows-Versionen	Nicht enthalten, Installation durch regsvr32 ADSFACTR.DLL
Dokumentation	Kurzbeschreibung in der Datei RTK.HTM im ADSI SDK

Tabelle 13.41:
Schnellinfo
ADsFactory

ADsFactory stellt nur zwei Methoden bereit, die jeweils einen Objektzeiger zurückliefern:

- Function `GetObject(strADsPath) As Object`
- Function `OpenDSObject(strADsPath, strUser, bstrPassword, lFlag) As Object`

Methoden

Die beiden Methoden entsprechen dem aus ADSI bekannten Vorgehen zur Erzeugung eines Zeigers auf ein ADSI-Objekt. Interessant ist, dass `ADsFactory.GetObject()` nicht nur für ADSI-Verzeichniseinträge, sondern auch für jedes andere persistente COM-Objekt verwendet werden kann. Das folgende Listing zeigt die Verwendung in einer HTML-Seite: Das Ergebnis einer Verzeichnisabfrage wird in einer vorhandenen Word-Datei protokolliert.

Anwendungsgebiete

```
Set oFact = CreateObject("ADsFactory")
' --- msgbox "Beispiel 1: Word"
Set w = oFact.GetObject("d:\buch\vertrag.doc")
```

Active Directory Service Interface (ADSI)

```
w.application.visible = True msgbox "Word wartet..."
w.application.quit
' --- msgbox "Beispiel 2: ADSI"
Set oCont = oFact.GetObject("WinNT://" & domainName )
say "Container name = " & oCont.Name
say "Container path = " & oCont.ADsPath
for each obj in oCont document.write "Object name = " & obj.Name
say "Object path = " & obj.ADsPath
Next
```

Listing 13.61: Einsatz von *ADsFactory* [*_ADSIEXTRA_factory.htm*]



Bitte beachten Sie: `GetObject()` ist nicht ohne Grund gesperrt. Mit der Installation von *ADsFactory* verringern Sie die Sicherheit des Internet Explorers.

13.11.4 RAS-Konfiguration mit ADsRAS

Namespace Extension

Die *ADsRAS* ermöglicht den Zugriff auf die *Remote Access Service (RAS)*-Konfiguration eines Benutzers über den WinNT-Provider. Im Gegensatz zu den anderen Komponenten des ADSI Resource Kits, die eigenständig instanziiierbare Klassen anbieten, ist *ADsRas* eine so genannte *ADSI Namespace Extension*, die die Fähigkeiten des WinNT-Providers erweitert. ADSI Namespace Extensions sind in der Registry unterhalb der Einträge der einzelnen Namespaces verzeichnet. *ADsRas* ist registriert unter `HKEY_LOCAL_MACHINE\SOFTWARE\MICROSOFT\ADS\PROVIDERS\WINNT\EXTENSIONS\USER\{F1F533F0-F118-11D2-BC88-00C04FD430AF}`. Wie dem Schlüssel zu entnehmen ist, erweitert *ADsRAS* die Klasse *User*.

Attribute und Methoden

Die Erweiterung besteht darin, dass *ADsRas* die Klasse *User* um ein Attribut und drei Methoden ergänzt:

- ▶ Das Attribut `DialInPrivilege` enthält *True*, wenn der Benutzer sich einwählen darf. Das Attribut darf verändert werden.
- ▶ Die Methode `GetRasCallback()` ermittelt den aktuellen Zustand der Rückrufeinstellung. Es gibt drei selbsterklärende Werte (siehe Listing 13.62).
- ▶ `GetRasPhoneNumber()` liefert im Fall von `ADS_RAS_ADMIN_SETCALLBACK` die gesetzte Telefonnummer.
- ▶ Die Methode `SetRasCallback(1Callback As Long, [szPhoneNumber As String])` ermöglicht das Setzen der Rückruf-Einstellung einschließlich der Telefonnummer für den Rückruf.

```
Const ADS_RAS_ADMIN_SETCALLBACK = 2
Const ADS_RAS_CALLER_SETCALLBACK = 4
Const ADS_RAS_NOCALLBACK = 1
```

Listing 13.62: Konstanten für `GetRasCallback()` und `SetRasCallback()`

Zusatzkomponenten für ADSI

Name und Abkürzung	ADsRAS
Name der Komponentendatei	ADsRAS.dll
Interner Name der Typbibliothek	ADsRASLib
Helpstring der Typbibliothek	ADsRAS 1.0 Type Library
Abweichende ProgID	Keine ProgID, da keine Klasse der Komponente instanziiert werden kann!
Hersteller	Microsoft
Lizenzierung	Kostenloses Add-on; die Komponente ist im ADSI SDK 2.5 enthalten
Besprochene Version	1.0.0.1
NT4 , Windows2000 und Whistler (Windows XP / Windows.NET)	Nicht enthalten, Installation durch regsvr32 adsras.dll
Windows 95/98/ME	Diese Komponente kann unter Windows 95/98 nicht registriert werden.
Dokumentation	Kurzbeschreibung in der Datei RTK.HTM im ADSI SDK

*Tabelle 13.42:
Schnellinfo
ADsRAS*

RAS-Konfigurations-Statistik

Das Beispiel für den Lesezugriff auf die RAS-Konfiguration ist ein statistisches Werkzeug. Es listet nicht nur für jeden Benutzer in einem Container die RAS-Konfiguration auf, sondern zählt auch mit. Es gibt insgesamt vier Möglichkeiten für die RAS-Konfiguration.

Beispiel 1

```
Dim oUser ' As IADs
Dim oCon ' As iadscontainer
Dim Status
Dim s ' As string
Dim c_nicht, c_ohne, c_var, c_fest
Set oCon = GetObject("WinNT://MArs")
oCon.Filter = Array("User")
For Each oUser In oCon ' -- für alle Nutzer
    s = "Benutzer " & oUser.Name & ": "
    If (oUser.DialinPrivilege = False) Then
        say s & " darf sich NICHT einwählen"
        c_nicht = c_nicht + 1
    Else ' --- hat Rechte...
        Status = oUser.GetRasCallback
        If Status And ADS_RAS_NOCALLBACK Then
            say s & "Einwahl, ohne Rückruf"
            c_ohne = c_ohne + 1
        End If
        If Status And ADS_RAS_CALLER_SETCALLBACK Then
            say s & "Benutzer setzt Rückrufnummer"
            c_var = c_var + 1
        End If
    End If
End For
```

Active Directory Service Interface (ADSI)

```
End If
If Status And ADS_RAS_ADMIN_SETCALLBACK Then
    say s & "Rückruf an " & oUser.GetRasPhoneNumber
    c_fest = c_fest + 1
End If
End If
Next
say "--- Statistik:"
say "Benutzer ohne Einwahlrechte: " & c_nicht
say "Benutzer mit Einwahlrechten, ohne Rückruf: " & c_ohne
say "Benutzer mit festem Rückruf: " & c_fest
say "Benutzer mit variablem Rückruf: " & c_var
```

Listing 13.63: Statistik über RAS-Einwahl [ADSIEXTRAS_RASStatistik.wsf]

Einstellungen ändern

Beispiel 2 Im zweiten Beispiel wird der Rückruf für den Benutzer MARS\HS auf eine feste Telefonnummer gesetzt. Die anderen Möglichkeiten sind auskommentiert.

```
Set x = GetObject("WinNT://mars/hs,user")
x.DialinPrivilege = True
'x.SetRasCallBack ADS_RAS_NOCALLBACK
'x.SetRasCallBack ADS_RAS_CALLER_SETCALLBACK
x.SetRasCallBack ADS_RAS_ADMIN_SETCALLBACK, "0201/7490700"
```

Listing 13.64: RAS-Konfiguration ändern [ADSIEXTRAS_RASAendern.wsf]

13.11.5 Zugriff auf Sicherheitseinstellungen mit ADsSecurity

**ACLs für
Exchange,
Dateien, Registry**

ADsSecurity ist eine sehr hilfreiche Komponente, auf die die Entwicklergemeinde leider sehr lange warten musste. Es heißt, man wäre in Redmond lange uneins darüber gewesen, ob Dateisystemsicherheit ein Teil von FSO oder von ADSI sein sollte. Das Kern-ADSI ermöglicht die Erstellung von *Access Control Lists (ACLs)*. Die Bindung an ein Verzeichnisobjekt wird jedoch nur für das Windows2000 Active Directory unterstützt, nicht jedoch für den Exchange Server 5.5. *ADsSecurity* kann Exchange 5.5-Objekten ACLs zuweisen und mehr: *ADsSecurity* kann ACLs auch an Ordner, Dateien und Registry-Einträge zuweisen. Die Komponente funktioniert aber auch mit Active Directory.

Tabelle 13.43:
Schnellinfo
ADsSecurity-
Komponente

Name und Abkürzung	ADsSecurity-Komponente
Name der Komponentendatei	ADsSecurity.DLL
Interner Name der Typbibliothek	ADsSecurityLib
Helpstring der Typbibliothek	ADsSecurity 2.5 Type Library

Zusatzkomponenten für ADSI

Abweichende ProgID	Die ProgID besteht abweichend von der üblichen Schreibweise Komponente. Klasse nur aus den Klassennamen AdsSecurity und ADSID
Hersteller	Microsoft
Lizenzierung	Kostenloses Add-on; die Komponente ist im ADSI SDK 2.5 enthalten
Besprochene Version	1.0.0.1
NT4, Windows2000 und Whistler (Windows XP / Windows.NET)	Nicht enthalten, Installation durch regsvr32 AdsSecurity.DLL
Windows 95/98/ME	Diese Komponente kann unter Windows 95/98/ME nicht registriert werden.
Dokumentation	Kurzbeschreibung in der Datei RTK.HTM im ADSI SDK

Tabelle 13.43:
Schnellinfo
ADsSecurity-
Komponente
(Forts.)

EXKURS: Grundlagen der Windows-Sicherheit

Zum besseren Verständnis des Umgangs mit der ADSSECURITY.DLL seien an dieser Stelle kurz die Grundlagen der Windows-Sicherheit dargestellt.

EXKURS

Jeder Benutzer und jede Benutzergruppe besitzt einen so genannten *Security Identifier* (kurz: *SID*), der den Benutzer bzw. die Gruppe eindeutig identifiziert. Ein SID ist ein Zahlenarray variabler Länge. Jedes Objekt (z.B. eine Datei, ein Dateiordner, ein Eintrag im Active Directory, ein Registrierungsschlüssel) besitzt zur Speicherung der Zugriffsrechte einen so genannten *Security Descriptor* (kurz: *SD*; dt. Sicherheitsbeschreibung). Ein SD besteht aus drei Teilen:

SD und SID

- ▶ aus dem *Security Identifier (SID)* des Besitzers. Ein SID ist ein Zahlenarray variabler Länge.
- ▶ aus einer *Discretionary ACL (DACL)*, die die Zugriffsrechte beschreibt
- ▶ aus einer *System ACL (SACL)*, die die Überwachungseinstellungen enthält

Eine *Access Control List (ACL)* (sowohl DACL als auch SACL) besteht aus *Access Control Entries (ACE)*. Ein ACE wiederum enthält folgende Informationen:

ACL und ACE

- ▶ **Trustee:** der SID des Benutzers bzw. der Gruppe
- ▶ **AceType:** Der Typ ist entweder *ACCESS_ALLOWED_ACE* (0) oder *ACCESS_DENIED_ACE* (1). ACEs in einer SACL haben immer den Typ *SYSTEM_AUDIT_ACE*.
- ▶ **AccessMask:** Die AccessMask definiert die Rechte. Für jeden Objekttyp gibt es unterschiedliche Rechte. Jedes Recht ist dabei ein Bit bzw. eine Kombination von Bits in diesem Long-Wert. Eine AccessMask besteht in der Regel aus der Addition mehrerer einzelner Zugriffsrechte.
- ▶ **AceFlags:** Über die AceFlags wird die Vererbung der Rechte gesteuert.

Active Directory Service Interface (ADSI)

Tabelle 13.44:
AceFlags

Symbolische Konstante	Wert	Erläuterung
OBJECT_INHERIT_ACE	1	Der ACE wird vererbt.
CONTAINER_INHERIT_ACE	2	Der ACE wird vererbt (gilt für Container).
NO_PROPAGATE_INHERIT_ACE	4	Der ACE wird vererbt, darf aber nicht weitervererbt werden.
INHERIT_ONLY_ACE	8	Der ACE wird vererbt, hat aber keinen Einfluss auf das Objekt, bei dem er gespeichert ist.
INHERITED_ACE	16	Wird vom System gesetzt, um einen geerbten ACE anzuzeigen.

AccessMask-Werte

Die folgenden Tabellen zeigen die Konstanten der AccessMask für Registry-, Datei-, Ordner- und Exchange-Objekte sowie für Objekte im Active Directory.

Tabelle 13.45:
Konstanten für
Rechte auf
Exchange-Objekten
(ADS_RIGHT_EXCH_ENUM)

Symbolische Konstante	Wert
ADS_RIGHT_EXCH_ADD_CHILD	1
ADS_RIGHT_EXCH_MODIFY_USER_ATT	2
ADS_RIGHT_EXCH_MODIFY_ADMIN_ATT	4
ADS_RIGHT_EXCH_DELETE	65536
ADS_RIGHT_EXCH_MAIL_SEND_AS	8
ADS_RIGHT_EXCH_MAIL_RECEIVE_AS	16
ADS_RIGHT_EXCH_MAIL_ADMIN_AS	32
ADS_RIGHT_EXCH_DS_REPLICATION	64
ADS_RIGHT_EXCH_MODIFY_SEC_ATT	128
ADS_RIGHT_EXCH_DS_SEARCH	256

Tabelle 13.46:
Konstanten für
Rechte auf Registrie-
rungsschlüssel
(ADS_RIGHT_RGY_ENUM in der
ADsSecurity.dll)

Symbolische Konstante	Wert
ADS_RIGHT_RGY_KEY_READ	131097
ADS_RIGHT_RGY_KEY_WRITE	131078
ADS_RIGHT_RGY_KEY_EXECUTE	131097
ADS_RIGHT_RGY_KEY_ALL_ACCESS	983103
ADS_RIGHT_RGY_KEY_QUERY_VALUE	1
ADS_RIGHT_RGY_KEY_SET_VALUE	2
ADS_RIGHT_RGY_CREATE_SUB_KEY	4

Zusatzkomponenten für ADSI

Symbolische Konstante	Wert
ADS_RIGHT_RGY_ENUMERATE_SUB_KEYS	8
ADS_RIGHT_RGY_NOTIFY	16
ADS_RIGHT_RGY_CREATE_LINK	32

*Tabelle 13.46:
Konstanten für
Rechte auf Registrie-
rungsschlüssel
(ADS_RIGHT_
RGY_ENUM in der
ADsSecurity.dll)
(Forts.)*

Symbolische Konstante	Wert	Bezeichnung im Windows2000 GUI
FILE_READ_DATA (Datei) bzw. FILE_LIST_DIRECTORY (Ordner)	1	Datei lesen bzw. Ordner auflisten
FILE_WRITE_DATA (Datei) bzw. FILE_ADD_FILE (Ordner)	2	Daten schreiben bzw. Dateien erstellen
FILE_APPEND_DATA (Datei) bzw. FILE_ADD_SUBDIRECTORY (Ordner)	4	Daten an Datei anhängen bzw. Ordner erstellen
FILE_READ_EA	8	Erweiterte Attribute lesen
FILE_WRITE_EA	16	Erweiterte Attribute schrei- ben
FILE_EXECUTE (Datei) bzw. FILE_TRAVERSE (Ordner)	32	Datei ausführen bzw. Ordner durchsuchen
FILE_DELETE_CHILD (nur Ordner)	64	Unterordner und Dateien löschen
FILE_READ_ATTRIBUTES	128	Attribute lesen
FILE_WRITE_ATTRIBUTES	256	Attribute schreiben
DELETE	65536	Löschen
READ_CONTROL	131072	Berechtigungen lesen
WRITE_DAC	262144	Berechtigungen ändern
WRITE_OWNER	524288	Besitzrechte übernehmen
SYNCHRONIZE	1048576	(Wird zur Synchronisierung verwendet)

*Tabelle 13.47:
Konstanten für
Rechte auf Dateien
und Ordner. Diese
Konstanten sind
leider weder in der
ADsSecurity.dll
noch in der
ActiveDS.tlb
definiert.*

Symbolische Konstante	Wert
ADS_RIGHT_DELETE	65536
ADS_RIGHT_READ_CONTROL	131072
ADS_RIGHT_WRITE_DAC	262144
ADS_RIGHT_WRITE_OWNER	524288
ADS_RIGHT_SYNCHRONIZE	1048576

*Tabelle 13.48:
Konstanten für
Rechte auf Active
Directory-Einträge
(ADS_RIGHTS_
ENUM in der
ActiveDS.tlb)*

Active Directory Service Interface (ADSI)

Tabelle 13.48:
Konstanten für
Rechte auf
Active Direc-
tory-Einträge
(ADS_RIGHTS
_ENUM in der
ActiveDS.tlb)

Symbolische Konstante	Wert
ADS_RIGHT_ACCESS_SYSTEM_SECURITY	16777216
ADS_RIGHT_GENERIC_READ	-2147483648
ADS_RIGHT_GENERIC_WRITE	1073741824
ADS_RIGHT_GENERIC_EXECUTE	536870912
ADS_RIGHT_GENERIC_ALL	268435456
ADS_RIGHT_DS_CREATE_CHILD	1
ADS_RIGHT_DS_DELETE_CHILD	2
ADS_RIGHT_ACTRL_DS_LIST	4
ADS_RIGHT_DS_SELF	8
ADS_RIGHT_DS_READ_PROP	16
ADS_RIGHT_DS_WRITE_PROP	32
ADS_RIGHT_DS_DELETE_TREE	64
ADS_RIGHT_DS_LIST_OBJECT	128
ADS_RIGHT_DS_CONTROL_ACCESS	256

Well Known Security Principals

Bekannte Benutzer

Neben Benutzern und Gruppen kennt Windows auch Pseudo-Gruppen wie »Jeder«, »Interaktive Benutzer« und »System«. Diese Gruppen werden *Well Known Security Principals* genannt. Für die Änderungen von Sicherheitseinstellungen werden die in der nächsten Tabelle gezeigten SIDs benötigt. Im Windows2000 Active Directory sind die *Well Known Security Principals* im *ConfigurationNamingContext* in dem Container CN=Well Known Security Principals abgelegt. Sie finden diese Benutzer jedoch nicht im *DefaultNamingContext*.



Verwechseln Sie die *Well Known Security Principals* nicht mit den *BuiltIn-Konten* (z.B. Gäste, Administratoren, Benutzer). Letztere finden Sie in Active Directory im *DefaultNamingContext* in cn=BuiltIn.

Tabelle 13.49:
SIDs der Well
Known Security
Principals

Well Known Security Principal	SID
Anonymous Logon	1;1;0;0;0;0;0;5;7;0;0;0
Authenticated Users	1;1;0;0;0;0;0;5;11;0;0;0
Batch	1;1;0;0;0;0;0;5;3;0;0;0
Creator Group	1;1;0;0;0;0;0;3;1;0;0;0
Creator Owner	1;1;0;0;0;0;0;3;0;0;0;0
Dialup	1;1;0;0;0;0;0;5;1;0;0;0
Enterprise Domain Controllers	1;1;0;0;0;0;0;5;9;0;0;0

Well Known Security Principal	SID
Everyone	1;1;0;0;0;0;0;1;0;0;0;0
Interactive	1;1;0;0;0;0;0;5;4;0;0;0
Network	1;1;0;0;0;0;0;5;2;0;0;0
Proxy	1;1;0;0;0;0;0;5;8;0;0;0
Restricted	1;1;0;0;0;0;0;5;12;0;0;0
Self	1;1;0;0;0;0;0;5;10;0;0;0
Service	1;1;0;0;0;0;0;5;6;0;0;0
System	1;1;0;0;0;0;0;5;18;0;0;0
Terminal Server User	1;1;0;0;0;0;0;5;13;0;0;0

Tabelle 13.49:
SIDs der Well
Known Security
Principals
(Forts.)

Ermittlung der SIDs der Well Known Security Principals

Das folgende Skript zeigt, wie die obige Tabelle ermittelt wurde.

```
Set c = GetObject("LDAP://sonne2000/CN=WellKnown Security
Principals,CN=Configuration,DC=IT-Visions,DC=de")
For Each o In c
    say o.cn & ": " & flat(o.Get("objectsid"))
Next
```

Listing 13.65: Ermittlung der SIDs der Well Known Security Principals

Das ist das Ende des Exkurses zur Windows-Sicherheit – zurück zur Komponente *ADsSecurity*.

Klassen der ADsSecurity-Komponente

Die Komponente *ADsSecurity* stellt zwei Klassen bereit:

- ADsSecurity
- ADsSID

Klassen

Klasse ADsSecurity

Die Klasse *ADsSecurity* ist eng verzahnt mit den Klassen *SecurityDescriptor*, *AccessControlList* und *AccessControlEntry* des ADSI-Kerns.

ADsSecurity

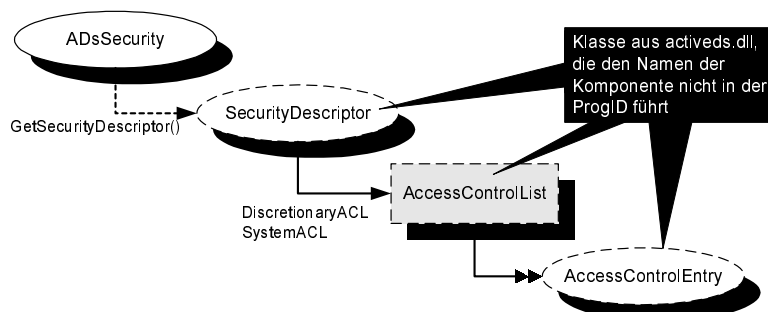


Abbildung 13.29:
Verzahnung der
ADsSecurity-Klasse
mit dem ADSI-Kern

Active Directory Service Interface (ADSI)

- Nur Methoden** Die Klasse `AdsSecurity` implementiert vier Methoden und keine Attribute:
- ▶ `GetSecurityDescriptor(pfad)` liefert den Security Descriptor (SD) als Zeiger auf die Schnittstelle des Typs `ActiveDs.SecurityDescriptor`.
 - ▶ `GetSecurityDescriptorAs (Format_Konst, pfad)` liefert den SD in einem bestimmten Format. Möglich sind
 - ▶ RAW (0),
 - ▶ HEXSTRING (1),
 - ▶ Security Descriptor Definition Language (SDDL) (2)
 - ▶ ein Zeiger auf `ActiveDs.SecurityDescriptor` (3)
- Mit dem Wert 3 ist die Methode äquivalent zu `GetSecurityDescriptor()`.

- ▶ `SetSecurityDescriptor (oSD, [pfad])` setzt den SD. Der Parameter `oSD` ist dabei ein Zeiger des Typs `ActiveDs.SecurityDescriptor`. Die Pfadangabe ist optional: Ohne Pfad wird der SD dorthin geschrieben, wo er zuvor mit `GetSecurityDescriptor()` hergeholt wurde.
- ▶ `GetSID(pfad)` liest den SID eines Verzeichniseintrags aus und liefert eine Instanz der Klasse `AdsSID`. Hier sind nur WinNT und LDAP-Pfade erlaubt.

Klasse `AdsSID`

- AdsSID** Die Klasse `AdsSID` speichert einen Security Identifier (SID) und ermöglicht es, diesen in einem bestimmten Format auszugeben bzw. zu setzen. Die Klasse dient also der Umwandlung von verschiedenen Darstellungen eines Benutzerkontos, insbesondere der Umwandlung eines textlichen Benutzernamens in einen SID.
- ▶ `GetAs(SID_Format)` gibt den SID in dem angegebenen Format aus.
 - ▶ `SetAs(SID_Format, wert)` belegt die Instanz von `AdsSID` mit einem neuen Wert. Dabei muss der Wert in dem angegebenen Format übergeben werden. Diese Methode initialisiert lediglich den internen Zustand von `AdsSID` neu. Damit wird keine Veränderung in einem persistenten Objekt bewirkt.

Tabelle 13.50:
Mögliche Werte für
SID-Formate
(`ADS_SID_`
`FORMAT`)

Konstante	Wert	Erläuterung
<code>ADS_SID_RAW</code>	0	Internes Format (nicht druckbar)
<code>ADS_SID_HEXSTRING</code>	1	Hexadezimaldarstellung
<code>ADS_SID_SAM</code>	2	Form <code>domain\user</code>
<code>ADS_SID_UPN</code>	3	Form <code>user@domain</code>
<code>ADS_SID_SDDL</code>	4	Security Descriptor Definition Language (SDDL)
<code>ADS_SID_WINNT_PATH</code>	5	WinNT-ADSI-Pfad
<code>ADS_SID_ACTIVE_DIRECTORY_PATH</code>	6	LDAP-ADSI-Pfad
<code>ADS_SID_SID_BINDING</code>	7	Form <code>GC://<SID= ...></code>

Identifikation und Bindung

Auch `AdsSecurity` verwendet ADSI-Pfade zur Identifikation der Objekte, deren ACL bearbeitet werden soll. Es gibt dabei für Exchange Server- und Active Directory LDAP-Objekte keinen Unterschied zwischen den Pfaden, die der ADSI-Kern auch benutzt. Sie können also direkt aus einem Exchange-Objekt dessen Pfad auslesen und damit auch auf die ACL zugreifen. Für Dateisystem und Registry gibt es eigene Moniker: `File://` und `RGY://`.

ADSI-Pfade

- Zugriff auf eine Datei

```
File://f:\test\test.txt
```

- Zugriff auf einen Ordner

```
File://f:\test\
```

- Zugriff auf einen lokalen Registry-Schlüssel

```
RGY://HKEY_LOCAL_MACHINE\SOFTWARE\IT-Visions\
```

- Zugriff auf einen entfernten Registry-Schlüssel

```
RGY://Sonne2000/HKEY_LOCAL_MACHINE\SOFTWARE\IT-Visions\
```

Arbeit mit der Komponente

Die grundsätzliche Vorgehensweise sieht folgendermaßen aus:

Vorgehensweise

- Erzeugen einer Instanz `AdsSecurity`. Bitte beachten Sie, dass die ProgID nicht wie üblich aus zwei Gliedern besteht.

```
Set oADSSEC = CreateObject("AdsSecurity")
```

- Auslesen des Security Descriptors (SD) aus einem Objekt, dessen ADSI-Pfad übergeben wird

```
Set oSD = oADSSEC.GetSecurityDescriptor(pfad)
```

- Zugriff auf die Discretionary Access Control List (DACL)

```
Set oDACL = oSD.DiscretionaryAcl
```

Danach kann die DACL ausgelesen werden, indem über den in der DACL enthaltenen ACE iteriert wird.

Ausgabe des Security Descriptors

Für die Ausgabe eines SDs seien zunächst zwei Hilfsroutinen definiert.

Einen SD ausgeben

- `list_acl()` gibt anhand eines übergebenen Objekts des Typs `AccessControlList` die ACL an. `list_acl()` benötigt die Angabe des Objekttyps, damit die richtige Konstantenliste für die Rechte angewendet werden kann. Der Objekttyp ist nicht aus dem Objektpfad eindeutig ablesbar, da verschiedene Objekttypen die gleiche ProgID verwenden (z.B. Exchange und Active Directory verwenden LDAP://, Datei und Ordner FILE://).

Active Directory Service Interface (ADSI)

- `list_sd()` setzt eine Ebene höher an und liefert den Besitzer und die ACL zu einem gegebenen Objektpfad. Dabei ist auch hier der Typ anzugeben, da dieser an `list_acl()` weitergereicht werden muss.

```
Sub list_sd(ByVal pfad, typ)
Dim oSD ' As ActiveDs.SecurityDescriptor
Dim oADSSEC ' As ADSecurity
Dim oACL ' As ActiveDs.AccessControlList
' -- Instanz erzeugen
Set oADSSEC = CreateObject("ADsSecurity")
' -- Security Descriptor holen
Const filepfad = "FILE://f:\test\1.txt"
Set oSD = oADSSEC.GetSecurityDescriptor(filepfad)
say "----- Rechte auf " & pfad
' Dim oACL As AccessControlList
say "Besitzer: " & oSD.Owner
' -- Zugriff auf Discretionary Access Control List
Set oACL = oSD.DiscretionaryACL
list_acl oACL, typ
End Sub
```

Listing 13.66: Ausgabe eines Security Descriptors [list_sd.wsf]

```
' ### Ausgabe einer ACL
Sub list_acl(oACL, typ)
' oACL As AccessControlList
Dim oACE ' As AccessControlEntry
Dim Ausgabe ' As String
Dim Rechtearray()
' *** Konstanten für Registry-Rechte
Dim arrADS_RIGHT_RGY_ENUM(10, 2)
arrADS_RIGHT_RGY_ENUM(0, 0) = "KEY_READ"
arrADS_RIGHT_RGY_ENUM(0, 1) = 131097
arrADS_RIGHT_RGY_ENUM(1, 0) = "KEY_WRITE"
...
' *** Konstanten für Datei-Rechte
Dim arrFile(14, 2)
arrFile(0, 0) = "READ_DATA"
arrFile(0, 1) = 1
arrFile(1, 0) = "WRITE_DATA"
...
' *** Konstanten für Ordner-Rechte
Dim arrFolder(15, 2)
arrFolder(0, 0) = "ADD_FILE "
arrFolder(0, 1) = 1
arrFolder(1, 0) = "ADD_SUBDIRECTORY "
...
' *** Konstanten für AD-Rechte
```

Zusatzkomponenten für ADSI

```
Dim arrADS_RIGHTS_ENUM(19, 2)
arrADS_RIGHTS_ENUM(0, 0) = "ADS_RIGHT_DELETE"
arrADS_RIGHTS_ENUM(0, 1) = 65536
...
' *** Konstanten für AD-Rechte
Dim arrADS_RIGHTS_ENUM(19, 2)
arrADS_RIGHTS_ENUM(0, 0) = "DELETE"
arrADS_RIGHTS_ENUM(0, 1) = 65536
arrADS_RIGHTS_ENUM(1, 0) = "READ_CONTROL"
...
' *** Konstanten für Exchange-Rechte
Dim arrADS_RIGHT_EXCH_ENUM(10, 2)
arrADS_RIGHT_EXCH_ENUM(0, 0) = "EXCH_ADD_CHILD"
arrADS_RIGHT_EXCH_ENUM(0, 1) = 1
arrADS_RIGHT_EXCH_ENUM(1, 0) = "EXCH_MODIFY_USER_ATT"
...
' -- Fallunterscheidung: Bindung an richtiges Array
Select Case LCase(typ)
    Case "file": Rechtearray = arrFile
    Case "folder": Rechtearray = arrFolder
    Case "registry": Rechtearray = arrADS_RIGHT_RGY_ENUM
    Case "exchange": Rechtearray = arrADS_RIGHT_EXCH_ENUM
    Case "ad": Rechtearray = arrADS_RIGHTS_ENUM
    Case Else: MsgBox "Ungültiger Typ!"
                Exit Sub
End Select

say "Anzahl der ACEs: " & oACL.AceCount
' -- Iteration über die Einträge
For Each oACE In oACL
    If oACE.AceType = 0 Then ' ACCESS_ALLOWED_ACE
        Ausgabe = oACE.Trustee & " darf "
    Else ' ACCESS_DENY_ACE
        Ausgabe = oACE.Trustee & " darf NICHT "
    End If
    Ausgabe = Ausgabe & get_from_array_mult(oACE.accessmask, Rechtearray)
    Ausgabe = Ausgabe & "(" & oACE.Flags & ")"
    say Ausgabe
Next
End Sub
```

Listing 13.67: Ausgabe der ACL. Die umfangreiche Array-Definition für die Rechtekonstanten ist hier aus Platzgründen nur ausschnittsweise abgedruckt, auf der Buch-CD jedoch enthalten. [list_acl.wsf]

```
Sub adsiextras_printacIs()
' Konstanten bitte anpassen!
Const filepath = "FILE://f:\test\1.txt"
```

Active Directory Service Interface (ADSI)

```
Const folderpfad = "FILE://f:\test"
Const regpfad = "RGY://Mars/HKEY_LOCAL_MACHINE\SOFTWARE\IT-Visions"
Const adspfad = "LDAP://sonne2000/CN=Schema,CN=Configuration,DC=IT-Visions,DC=de"
list_sd filepfad, "file"
list_sd folderpfad, "folder"
list_sd regpfad, "Registry"
list_sd adspfad, "AD"
list_sd expfad, "Exchange"
End Sub
```

Listing 13.68: Beispiel für die Ausgabe der Rechte auf verschiedenen Objekten mit Hilfe der Routinen list_sd() und list_acl()

Rechteänderung

Einen SD verändern

Die Rechte auf ein Objekt können geändert werden, indem der Security Descriptor (SD) mit GetSecurityDescriptor() gelesen wird, danach die ACEs in der ACL geändert werden und anschließend der SD mit SetSecurityDescriptor() zurückgeschrieben wird. Ein ACE-Objekt wird extern instanziiert und mit der Methode AddAce() an ein AccessControlList-Objekt angefügt.

```
Sub perm_add(pfad, Trustee, Mask, Typ, Flags)
Dim oSD ' As ActiveDs.SecurityDescriptor
Dim oADSSEC ' As ADsSecurity
Dim oACL ' As ActiveDs.AccessControlList
Dim oACE ' As ActiveDs.AccessControlEntry
' -- Instanz erzeugen
Set oADSSEC = CreateObject("ADsSecurity")
' -- Security Descriptor holen
Set oSD = oADSSEC.GetSecurityDescriptor(CStr(pfad))
' -- Zugriff auf DACL
Set oACL = oSD.DiscretionaryAcl
' -- Neuer ACE
Set oACE = CreateObject("AccessControlEntry")
' -- Werte eintragen
oACE.Trustee = Trustee
oACE.accessmask = Mask
oACE.AceType = Typ
oACE.AceFlags = Flags
' -- ACE anfügen
oACL.AddAce oACE
' -- Neuen SD schreiben
oADSSEC.SetSecurityDescriptor oSD
say "Recht für " & Trustee & " eingetragen"
End Sub
```

Listing 13.69: Die Hilfsroutine perm_add() fügt ein Recht an eine ACL an. [perm_add.wsf]

Zusatzkomponenten für ADSI

```
' === Test von Perm_Add()
Const ADS_ACETYPE_ACCESS_ALLOWED = 0
Const ADS_ACETYPE_ACCESS_DENIED = 1
Const ADS_RIGHT_GENERIC_ALL = &H10000000
Const ADS_ACEFLAG_INHERIT_ACE = 2
Const pfad = "RGY://HKEY_LOCAL_MACHINE\Software\it-visions"
perm_add pfad, "mars\hs", ADS_RIGHT_GENERIC_ALL, _
ADS_ACETYPE_ACCESS_DENIED, ADS_ACEFLAG_INHERIT_ACE
perm_add pfad, "S-1-1-0", ADS_RIGHT_GENERIC_ALL, _
ADS_ACETYPE_ACCESS_ALLOWED, ADS_ACEFLAG_INHERIT_ACE
```

Listing 13.70: Test von perm_add(). Bitte passen Sie den Pfad auf Ihre Umgebung an.
[ADSIEXTRAS_filepermadd.wsf]

Ein ACE kann mit der Methode RemoveAce(oACE) auf einem AccessControlList-Objekt unter Angabe eines AccessControlEntry-Objekts entfernt werden.

ACE entfernen

Die Klasse ActiveDs.AccessControlEntry erwartet im Attribut Trustee einen String, keinen SID in Array-Form. Es gibt eine textliche Umschreibung eines SIDs. Wichtig ist dies bei den *Well Known Security Principals*, die nicht wie alle Benutzer in der Form DOMAIN/USER spezifiziert werden können. Der textliche SID für *Well Known Security Principals* hat die Form S-x-y-z, wobei

- ▶ x die erste Zahl im SID
- ▶ y die achte Zahl im SID
- ▶ z die neunte Zahl im SID

ist. Aus 1;1;0;0;0;0;1;0;0;0;0 (Everyone) wird so S-1-1-0 und aus 1;1;0;0;0;0;5;18;0;0;0 (System) wird S-1-5-18.



13.11.6 IADsTools

Im Rahmen der Windows2000 Support Tools, die auf der Windows2000 Installations-CD-ROM enthalten sind, liefert Microsoft eine automationsfähige COM-Komponente, die viele Funktionen für die Programmierung mit dem Active Directory bereitstellt.

Name und Abkürzung	IADsTools
Name der Komponentendatei	IADsTools.dll
Interner Name der Typbibliothek	IADsTools
Helpstring der Typbibliothek	IADsTools
Hersteller	Microsoft
Lizenzierung	Kostenloses Add-on; die Komponente ist im ADSI SDK 2.5 enthalten

Tabelle 13.51:
Schnellinfo
IADsTools

Tabelle 13.51:
Schnellinfo
IADsTools
(Forts.)

Besprochene Version	1.0.0.2182
Windows2000 und Whistler (Windows XP / Windows.NET)	Installation der Windows2000 Support Tools aus dem Verzeichnis /SUPPORT/TOOL der Windows2000 Server-Installations-CD
NT4, Windows 9x, Windows ME	nicht verfügbar
Dokumentation	Teilweise dokumentiert in einer mit den Support Tools mitgelieferten .DOC-Datei (IADSTOOLS.DOC)

Feature-Überblick

IADsTools definiert 32 Klassen. Allein die Klasse `DCFunctions` implementiert 74 Methoden. Da eine ausführliche Besprechung den Rahmen dieses Buchs sprengen würde, seien an dieser Stelle nur einige der Möglichkeiten von *IADsTools* aufgelistet:

- ▶ Umformung von Namen (z.B. zwischen AD-RDN und WinNT-RDN, zwischen LDAP-Syntax und DNS-Syntax, Extraktion eines RDNs aus einem DN, GUID in ADSI-Pfad)
- ▶ Unterstützung der Arbeit mit Standorten und Servern
- ▶ Ermittlung der Server, die eine bestimmte Flexible Single Master Operations (FSMO)-Rolle realisieren (z.B. Schema-Master, PDC-Emulator)
- ▶ Arbeit mit Group Policy Objects (GPO)
- ▶ Arbeit mit Vertrauensstellungen
- ▶ Test einer LDAP-Verbindung (Methode `TestBind()`)
- ▶ Senden einer Netzwerknachricht (Methode `NetSendMessage()`)
- ▶ Informationen über Replikationseinstellungen und -abläufe
- ▶ Informationen über IP-Konfiguration

14 Windows Management Instrumentation (WMI)

Die *Windows Management Instrumentation (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 Einzelkomponenten 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**

Der Microsoft Systems Management Server (SMS) verwendet WMI zum Zugriff auf Systeminformationen.

14.1 WMI-Grundlagen

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 im WBEM die Repräsentation einer (Hardware- oder Software-)Ressource ein *Managed Object (MO)*. **Managed Objects**

Der Name *Web Based Enterprise Management* ist irreführend, weil er nahelegt, 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.



Da sich dieses Buch nur mit der Windows-Plattform beschäftigt, wird im Folgenden nicht WBEM, sondern hauptsächlich der Begriff WMI verwendet. Als deutsche Übersetzung für Windows Management Instrumentation verwendet Microsoft »Windows-Verwaltungsinstrumentation«.

Features

14.1.1 WMI-Funktionsumfang

Die Informationsfülle, die WMI bereits heute liefert, ist riesig. Beispiele für Bereiche, aus denen WMI Informationen liefert, zeigt die folgende Tabelle.

Tabelle 14.1:
Überblick über
WMI-Informationen

- ▶ Installierte Software
- ▶ BIOS
- ▶ Boot-Konfiguration
- ▶ Installierte Hardware (z.B. Netzwerkkarten, Grafikkarten) einschließlich Treiber und deren Zuordnung zu LoadOrder-Groups, belegter Ressourcen (IRQ, Port, DMA), Konfiguration (z.B. Druckereinstellungen)
- ▶ Ordner und Dateien des Dateisystems
- ▶ Netzlaufwerksverbindungen
- ▶ Installiertes Betriebssystem (z.B. Betriebssystemname, Build-Version, Installationsdatum, Datum und Uhrzeit des letzten Boot-Vorgangs) Installierte Updates und Hotfixes
- ▶ Umgebungsvariablen
- ▶ Performance Monitor-Daten
- ▶ SNMP-Daten
- ▶ Laufende Prozesse
- ▶ Programmgruppen im Startmenü
- ▶ NT-Dienste (Win32_SystemServices)
- ▶ NT-Ereignisprotokoll
- ▶ Eingerichtete Zeitzonen

- ▶ Benutzerkonten (inkl. deren Gruppenzuordnung, Desktop-Einstellungen und Ereignisprotokolleinträge)
- ▶ Registry
- ▶ Dateisicherheit, Freigabesicherheit
- ▶ Active Directory
- ▶ Installierte COM-Komponenten einschließlich Zuordnung zu Komponentenkategorien und DCOM-Einstellungen
- ▶ Druckaufträge
- ▶ Geplante Vorgänge (Taskscheduler)
- ▶ ODBC-Einstellungen
- ▶ Auslagerungsdateien

14.1.2 WMI in Whistler

Whistler (Windows XP/Windows.NET) enthält eine erweiterte Version von WMI. Neu sind folgende Funktionen:

Neue Funktionen

- ▶ Event Correlation (Zusammenfassung von Ereignissen)
- ▶ Garantierte Zustellung von Ereignissen durch optionale Integration mit dem Microsoft Message Queue Service (MSMQ)
- ▶ Zahlreiche neue WMI-Klassen zur Verwaltung folgender Bereiche:

Neue Klassen

- ▶ Distributed File System (DFS)
- ▶ IP-Routing
- ▶ Disk Quotas
- ▶ Ausführung eines Ping
- ▶ Ausführung von CHKDSK
- ▶ Internet Information Services (IIS)
- ▶ Netzwerkverbindungen und Sitzungen
- ▶ Drucker und Druckerwarteschlangen
- ▶ Terminal Services
- ▶ Datum und Uhrzeit
- ▶ Active Directory-Replikation
- ▶ Clustering

- ▶ DNS Server
- ▶ Network Load Balancing (NLB)
- ▶ Security Configuration Editor (SCE)
- ▶ Prozessüberwachung
- ▶ Erweiterungen im WMI-Meta-Objektmodell
- ▶ Verbesserung des Zugriffs auf das Active Directory
- ▶ Neue Syntax zur Objektidentifikation: Universal Management Interface (UMI) Syntax

14.1.3 Installation und Konfiguration

WMI ist in Windows2000 und den Nachfolgeversionen (Windows XP/Windows.NET) integriert und als Add-on verfügbar für Windows 95/98/ME und NT4. Die WMI-Kernel für NT4 und Windows 95/98/ME sind auf der CD enthalten [CD:/install/komponenten/wmi/]. Im Gegensatz zu ADSI muss WMI bei entfernten Operationen sowohl auf dem Client als auch auf dem Server installiert sein.

Die in Windows2000 enthaltene Version von WMI hatte die offizielle Versionsnummer 1.5 (Intern: 1085). In Windows XP/Windows.NET gibt es eine erweiterte Version von WMI, die zum Zeitpunkt des Redaktionsschlusses dieses Buchs noch keine offizielle Versionsnummer hatte.

```
Dim objW0 ' As WbemScripting.SwbemObject
Set objW0 = GetObject("winmgmts:root\default:__cimomidentification=@")
say "WMI-Version: " & objW0.versionusedtocreatedb
```

Listing 14.1: Ermittlung der WMI-Version [wmi_vers.wsfl]



WinMgmt.exe

Ob WMI installiert ist oder nicht, erkennt man daran, ob der Registry-Schlüssel `HKEY_LOCAL_MACHINE\SOFTWARE\MICROSOFT\WBEM` existiert oder nicht.

WMI wird durch die ausführbare Datei `WINMGMT.EXE` implementiert. `WINMGMT.EXE` läuft unter NT4 und Windows2000 als Dienst unter dem Namen »WinMgmt (Windows-Verwaltungsinstrumentation)«. Auf Windows 95/98/ME wird `WINMGMT.EXE` beim ersten WMI-Aufruf als normaler Prozess gestartet. Das ist auch unter NT möglich.

```
winmgmt [/exe] [/regserver] [/unregserver]
```

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`.

WMI-Konfiguration in der MMC

Zur Konfiguration von WMI liefert Microsoft unter Windows2000 ein MMC-Snap-In und für NT4 sowie für Windows 95/98/ME die eigenständige WMI Control Application (WBEMCNTL.EXE).

wbemcntl.exe

Das Windows2000-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 des Computerverwaltungs-Snap-Ins.

MMC Snap-In

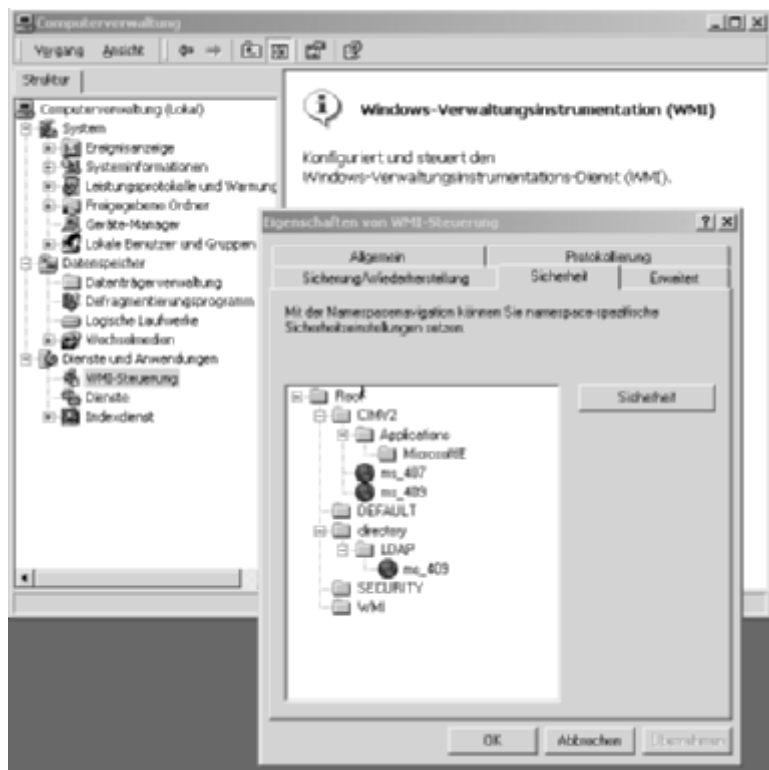


Abbildung 14.1:
WMI-Steuerung in
der MMC

Das Register SICHERHEIT ermöglicht die Einstellung der Zugriffsrechte auf Ebene jedes einzelnen WMI-Namespaces.

Weitere Tools

Die Erläuterung von WMI-Tools finden Sie in Kapitel 14.7



Abbildung 14.2:
Standardrechte der
Administratoren auf
dem Namespace /
ROOT/CIMV2. Die
Rechte werden alle
von /ROOT geerbt.



14.1.4 WMI-Provider

Provider WMI basiert ebenso wie ADSI und ADO 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.

Tabelle 14.2:
Verfügbare
WMI-Provider in
WMI 1.5

WMI-Provider	Erläuterung
Directory Services Provider	Zugriff auf die ADSI-Informationen aus Verzeichnisdiensten
Event Log Provider	Zugriff auf das NT-Ereignisprotokoll
Exchange Queue Provider, Exchange Routing TableProvider, ExchangeClusterProvider	Zugriff auf Microsoft Exchange Server 2000
Microsoft Windows Installer Provider	Zugriff auf Software, die durch den Windows Installer (MSI) installiert wurde

WMI-Provider	Erläuterung
Performance Counters Provider	Zugriff auf rohe Performance Counter-Daten
Performance Monitor Provider	Zugriff auf Performance-Daten, wie sie der NT-Performance Monitor sieht
Power Management Event Provider	Events aus dem Bereich Power Management
Registry Event Provider	Events bei Änderungen in der Registry
Registry Provider	Zugriff auf die Registry
Security Provider	Zugriff auf Sicherheitsinformationen im NTFS-Dateisystem
SMS Provider	Zugriff auf Microsoft System Management Server
SNA Provider	Zugriff auf Microsoft SNA Server
SNMP Provider	Zugriff auf SNMP-Daten
View Provider	Dient der Erzeugung neuer Klassen
WDM Provider	Zugriff auf Gerätetreiber via Windows Driver Model (WDM)
Win32 Provider	Zugriff auf das Win-32-Subsystem

Nicht alle diese Provider werden automatisch registriert und im CIM Repository eingetragen. In der MSDN Library finden Sie die Informationen darüber, wie die einzelnen Provider registriert werden.

Eine komplette Referenz der durch die verschiedenen WMI-Provider implementierten Managed Objects finden Sie jedoch nicht in der MSDN Library, sondern nur online über [HTTP://MSDN.MICROSOFT.COM/DEVELOPER/SDK/WMISDK/SCHEMA/DEFAULT.HTM](http://msdn.microsoft.com/developer/sdk/wmisdk/schema/default.htm).



14.1.5 Programmierung mit der WMI-COM-Komponente

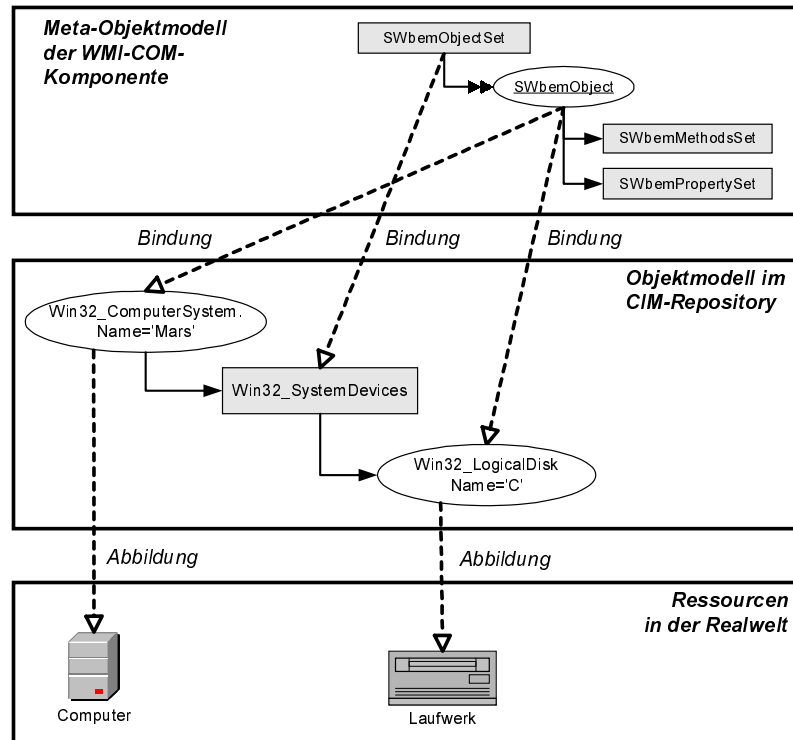
Vom eigentlichen WMI ist die WMI-COM-Komponente abzugrenzen. Sie realisiert ein Meta-Objektmodell (vgl. Erläuterungen dazu in Anhang A), um die sehr große Anzahl von WMI-Grundklassen ansteuern zu können und zu ermöglichen, dass zusätzliche WMI-Provider ohne Veränderung der WMI-COM-Komponente adressiert werden können.

In der Dokumentation spricht Microsoft von einem *WMI Scripting API* und einem *WMI COM API*. Als Scripting API bezeichnet Microsoft die IDispatch-fähigen Schnittstellen, als COM API die Schnittstellen, die nicht via IDispatch und damit nicht per Skript angesprochen werden können. Das vorliegende Buch macht diesen MINFU nicht mit und spricht von der WMI-Komponente.

WMI Scripting API

Windows Management Instrumentation (WMI)

Abbildung 14.3:
Das WMI-Meta-
Objektmodells: Die
Klasse SWbem
Object kann sowohl
an Instanzen der
Klasse
Win32_Computer-
System als auch an
Instanzen der Klasse
Win32_LogicalDisk
gebunden werden.
Win32_Computer-
System und
Win32_LogicalDisk
sind jeweils Abbil-
dungen von realen
Ressourcen.



Dieses Kapitel verfolgt zwei Ziele: Sie sollen zum einen die Grundkonzepte von WMI, zum anderen den Einsatz der WMI-Komponente verstehen. Das Buch liefert keine vollständige Darstellung der Managed Objects. Das liegt nicht nur daran, dass eine solche Referenz ein eigenes, mehrere hundert Seiten starkes Buch füllen würde, sondern auch daran, dass es gute Werkzeuge zum Zugriff auf das Schema und eine gute Online-Dokumentation der Managed Objects gibt. Mit den hier vermittelten grundlegenden Informationen werden Sie in der Lage sein, auch die hier nicht besprochenen Managed Objects schnell einzusetzen.

Tabelle 14.3:
Schnellinfo Win-
dows Management
Instrumentation
(WMI)

Name und Abkürzung	Windows Management Instrumentation (WMI)
Name der Komponentendatei	wbemdisp.dll (Typbibliothek: wbemdisp.tlb)
Interner Name der Typbibliothek	WbemScripting
Helpstring der Typbibliothek	Microsoft WMI Scripting Library
Hersteller	Microsoft

Lizenzierung	Bestandteile des Betriebssystems bzw. kostenloses Add-on
Besprochene Version	2600 (DLL-Version: 5.1.2600.0) Die WMI-Versionsnummer entspricht der Build-Nummer des Betriebssystems.
NT4 und Windows 95/98/ME	WMI-Add-on [CD:/install/komponenten/wmi/]
Windows2000	Version 1085 enthalten
Windows XP	Version 2600 enthalten
Windows.NET (Whistler Server) Beta 2	Version 2462 enthalten
Dokumentation	MSDN Library [MSL00]

Tabelle 14.3:
Schnellinfo Win-
dows Management
Instrumentation
(WMI)
(Forts.)

Programmiermöglichkeiten

WMI unterstützt sowohl den Zugriff auf einzelne Managed Objects als auch den Zugriff auf Objektmengen. Weiterhin unterstützt WMI folgende Vorgehensweisen:

- ▶ transparentes Schema zur Erforschung der Managed Objects
- ▶ Möglichkeit zur Definition eigener Ereignisse auf jede beliebige Veränderung in einem Managed Object
- ▶ asynchrone Befehlsausführungen, die ihre Ergebnisse durch Ereignisse zurückliefern
- ▶ Erstellen von Abfragen im SQL-Stil durch die Abfragesprache WQL

Programmierung

WMI Software Development Kit

Microsoft bietet zu WMI ein Software Development Kit (SDK) an, das Dokumentation, Beispielcode, C++-Include-Dateien und einen Teil der in Kapitel 14.7 beschriebenen Werkzeuge enthält. Das SDK ist auf der Buch-CD enthalten. Die jeweils aktuellste Version erhalten Sie unter [SDK00].



14.1.6 Wichtige WMI-Konzepte im Überblick

In diesem Unterkapitel erhalten Sie eine Einführung in die wichtigsten Konzepte von WMI.

Schema

WMI besitzt ein Schema (ähnlich wie ADSI). Ein WMI-Schema definiert die Klassen mit ihren Attributen und Methoden, die Vererbungs- und Containment-Hierarchie. Managed Objects können zur Laufzeit durch Assoziationen miteinander verbunden sein, so dass eine Containment-Hierarchie entsteht. WMI-Klassen verwenden Vererbung, so dass die Klassen in einer Vererbungshierarchie zueinander stehen.

Schema

Abbildung 14.4:
Die Objekt-
hierarchie von
Win32_LogicalDisk
über CIM_DataFile
bis zur Win32_SID



WMI-Repository

Genau genommen besitzt WMI mehrere Schemata, da jeder WMI-Name-space ein eigenes Schema hat. Ein WMI-Schema ist oft sehr umfangreich und enthält auch Hilfeinformationen, wahlweise in verschiedenen Sprachen. Das Schema ist im so genannten *WMI-Repository* abgelegt. Dieses liegt in einem gleichnamigen Verzeichnis unterhalb von %SYSTEMROOT%\SYSTEM32\WBEM. Das Schema für Windows heißt Win32 Schema Version 2 und ist eine Microsoft-Erweiterung des CIM-Schemas Version 2.

Auch der Aufbau des Schemas ist Teil des Schemas. Dieser Teil des Schemas wird als *Metamodell* bezeichnet.

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.



Abbildung 14.5:
WMI-Namens-
räume in Windows
Whistler Server
Beta2

Ein Namespace ist unabhängig von einem WMI-Provider: Ein Provider kann mehrere Namespaces realisieren, ebenso wie ein Namespace Klassen aus mehreren Providern enthalten kann.

Namespace versus Provider

Lokalisierung

WMI erlaubt Lokalisierung (d.h. landesspezifische Anpassung) von Schema-Informationen 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.

LocaleID Die landesspezifischen Informationen werden in Unter-Namespaces gespeichert. Jedes Land hat eine *LocaleID*.

- ▶ *ms_407* steht für Deutsch.
- ▶ *ms_409* steht für amerikanisches Englisch.

Der Namespace `\root\CIMV2\ms_407` ist also der `\root\CIMV2`-Namespace mit deutschen Hilfeinformationen, `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 14.6:
Anzeige der Name-
space-Hierarchie im
WMI-Snap-In in
der MMC (Eigen-
schaftsfenster des
Eintrags WMI-
Steuerung)



Klassen

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. WMI 1.5 enthält ca. 600 Klassen. In WMI 1.1 waren es 400.

Es ist eine Konvention, dass ein kompletter WMI-Klassenname aus dem Namespace-Namen und dem eigentlichen Klassennamen besteht, getrennt durch einen Unterstrich.

Namenskonvention

NamespaceName_ClassName

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.

Statische Methoden

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 sechzehn Standarddatentypen.

*Tabelle 14.4:
CIM-Standard-
datentypen
(definiert in der
Konstantenliste
WbemCimTypeEnum
in wbemdisp.tlb)*

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

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`.

Die Funktion `wmi_date()` konvertiert ein WMI-Datum in ein Visual Basic-Datum. Millisekunden werden dabei nicht berücksichtigt.

```
Function wmi_date(strWmiDat)
Dim jahr, monat, tag, stunde, min, sek
' -- Aufspaltung
jahr = Left(strWmiDat, 4)
monat = Mid(strWmiDat, 5, 2)
tag = Mid(strWmiDat, 7, 2)
stunde = Mid(strWmiDat, 9, 2)
```


WMI-Grundlagen

```
min = Mid(strWmiDat, 11, 2)
sek = Mid(strWmiDat, 13, 2)
' -- Datum zusammenbauen
wmi_date = tag & "." & monat & "." & jahr & " "
wmi_date = wmi_date & stunde & ":" & min & ":" & sek
wmi_date = CDate(wmi_date)
End Function
```

Listing 14.2: Konvertieren eines WMI-Datums in ein VB-Datum [wmi_date.wsf]

WMI kennt ein eigenes Format für Zeitintervalle: ddddddHHMMSS.mmmmm:000. Auch ein Zeitintervall wird als String abgelegt. Dabei repräsentiert dddddd die Anzahl der Tage. Der String endet immer auf :000.

Systemattribute

Alle WMI-Klassen und alle Instanzen dieser Klasse 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.

Systemattribute

Attribut	Erläuterung
__Class	Name der Klasse. Dieser Wert kann für Klassen geändert werden.
__Derivation	Ein Array of String, der die Vererbungshierarchie wiedergibt. Der erste Eintrag ist die direkte Oberklasse.
__Dynasty	Name der obersten Klasse der Vererbungshierarchie. Bei der obersten Klasse steht hier kein Leerstring, sondern der gleiche String wie bei __Class.
__Genus	1 = SWbemObject ist eine Klasse. 2 = SWbemObject ist eine Instanz.
__Namespace	Namespace, in dem die Klasse oder die Instanz existiert
__Path	Vollständiger WMI-Pfad einschließlich Server und Namespace
__Property_Count	Anzahl der Attribute der Klasse. Dabei werden diese systemattribute nicht mitgezählt.
__Relpath	WMI-Pfad ohne Server und Namespace
__Server	Name des Servers
__Superclass	Name der direkten Oberklasse

Tabelle 14.5:
WMI-Systemeigenschaften

Managed Object Format (MOF)

MO-Beschreibung durch 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.

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;
```

WMI-Grundlagen

```
[read: ToInstance ToSubClass] real32 Size;  
[read: ToInstance ToSubClass] datetime CreateDate;  
[read: ToInstance ToSubClass] string User;  
[read: ToInstance ToSubClass] string JetVersion;  
};
```

Listing 14.3: 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.



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.

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«.

Im Gegensatz zu anderen Komponenten erlaubt WMI Vorgaben für die COM-Sicherheit durch den Client. Sowohl Impersonifizierungs- als auch Authentifizierungsmodus (vgl. Kapitel 2) können beim Verbindungsaufbau eingestellt werden. Der Client kann unter Windows2000 auch den Security Service Provider (SSPI) 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.

Sicherheit

Impersonifizierung

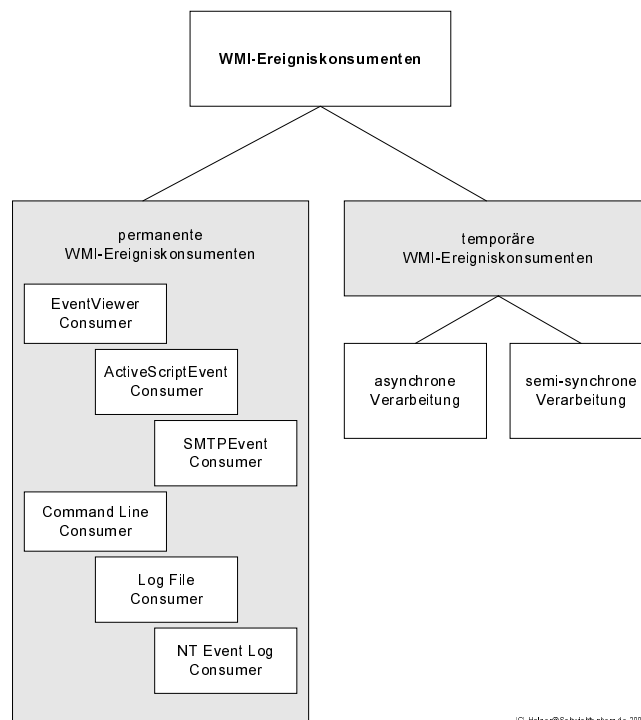
Programmatische Sicherheit

Ereignisse

Konsumenten WMI bietet ein komplexes System für Ereignisse in Managed Objects. Dabei registrieren sich so genannte *Ereigniskonsumenten* (*Event Consumers*) bei WMI für bestimmte Ereignisse. Der Event Consumer führt beim Eintritt eines Ereignisses eine bestimmte Aktion aus.

Permanente versus temporäre Konsumenten WMI unterscheidet zwei Arten von Ereigniskonsumenten: *temporäre Event Consumer* und *permanente Event Consumer*. Der Unterschied zwischen den beiden Typen ist, dass ein temporärer Event Consumer nur Ereignisbenachrichtigungen erhält, wenn er aktiv ist. Ein temporärer Event Consumer wird durch ein Skript oder ein Programm implementiert. Nach Beendigung des Skriptes/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.

Abbildung 14.7:
Überblick über die
Ereignis-
konsumenten



© Holger@Schneideholz.de 2001

Permanente Event Consumer sind z.B.:

- EventViewerConsumer** ► 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 14.7 vorgestellt.

WMI-Grundlagen

- ▶ der *ActiveScriptEventConsumer* (im Namespace /ROOT/DEFAULT), der beim Auftreten eines Ereignisses ein Active Script ausführt **ActiveScriptEvent-Consumer**
- ▶ Mit Hilfe des *SMTPEventConsumer* 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. **SMTPEvent-Consumer**

Whistler (Windows XP/Windows.NET) enthält drei weitere permanente Ereigniskonsumenten: **Neu in Whistler**

- ▶ Command Line Event Consumer: Start einer Anwendung
- ▶ NT Event Log Event Consumer: Eintrag in das NT-Ereignisprotokoll
- ▶ Log File Event Consumer: Eintrag in eine Protokolldatei

Der Event Consumer 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 Event Consumer ist eine Instanz einer von der Systemklasse `__EventConsumer` ererbenden Klasse. Ein WMI-Filter ist eine Instanz der Systemklasse `__EventFilter`. Die Bindung zwischen einem Consumer und einem Filter ist als Instanz von `__FilterToConsumerBinding` 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.

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. **Provider**

WMI Query Language (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. **WQL**

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`. **Schlüsselwörter**

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 GallischesDorf 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 Systems Management Server enthält eine erweiterte Fassung von WQL unter dem Namen *Extended WQL*.

14.1.7 Bindung an Managed Objects

Auch in WMI werden Objekte durch Moniker identifiziert. Ein WMI-Moniker besteht aus:

- ▶ der ProgID *WinMgmts*
- ▶ einer Sicherheitseinstellungsdefinition (optional)
- ▶ einer Lokalisierungsangabe (optional)
- ▶ dem WMI-Pfad (nur im Ausnahmefall nicht vorhanden)

WMI-Moniker **WMI-Pfade**

Grundsätzlich gibt es zwei Arten von WMI-Pfaden:

Klassenpfade ▶ **Pfade für Klassen:** Pfade für Klassen bestehen lediglich aus dem Klassennamen und optional der Angabe eines Servers und/oder eines Namespaces. Ohne Serverangabe wird der lokale Server benutzt, ohne Namespace-Angabe der Standard-Namespace (*Default Namespace*). Als Trennstrich kann sowohl der Slash (»/«) als auch der Backslash (»\«) verwendet werden.

- ▶ \\Server\Namespace:Class
- ▶ Class

Instanzenpfade ▶ **Pfade für Objekte:** Pfade für WMI-Objekte (Instanzen von WMI-Klassen) bestehen zusätzlich aus der Angabe der Schlüsselattribute zur eindeutigen Identifizierung der Instanz. Die allgemeine Form lautet:

```
\\Server\Namespace:Class.Key="wert"
```

In dem Fall, dass die Klasse mehrere Attribute als Schlüsselattribute definiert hat, sind alle Schlüsselattribute anzugeben, da ein einzelnes dieser Attribute ja keine eindeutigen Werte besitzen muss.

```
\\Server\Namespace:Class.Key1="wert1",Key2="Wert2"
```

WMI-Grundlagen

Für den Fall, dass die Klasse nur ein Schlüsselattribut besitzt, kann die Schreibweise verkürzt werden, indem der Name des Schlüsselattributs weggelassen wird.

```
\\Server\Namespace:Class ="wert"
```

Für die Verwendung von Servernamen und Namespacenamen gilt das für Klassenpfade Gesagte.

Der Standard-Namespace ist in der Registry festgelegt (HKEY_LOCAL_MACHINE\SOFTWARE\MICROSOFT\WBEM\SCRIPTING\DEFAULT_NAMESPACE) und über das WMI-Snap-In in der MMC oder die WMI-Klasse Win32_WMISetting in dem Attribut ASPScriptDefaultNamespace veränderbar.



Objekt	Pfad
Lokaler Default Namespace	WINMGMTS:
Der Default Namespace auf dem Rechner <i>Sonne</i>	WINMGMTS:\\SONNE
Der Namespace ROOT/CIMV2 auf dem lokalen System	WINMGMTS:ROOT/CIMV2
Der Namespace ROOT/CIMV2 auf dem Rechner » <i>Sonne</i> «	WINMGMTS:\\SONNE/ROOT/CIMV2
Die Klasse Win32_LogicalDisk aus dem Namespace \ROOT\CIMV2: auf dem Computer » <i>Mars</i> «	WINMGMTS:\\MARS\ROOT\CIMV2:WIN32_LOGICALDISK
Die Instanz der Klasse Win32_LogicalDisk aus dem Namespace \ROOT\CIMV2: auf dem Computer <i>Mars</i> mit dem Namen »D:«	WINMGMTS:\\MARS\ROOT\CIMV2:WIN32_LOGICALDISK.DEVICEID="D:"
Die Instanz der Klasse Win32_LogicalDisk aus dem Default Namespace auf dem lokalen Computer mit dem Namen »D:«	WINMGMTS:WIN32_LOGICALDISK.DEVICEID="D:"

Tabelle 14.6:
Beispiele für WMI-Moniker

Lokalisierungsangabe

Innerhalb des WMI-Monikers kann ein Unter-Namespace, d.h. eine bestimmte landesspezifische Version eines Namespaces, ausgewählt werden. Die LocaleID wird in eckigen Klammern vor dem WMI-Pfad ergänzt.

```
WinMgmts:[locale=ms_407]!root/cimv2:Win32_LogicalDisk
WinMgmts:[locale=ms_409]!root/cimv2:Win32_LogicalDisk
```

LocaleID

Beispiele

Sicherheitseinstellungen

Sicherheit Alle Sicherheitseinstellungen können wahlweise über den Moniker bei Objektbindung oder über spezielle Verwaltungsklassen vorgenommen werden. Die Sicherheitseinstellungen erfolgen vor der Angabe des WMI-Pfads in geschweiften Klammern. Sie werden durch ein Ausrufezeichen vom WMI-Pfad abgetrennt. Mehrere Sicherheitsangaben werden durch Kommata getrennt. Folgende Eigenschaften können gesetzt werden:

► AuthenticationLevel

Authentication-Level DCOM- und Authentifizierungsstufe; mögliche Werte sind: *default*, *none*, *connect*, *call*, *pkt*, *pktIntegrity* und *pktPrivacy* (vgl. DCOM-Sicherheit in Kapitel 2).

► ImpersonationLevel

Impersonation-Level DCOM-Impersonifizierungsstufe; mögliche Werte sind: *anonymous*, *identify*, *impersonate* und *delegate* (vgl. Ausführungen zur DCOM-Sicherheit in Kapitel 2).

► Authority

Authority Gibt den zu verwendenden SSPI an; möglich sind *authority=kerberos:domainname\servername* und *authority=ntlm:domain:domainname*

► Privileges

Privileges Privileges sind Einzelrechte, die beim lokalen Zugriff zum Tragen kommen. Mögliche Werte sind: *CreateToken*, *PrimaryToken*, *LockMemory*, *IncreaseQuota*, *MachineAccount*, *Tcb*, *Security*, *TakeOwnership*, *LoadDriver*, *SystemProfile*, *SystemTime*, *ProfileSingleProcess*, *IncreaseBasePriority*, *CreatePagefile*, *CreatePermanent*, *Backup*, *Restore*, *Shutdown*, *Debug*, *Audit*, *SystemEnvironment*, *ChangeNotify*, *RemoteShutdown*, *Udock*, *SyncAgent*, *EnableDelegation*.

Da mehrere Werte gleichzeitig erlaubt sind, werden die Werte in runden Klammern zusammengefasst. Dabei wird das Schlüsselwort *Privileges* nicht vorangestellt.

Tabelle 14.7:
Beispiele für WMI-
Moniker mit Sicher-
heitseinstellungen

Objekt	Pfad
Zugriff auf den CIMV2-Namespace auf dem Server <i>Sonne</i> unter Verwendung bestimmter Impersonifizierungs- und Authentifizierungsstufen	WINMGMTS: {IMPERSONATIONLEVEL=IMPERSONATE, AUTHENTICATIONLEVEL=PKTPRIVACY} !//SONNE/ROOT/CIMV2
Zugriff auf den CIMV2-Namespace auf dem Server <i>Sonne</i> unter Verwendung der Impersonifizierungsstufe <i>Impersonate</i> und unter Verwendung des NTLM-SSPIs	WINMGMTS: {IMPERSONATIONLEVEL=IMPERSONATE, AUTHORITY=NTLMDOMAIN:SONNE} !\\SONNE/ROOT/ CIMV2

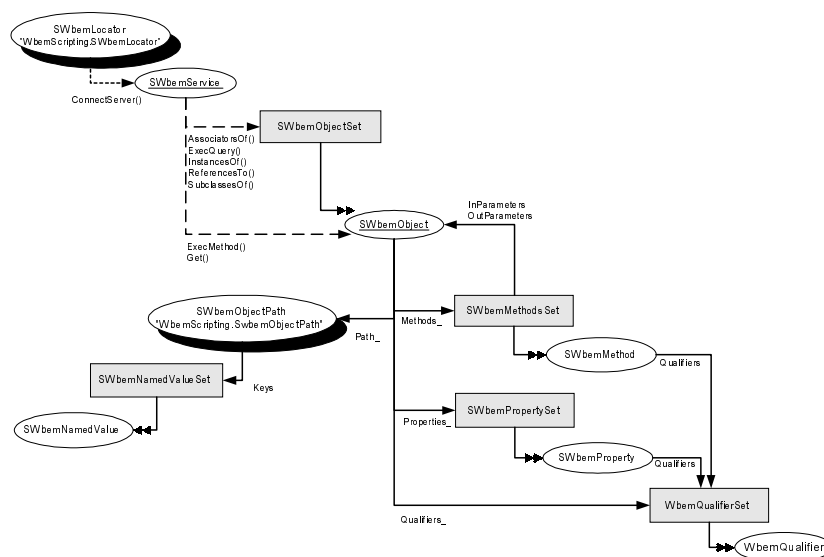
14.2 Objektmodell der WMI-Komponente

Den Kern des Objektmodells der WMI-Komponente bildet die WMI-Meta-Klasse `SWbemObject`, deren Instanzen an jedes Managed Object (MO) gebunden werden können. `SWbemObject` ist dabei in der Lage, das MO zu emulieren und wie ein Proxy-Objekt aufzutreten, so dass die zusätzliche Ebene für den Programmierer transparent ist.

`SWbemObject` kann auch an eine Klassendefinition im WMI-Schema gebunden werden. Da es schwierig ist, diesen Umstand in der Beschreibung des Objektmodells stets zu berücksichtigen, soll hier als Konvention eingeführt werden, dass der Begriff MO nicht nur die Instanzen, sondern auch die Klassendefinition eines MOs umfassen soll.

Die WMI-Komponente besitzt fünf instanziiierbare Klassen:

- ▶ `WbemScripting.SWbemLocator`
- ▶ `WbemScripting.SWbemLastError`
- ▶ `WbemScripting.SWbemObjectPath`
- ▶ `WbemScripting.SWbemNamedValueSet`
- ▶ `WbemScripting.SWbemSink`



Objektmodell

Bindung auch an Klassen

WbemScripting-Klassen

Abbildung 14.8: Objektmodell der WMI-Komponente

Tabelle 14.8:
Klassen der WMI-
COM-Komponente

Klasse	Erläuterung
SWbemLocator	Diese Klasse stellt eine (von mehreren) Möglichkeiten dar, die Verbindung zu einem WMI-Server aufzubauen und Zugriff auf einen Namespace zu nehmen. Diese Klasse ist instanzierbar durch <code>WbemScripting.SwbemLocator</code> .
SWbemObject	SWbemObject ist die zentrale Meta-Klasse für den Zugriff auf WMI-Instanzen und WMI-Klassen.
SWbemObjectSet	Eine Collection von Objekten des Typs <code>SWbemObject</code> .
SWbemServices	Ein <code>SWbemServices</code> -Objekt repräsentiert einen WMI-Namespace als Ganzes.
SWbemMethod	Repräsentiert eine Methode in einem MO
SWbemMethodSet	Collection aller Methoden in einem MO
SWbemProperty	Repräsentiert ein Attribut in einem MO
SWbemPropertySet	Collection aller Attribute in einem MO
SWbemObjectPath	Dient dem Lesen und Schreiben von WMI-Pfaden
SWbemNamedValue	Ein <code>SWbemNamedValue</code> speichert ein Attribut-Wert-Paar. Die Klasse besitzt nur zwei Attribute (<code>Name</code> und <code>Value</code>).
SWbemNamedValueSet	Liste von <code>SWbemNamedValue</code> -Objekten, repräsentiert also eine Liste von Attribut-Wert-Paaren. Dieses Instrument wird z.B. eingesetzt, um die Schlüsselwerte eines MOs zu ermitteln und um bei asynchronen Aufrufen Informationen an die Ereignisbehandlungsroutinen zu übermitteln.
SWbemPrivilege	Repräsentiert ein einzelnes Privileg
SWbemPrivilegeSet	Liste aller Privilegien
SWbemQualifier	Repräsentiert einen Qualifier
SWbemQualifierSet	Repräsentiert eine Liste von Qualifiern
SWbemSecurity	Sicherheitseinstellungen, die aber auch über den WMI-Pfad vorgenommen werden können. Die Klassen <code>SWbemLocator</code> , <code>SWbemServices</code> , <code>SWbemObject</code> , <code>SWbemObjectSet</code> , <code>SWbemObjectPath</code> , <code>SWbemLastError</code> und <code>SWbemEventSource</code> besitzen ein Attribut <code>Security_</code> vom Typ <code>SWbemSecurity</code> .
SWbemEventSource	Ein Objekt dieser Klasse ist das Ergebnis der Methode <code>ExecNotificationQuery()</code> aus der Klasse <code>SWbemServices</code> . Nach der Ausführung einer Aktion dient das <code>SWbemEventSource</code> -Objekt dazu, die auftretenden Ereignisse nacheinander abzugreifen.

Objektmodell der WMI-Komponente

Klasse	Erläuterung
SWbemLastError	Informationen über den letzten Fehler. Der Aufbau der Klasse entspricht exakt dem der Klasse SWbemObject.
SWbemSink	SWbemSink dient COM-Clients dazu, Benachrichtigungen von Ereignissen und im Rahmen von asynchronen WMI-Operationen zu empfangen.

Tabelle 14.8:
Klassen der
WMI-COM-
Komponente
(Forts.)

Änderungen in Whistler

In Whistler (Windows XP/Windows.NET) wurden drei dieser Klassen erweitert:

- SWbemServices (Erweitert um zahlreiche Methoden zur Klasse SWbemServicesEx)
- SWbemObject (Erweitert um zahlreiche Methoden zur Klasse SWbemObjectEx)
- SWbemSink (Erweitert um eine Methode)

Neue Methoden



Abbildung 14.9:
Ansicht der Klasse
»SWbemObject« in
der Typbibliothek der
WMI-COM-Komponente

Typbibliothek

Die Typbibliothek definiert nur die Meta-Klassen der WMI-COM-Komponente, nicht aber die Klassen der Managed Objects. Eine solche Definition in der Typbibliothek hätte auch wenig Wert, da das Konzept eines Meta-

WMI-Typbibliothek

Objektmodells eine Form des späten Bindens ist. Entwicklungsumgebungen können IntelliSense-Unterstützung nur für die Mitglieder der Meta-Klassen anbieten. Eine Unterstützung für die Funktionen der Managed Objects wäre zwar über einen Zugriff auf das WMI-Schema denkbar, ist aber heute leider noch in keiner Entwicklungsumgebung implementiert.

NET und XML

Zukunft WMI wird in zukünftigen Versionen XML als Datenformat unterstützen. Im .NET-Framework ist das WMI-Meta-Objektmodell Bestandteil der .NET Base Class Library (BCL) in Form des Namespaces `System.Management`. In diesem Zusammenhang ist bei Microsoft auch der Begriff *WMI+* gefallen (passend dazu, dass zeitweise auch von ASP+ und ADO+ die Rede war).

14.2.1 Die Meta-Klasse `SWbemObject`

SWbemObject `SWbemObject` ist die zentrale Klasse von WMI, deren Instanzen Proxy-Objekte für WMI-MOs sind. `SWbemObject` ermöglicht den direkten Zugriff auf die Attribute und Methoden der MOs. Die Klasse `SWbemObject` bietet aber eine Reihe von Attributen und Methoden zur Verwaltung der Managed Objects. Alle diese Mitglieder enden auf einen Unterstrich »_«, um sie von den Mitgliedern der Managed Objects abzugrenzen.

Ein `SWbemObject`-Objekt besitzt eine `Properties-Collection`, in der die Attribute des Managed Objects gelistet werden, eine `Methods-Collection` für die Methoden des MOs und eine `Qualifier-Collection` für Qualifier-Objekte. Ein Qualifier ist die Beschreibung eines Attributs oder einer Methode.

Tabelle 14.9:
Attribute von
`SWbemObject`

Attribut	Erläuterung
<code>Derivation_</code>	Entspricht dem Systemattribut <code>_Derivation</code> , enthält also die Vererbungshierarchie der Klasse, zu der das Objekt gehört
<code>Properties_</code>	Verweis auf die zugehörige <code>SWbemPropertySet-Collection</code> , die alle Attribute der MO-Klasse enthält
<code>Methods_</code>	Verweis auf die zugehörige <code>SWbemMethodSet-Collection</code> , die alle Methoden der MO-Klasse enthält
<code>Qualifiers_</code>	Verweis auf die zugehörige <code>SWbemQualifierSet-Collection</code>
<code>Path_</code>	Verweis auf ein <code>SWbemObjectPath</code> -Objekt, das den WMI-Pfad des MOs enthält
<code>Security_</code>	Verweis auf das zugehörige <code>SWbemSecurity</code> -Objekt

Methode	Erläuterung
Associators_() AssociatorsAsync_()	Liefert einen SWbemObjectSet aller mit diesem MO assoziierten MOs. Dies sind alle Assoziationen, bei denen das aktuelle MO der Ausgangspunkt ist.
References_() ReferencesAsync_()	Liefert einen SWbemObjectSet aller MOs, die auf dieses MO verweisen. Dies sind alle Assoziationen, bei denen das aktuelle MO der Endpunkt ist.
Clone_()	Erzeugt eine Kopie des SWbemObjects
CompareTo_()	Prüft bei zwei SWbemObject-Instanzen, ob sie an das gleiche MO gebunden sind
Delete_() DeleteAsync_()	Löscht das an SWbemObject gebundene MO
ExecMethod_() ExecMethodAsync_()	Ausführen einer Methode auf diesem MO
GetObjectText_()	Liefert die Beschreibung des MOs in MOF-Syntax. Für eine Instanz enthält die MOF-Beschreibung alle konkreten Werte.
Instances_() InstancesAsync_()	Liefert einen SWbemObjectSet der Instanzen einer WMI-Klasse. Diese Methode ist nur verfügbar, wenn das MO eine Klasse ist.
Put_() PutAsync_()	Speichert die Änderungen im MO ab.
SpawnDerivedClass_()	Erzeugt eine neue, abgeleitete Klasse von der aktuellen WMI-Klasse. Diese Methode ist natürlich nur für MOs verfügbar, die Klassen sind.
SpawnInstance_()	Erzeugt eine neue Instanz einer WMI-Klasse. Diese Methode ist natürlich nur für MOs verfügbar, die Klassen sind.
Subclasses_() SubclassesAsync_()	Liefert einen SWbemObjectSet der Unterklasse einer WMI-Klasse (nur für MOs verfügbar, die WMI-Klassen sind)

Tabelle 14.10:
Methoden von
SWbemObject. Zur
Vereinfachung sind
die entsprechenden
asynchronen Metho-
den ohne weiteren
Kommentar mitge-
nannt.

14.2.2 SWbemServices

Ein SWbemServices-Objekt repräsentiert einen WMI-Namespace als Ganzes. Sie erhalten ein SWbemServices-Objekt, wenn Sie einen WMI-Moniker ohne Angabe einer konkreten Klasse oder Instanz verwenden.

SWbemServices

```
Set objServ = GetObject("winmgmts:")
```

Viele Methoden von SWbemServices entsprechen Methoden aus der Klasse SWbemObject, wobei bei SWbemServices natürlich ein gültiger WMI-Pfad angegeben werden muss, da sonst der Bezug nicht klar ist. SWbemServices bietet nur eine Eigenschaft an: Security_ ist ein Verweis auf das SWbemSecurity-Unterobjekt.

*Tabelle 14.11:
Methoden von
SWbemServices. Die
zweite Spalte gibt
an, ob die Methode
ein Einzelobjekt,
einen ObjectSet oder
gar keinen Wert
zurückgibt. Zur
Vereinfachung ist
die eventuell vor-
handene asynchrone
Variante einer
Methode jeweils
durch einen Quer-
strich getrennt
genannt.*

Methoden	Ergebnis	Erläuterung
AssociatorsOf() AssociatorsOfAsync()	ObjectSet / Kein	Liefert die Associators_-Collec- tion eines MOs
Delete() DeleteAsync()	Kein	Löscht eine Instanz oder Klasse
ExecMethod() ExecMethodAsync()	Object / Kein	Führt eine Methode auf einem Objekt aus
ExecNotificationQuery() ExecNotificationQue- ryAsync()	EventSource / Kein	Ausführung einer Event Query. Das Ergebnis ist ein SWbemEvent Source-Objekt.
ExecQuery() ExecQueryAsync()	ObjectSet / Kein	Ausführung einer WQL-Data Query oder Schema Query
Get() GetAsync()	Object Kein	Zugriff auf ein einzelnes MO
InstancesOf() InstancesOfAsync()	ObjectSet / Kein	Liefert die Menge der Instanzen einer Klasse
ReferencesTo() Reference- sToAsync()	ObjectSet / Kein	Liefert die References_-Collec- tion eines MOs
SubclassesOf() SubclassesOfAsync()	ObjectSet / Kein	Liefert die Subclasses_-Collec- tion eines MOs

14.3 Arbeit mit der WMI-Komponente

Dieses Kapitel stellt verschiedene Vorgehensweisen in WMI an konkreten Beispielen vor.

14.3.1 Zugriff auf lokale Managed Objects

Es gibt folgende Möglichkeiten, ein SWbemObject an ein Managed Object zu binden:

- Direkter Zugriff** ► direkter Zugriff auf ein Einzelobjekt mit GetObject() und dem WMI-Pfad


```
Set objWO = GetObject("winmgmts:Win32_ComputerSystem='MARS'")
```
- SWbemServices.Get()** ► Zugriff auf ein Einzelobjekt über ein SWbemServices-Objekt. Die SWbemServices-Klasse stellt eine Methode Get() bereit, mit der ein WMI-Objekt über den relativen Pfad angesprochen werden kann.


```
Set objServ = GetObject("winmgmts:")
Set objWO = objServ.Get("Win32_ComputerSystem='MARS'")
```
- InstancesOf()** ► Einlesen aller Instanzen einer Klasse in ein SWbemObjectSet-Objekt. Die Methode InstancesOf() liefert ein SWbemObjectSet-Objekt, das eine Collection von SWbemObject ist. Die Collection kann mit For...Each durchlau-

fen werden und bietet auch eine `Item()`-Methode. Dabei ist aber zu beachten, dass nicht nur der Wert des Schlüsselattributs anzugeben ist, sondern der komplette relative Pfad inklusive Klassenname. Indexwerte sind bei `Item()` nicht möglich!

```
Set objServ = GetObject("winmgmts:")
Set objOSet = objServ.InstancesOf("Win32_ComputerSystem")
' -- Zugriff auf WMI-Objekt
Set objWO = objOSet.Item("Win32_ComputerSystem='MARS'")
```

- Die vierte Möglichkeit ist die Ausführung einer WQL-Abfrage über die Methode `ExecQuery()` aus der Klasse `SWbemServices`. Das Ergebnis der Abfrage ist ebenfalls ein `SWbemObjectSet`-Objekt.

WQL-Query

```
Set objServ = GetObject("winmgmts:")
Set objOSet = _
objServ.ExecQuery("select * from Win32_ComputerSystem")
Set objWO = objOSet.Item("Win32_ComputerSystem='MARS'")
```

Das nächste Listing zeigt alle vier Möglichkeiten am Beispiel des `Win32_ComputerSystem`-Objekts des Rechners »MARS«. Dabei werden nur im ersten Fall mehrere Eigenschaften des `Win32_ComputerSystem`-Objekts ausgegeben.

Mit der vorgestellten Syntax können nur die Computerinformationen des lokalen Systems ermittelt werden. Trotzdem muss stets der Name des lokalen Systems angegeben werden (hier: »MARS«). Entspricht der Name nicht dem Namen des lokalen Systems, wird das WMI-Objekt nicht gefunden. Dies ist damit zu erklären, dass die gesamte Architektur darauf ausgelegt ist, dass es mehrere Instanzen einer Klasse gibt. Im Fall `Win32_ComputerSystem` gibt es auf jedem Computer natürlich nur eine Instanz, die aber trotzdem stets über ihren Namen angesprochen werden muss. Die Verfahren zum Zugriff auf entfernte Systeme werden später in diesem Unterkapitel vorgestellt.



```
Dim objWO ' As WbemScripting.SWbemObject
Dim objServ ' As WbemScripting.SWbemServices
Dim objOSet ' As WbemScripting.SWbemObjectSet
' --- 1. Möglichkeit -----
' -- Zugriff auf WMI-Objekt
Set objWO = GetObject("winmgmts:Win32_ComputerSystem='MARS'")
' -- Ausgabe von Eigenschaften des Managed Objects
say "Der Computer heißt: " & objWO.Caption
say "Der Computer gehört: " & objWO.PrimaryOwnerName
say "Domain: " & objWO.domain
say "Typ: " & objWO.SystemType
' --- 2. Möglichkeit -----
' -- ServiceObject ermitteln
Set objServ = GetObject("winmgmts:")
```

Windows Management Instrumentation (WMI)

```
' -- Zugriff auf WMI-Objekt
Set objWO = objServ.Get("Win32_ComputerSystem='MARS'")
say "Der Computer heit: " & objWO.Caption

' --- 3. Mglichkeit -----
' -- ServiceObject ermitteln
Set objServ = GetObject("winmgmts:")
' -- Zugriff auf WMI-Objekt
Set objOSet = objServ.InstancesOf("Win32_ComputerSystem")
' -- Zugriff auf WMI-Objekt
Set objWO = objOSet.Item("Win32_ComputerSystem='MARS'")
say "Der Computer heit: " & objWO.Caption
' --- 4. Mglichkeit -----
' -- ServiceObject ermitteln
Set objServ = GetObject("winmgmts:")
' -- Ausfhrung einer WQL-Abfrage
Set objOSet = objServ.ExecQuery("select * from Win32_ComputerSystem")
' -- Zugriff auf WMI-Objekt
Set objWO = objOSet.Item("Win32_ComputerSystem='MARS'")
say "Der Computer heit: " & objWO.Caption
```

Listing 14.4: Vier Wege zu einem WMI-Objekt [wmi_computerinfo.wsf]

14.3.2 Zugriff auf Schemainformationen

Schemazugriff via Objekt

Die Systemattribute stehen leider nicht direkt zur Verfgung. Die wichtigsten Systemattribute werden ber das Unterobjekt `Path_` vom Typ `SWbemObjectPath` geliefert. Im nachfolgenden Beispiel wird neben dem Anzeigenamen und dem Klassennamen durch `GetObjectText_` auch die MOF-Beschreibung ausgegeben, die auf einen Blick die Fhigkeiten des MOs liefert. Ersetzen Sie den Computernamen durch Ihren eigenen. Bitte beachten Sie, dass die Syntax in dieser Form keinen Remote-Zugriff ermglicht: Sie mssen den Namen des lokalen Computers eintragen.

```
' -- Zugriff auf ein WMI-Objekt
Set objWO = GetObject("winmgmts:Win32_ComputerSystem='MARS'")
' -- Ausgabe von Systemeigenschaften
say "WMI-Pfad: " & objWO.Path_.DisplayName
say "ist eine Instanz der Klasse: " & objWO.Path_.Class
say objWO.GetObjectText_
```

Listing 14.5: Ausgabe von Systemattributen und MOF-Beschreibung am Beispiel eines WMI-Objekts [wmi_sysatt_objekt.wsf]

Schemazugriff via Klasse

Whrend im vorhergehenden Beispiel an eine Instanz gebunden wurde, knnen Sie auch ber den WMI-Pfad direkt an eine Klasse binden. Dies ist im nchsten Skript am Beispiel der Klasse `Win32_LogicalDisk` gezeigt.

Arbeit mit der WMI-Komponente

```
' -- Zugriff auf eine WMI-Klasse
Dim cl ' As WbemScripting.SWbemObject
Set class = GetObject("winmgmts:Win32_LogicalDisk")
' -- Ausgabe von Systemeigenschaften
say "WMI-Pfad: " & cl.Path_.DisplayName
say "Ist eine Klasse von Typ: " & cl.Path_.class
say cl.GetObjectText_
```

Listing 14.6: Ausgabe von Systemattributen und MOF-Beschreibung am Beispiel der Klasse Win32_LogicalDisk [wmi_sysatt_klasse.wsf]

Liste der Attribute und Methoden

Über die Collections `Properties_` und `Methods_`, das Unterobjekt `Path_` sowie über dessen untergeordnete Collection `Keys` können Schemainformationen eines WMI-Objekts ermittelt werden. Die `Keys`-Collection enthält Objekte des Typs `WbemScripting.SWbemNamedValue`. Die Klasse `WbemScripting.SWbemNamedValue` verfügt nur über die Attribute `Name` und `Value`.

**Properties_ ,
Methods_**

`wmi_list_schema(obj)` ist eine Hilfsroutine, die das Schema für ein übergebenes Objekt des Typs `SWbemObject` auflistet. `list_schema()` bedient sich der Hilfsroutine `CIM_TypeAsString()`, um den Datentyp als Zeichenkette auszugeben.

wmi_list_schema()

```
Sub wmi_list_schema(obj)
'obj As WbemScripting.SWbemObject
Dim objProp ' As WbemScripting.SWbemProperty
Dim objmeth ' As WbemScripting.SWbemMethod
Dim objPath ' As WbemScripting.SWbemObjectPath
Dim objNV ' As WbemScripting.SWbemNamedValue
' -- Zugriff auf das Path-Unterobjekt
Set objPath = obj.Path_
' -- Ausgabe von Systemeigenschaften
say "Klassenname: " & objPath.Class
say "Pfad: " & objPath.DisplayName
say "relativer Pfad: " & objPath.RelPath
say "Namespace: " & objPath.Namespace
say "Server: " & objPath.Server
say "IsClass: " & objPath.IsClass

say "----- Attribute:"
For Each objProp In obj.Properties_
say objProp.name & " (" & CIM_TypeAsString(objProp) & _
"); Ursprung: " & objProp.Origin
Next
say "----- Schlüsselattribute:"
For Each objNV In objPath.Keys
say objNV.name
Next
say "----- Methoden:"
```

Windows Management Instrumentation (WMI)

```
For Each objmeth In obj.Methods_  
    say objmeth.name & "; Ursprung:" & objmeth.Origin  
Next  
End Sub
```

*Listing 14.7: Hilfsfunktion zur Ausgabe des Schemas eines WMI-Objekts
[wmi_list_schema.wsfl]*

```
Function CIM_TypeAsString(Property)  
' -- Fallunterscheidung mit CIMType  
Select Case Property.CIMType  
    Case 19: CIM_TypeAsString = "uint32"  
    Case 16: CIM_TypeAsString = "sint8"  
    Case 17: CIM_TypeAsString = "uint8"  
    Case 2:  CIM_TypeAsString = "sint16"  
    Case 18: CIM_TypeAsString = "uint16"  
    Case 3:  CIM_TypeAsString = "uint32"  
    Case 20: CIM_TypeAsString = "sint64"  
    Case 21: CIM_TypeAsString = "uint64"  
    Case 4:  CIM_TypeAsString = "real32"  
    Case 5:  CIM_TypeAsString = "real64"  
    Case 11: CIM_TypeAsString = "boolean"  
    Case 8:  CIM_TypeAsString = "string"  
    Case 101: CIM_TypeAsString = "datetime"  
    Case 103: CIM_TypeAsString = "char16"  
    Case 102: CIM_TypeAsString = "ref"  
        Set qualifier = Property.Qualifiers_("cimtype")  
        StrongRefArray = Split(qualifier.Value, ":")  
        If (UBound(StrongRefArray) > 0) Then  
            CIM_TypeAsString = CIM_TypeAsString & " " & _  
                StrongRefArray(1)  
        End If  
    Case 13  
        CIM_TypeAsString = "object"  
        Set qualifier = Property.Qualifiers_("cimtype")  
        StrongObjArray = Split(qualifier.Value, ":")  
        If (UBound(StrongObjArray) > 0) Then  
            CIM_TypeAsString = CIM_TypeAsString & " " & _  
                StrongObjArray(1)  
        End If  
    Case Else  
        MsgBox "Fehler: CIMType=" & Property.CIMType  
End Select  
' -- Ist das ein Array?  
If Property.IsArray = True Then  
    CIM_TypeAsString = "Array of " & CIM_TypeAsString
```

Arbeit mit der WMI-Komponente

```
End If  
End Function
```

Listing 14.8: Die Funktion CIM_TypeAsString() konvertiert einen CIM-Type in einen String. [CIM_TypeAsString.wsf]

Das nächste Listing testet WMI_list_schema() mit einer Klasse und einem Objekt.

```
Dim obj 'As WbemScripting.SWbemObject  
Dim class 'As WbemScripting.SWbemObject  
' -- Schema einer Klasse  
Set class = GetObject("winmgmts:Win32_LogicalDisk")  
wmi_list_schema class  
' -- Schema eines Objekts  
Set obj = GetObject("winmgmts:Win32_LogicalDisk='C:'")  
wmi_list_schema obj
```

Listing 14.9: Ausgabe des Schemas einer Klasse und eines Objekts [wmi_Schemaausgabe.wsf]

Die Ausgabe dieses Skripts sieht in gekürzter Form so aus:

```
Klassenname: Win32_LogicalDisk  
Pfad:  
WINMGMTS:{authenticationLevel=pktPrivacy,impersonationLevel=impersonate}!\MARS\root\cimv2:Win32_LogicalDisk.DeviceID="C:"  
relativer Pfad: Win32_LogicalDisk.DeviceID="C:"  
  
Namespace: root\cimv2  
Server: MARS  
IsClass: Falsch  
----- Attribute:  
Access (); Ursprung:CIM_StorageExtent  
Availability (); Ursprung:CIM_LogicalDevice  
BlockSize (); Ursprung:CIM_StorageExtent  
Caption (); Ursprung:CIM_ManagedSystemElement  
...  
DeviceID (); Ursprung:CIM_LogicalDevice  
...  
----- Schlüsselattribute:  
DeviceID  
----- Methoden:  
SetPowerState; Ursprung:CIM_LogicalDevice  
Reset; Ursprung:CIM_LogicalDevice
```

Listing 14.10: Ausgabe des Skripts aus Listing 5.173

14.3.3 Zugriff auf entfernte Systeme

WMI-Fernzugriff Für den Zugriff auf entfernte Systeme gibt es zwei Wege:

- Spezifikation des entfernten Systems über einen WMI-Pfad

```
Set objServ = GetObject("winmgmts://sonne2000")
```
- Alternativ dazu kann auch erst ein Locator-Objekt instanziiert werden, um dann mit der Methode `ConnectServer()` die Verbindung aufzunehmen.

```
Set objLoc = CreateObject("WbemScripting.SWbemLocator")
Set objServ = objLoc.ConnectServer("Mars")
```

ConnectServer() Die Methode `ConnectServer()` ermöglicht auch die Angabe eines Benutzernamens, eines Kennworts und eines Namespaces. Benutzername und Passwort können für lokale Aufrufe nicht angegeben werden!

```
Server = "Sonne2000"
UserName = "Administrator"
Password = "ds9"
namespace = "root\CIMV2"
Set objLoc = CreateObject("WbemScripting.SWbemLocator")
Set objServ = objLoc.ConnectServer(Server, namespace, UserName, _
Password)
Const MBFaktor = 1048576 ' 1024 * 1024
' -- Instanzen einlesen
Set objOSet = objServ.InstancesOf("Win32_logicaldisk")
' -- Instanzen auflisten
For Each objWO In objOSet
    say objWO.Name & " " & objWO.Description & " (" & _
Int(objWO.FreeSpace / MBFaktor) & " MB free)"
Next
```

*Listing 14.11: Abfrage des freien Speichers aller Laufwerke auf einem entfernten System
[wmi_laufwerk_remote.wsf]*

14.3.4 Änderung von Attributwerten

Werte ändern Einige, aber nicht alle Attribute von Managed Objects können geändert werden. Viele Attributwerte sind nur über Methodenaufrufe beeinflussbar. Beim Schreibzugriff auf Attribute der Managed Objects ist zu beachten, dass diese erst persistent werden, nachdem die Methode `Put_()` aufgerufen wurde. Dies ist bei Meta-Objektmodellen wie ADSI und WMI üblich. In ADSI heißt die entsprechende Methode allerdings `SetInfo()`.

```
Dim objDisk ' As WbemScripting.SWbemObject
' -- Zugriff auf Objekt
Set objDisk =
GetObject("winmgmts:{impersonationLevel=impersonate}!Win32_LogicalDisk
='C:')
```

```
' -- Wert zuweisen
objDisk.VolumeName = "Laufwerk C"
' -- Änderungen persistent machen
objDisk.Put_
```

*Listing 14.12: Ändern der Laufwerksbezeichnung für Laufwerk C:
[wmi_laufwerksnamenänderung.wsf]*

14.3.5 Aufruf von Methoden

Auch für den Methodenaufruf haben Sie die Wahl zwischen dem direkten Aufruf oder der umständlichen Variante über die `Methods_`-Collection. Die folgenden Beispiele zeigen die einfachere Variante.

**WMI-Methoden-
aufrufe**

Prozesse beenden

Im ersten Beispiel werden alle Prozesse, die `NOTEPAD.EXE` heißen, gelöscht. Dazu wird zunächst eine WQL-Abfrage ausgeführt. Auf allen Objekten in dem `SWbemObjectSet` wird dann die Methode `Terminate()` ausgeführt.

Prozesse

```
Dim obj ' As WbemScripting.SWbemObject
Dim obj0Set ' As WbemScripting.SWbemObjectSet
Dim a ' Zähler
a = 0
Set obj0Set = GetObject("winmgmts:").ExecQuery("select * from
Win32_Process where Name='notepad.exe'")
For Each obj In obj0Set
    obj.Terminate
    a = a + 1
Next
say a & " Instanzen des Prozesses beendet!"
```

Listing 14.13: Beenden aller Instanzen von Notepad.exe [WMI_Prozessloeschen.wsf]

Ausführung einer statischen Methode

Das bereits zuvor erwähnte Beispiel einer statischen Methode, die auf einer Klasse und nicht auf einer Instanz ausgeführt wird, ist die Konstruktor-methode `Create()` in der Klasse `Win32_Process`. Das folgende Skript startet `NOTEPAD.EXE`.

**Methodenaufrufe
auf Klassenebene**

```
Dim class ' As WbemScripting.SWbemObject
Dim Ergebnis
Dim ProcessID
' --- Neuen Notepad-Prozess erzeugen
Set class = GetObject("winmgmts:\\mars\root\cimv2:Win32_Process")
Ergebnis = class.create("notepad", Null, Null, ProcessID)
say "Rückgabewert von Create(): " & Ergebnis
say "Prozess-ID: " & ProcessID
```

Listing 14.14: Startet Notepad.exe [WMI_NeuerProzess.wsf]



Sie können mit diesem Skript auch auf einem entfernten System einen Prozess erzeugen. Dafür müssen Sie lediglich im WMI-Pfad nach \\ den Namen oder die IP-Adresse des entfernten Systems angeben. Die Prozesserschöpfung funktioniert auch dann, wenn an dem Computer niemand angemeldet ist. Jedoch wird die grafische Benutzeroberfläche erst nach der nächsten Anmeldung angezeigt.

Dienste starten und stoppen

NT-Dienste Das Skript im nächsten Listing gibt zunächst eine Liste aller Dienste mit dem Status und dem Startzustand aus. Im Vergleich zum Zugriff auf Dienste in ADSI fällt auf, dass die Zustände in Form von Strings zurückgegeben werden. Eine Umwandlung einer numerischen Konstante in eine für den Nutzer verständliche Zeichenkette entfällt also.



Danach experimentiert das Skript mit dem *Fax*-Dienst. Bitte beachten Sie folgende Besonderheiten:

- Ein direkter Zugriff auf das Attribut `StartMode` bleibt ohne Fehler, aber auch ohne Wirkung.

```
' So geht es nicht:
objW0.StartMode = "Automatic"
objW0.Put_
```

Sie müssen die Methode `ChangeStartMode()` verwenden.

- `StartMode` liefert u. a. den Wert »Auto« zurück. Sie müssen jedoch beim Aufruf von `ChangeStartMode()` den String »Automatic« übergeben. Dies ist eine sehr hässliche Inkonsistenz, die dazu führt, dass Sie sich bei der WMI-Programmierung nicht auf die Werte verlassen können, die Sie bei bestehenden Instanzen sehen. Wenigstens liefern der WBEM Object Browser und das CIM Studio nicht nur die Namen der Attribute, sondern auch eine Liste der möglichen Werte.

```
Dim objW0      ' As WbemScripting.SWbemObject
Dim objServ    ' As WbemScripting.SWbemServices
Dim obj0Set    ' As WbemScripting.SWbemObjectSet
Dim objLocator ' As WbemScripting.SWbemLocator
Const strServer = "MARS"
' -- Verbindungsaufbau
Set objLocator = CreateObject("WbemScripting.SWbemLocator")
Set objServ = objLocator.ConnectServer(strServer)
```

Liste aller Dienste

```
' -- Liste aller Dienste
Set obj0Set = objServ.ExecQuery("Select * From Win32_Service")
say "Status aller Dienste:"
For Each objW0 In obj0Set
    say objW0.Name & " (" & objW0.Description & "): " & objW0.State &
```

Arbeit mit der WMI-Komponente

```

", Startmode: " & objW0.StartMode
Next

' -- Veränderungen am Fax-Dienst...
Set objW0 = objServ.Get("Win32_Service='Fax'")
say "Dienst wird angehalten..."
objW0.PauseService
say "Dienst läuft weiter..."
objW0.ResumeService
say "Dienst wird gestoppt..."
objW0.StopService
say "Dienst neu gestartet..."
objW0.StopService
say "Setze Startmodus auf DISABLED..."
objW0.ChangeStartMode ("DISABLED")
say "Setze Startmodus auf MANUAL..."
objW0.ChangeStartMode ("MANUAL")
say "Setze Startmodus auf AUTO..."
objW0.ChangeStartMode ("Automatic")

```

Listing 14.15: Arbeit mit Diensten unter WMI [wmi_Services.wsfl]

Starten und Stoppen

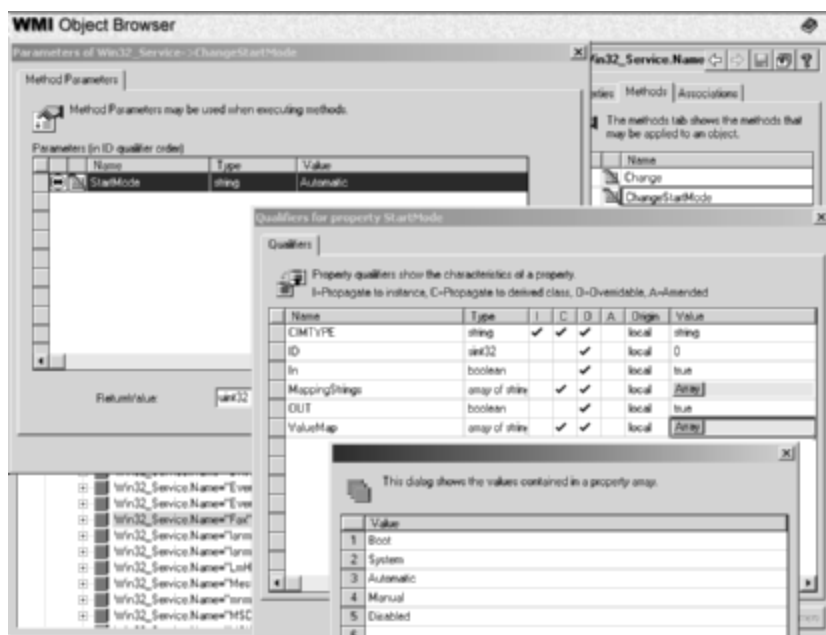


Abbildung 14.10:
Im WBEM Object Browser in verschiedenen Fenstern, hier alles auf einen Blick: Die Methode `ChangeStartMode()` aus dem WMI-Objekt »Fax« (Klasse `Win32_Service`) erwartet einen Parameter `StartMode`. Dazu gibt es im Schema eine `ValueMap` mit den fünf möglichen Werten.



Für die Verwendung von Attributen und Methoden gibt es neben dem direkten Zugriff auch die Möglichkeit, über die `Properties_-` bzw. `Methods_-Collection` auf die Mitglieder zuzugreifen. Dies ist jedoch umständlich, wie nachfolgende Code-Ausschnitte belegen.

```
wert = Obj.Properties_("Attributname") und
Obj.Properties_("Attributname") = wert.

Set method = process.Methods_("Create")
Set inParam = method.inParameters.SpawnInstance()
inParam.CommandLine = "calc.exe"
Set outParam = process.ExecMethod_("Create", inParam)
```

14.3.6 Ausgabe eines ObjectSets

Arbeit mit Objekt-mengen

Die Routinen `WMI_List0()` und `WMI_List0Set()` dienen dazu, die Attribute eines gebundenen MOs bzw. die in einem `SWbemObjectSet` enthaltenen MOs auszugeben.

```
Sub wmi_List0Set(obj0Set)
' obj0Set As WbemScripting.SWbemObjectSet
Dim objprop ' As WbemScripting.SWbemProperty
say "0Set enthält " & obj0Set.Count & " MOs"
' -- Iteration über alle MOs
For Each objW0 In obj0Set
    say "----- "
    WMI_List0 objW0
Next
End Sub
```

Listing 14.16: Hilfsroutine zur Ausgabe eines `SWbemObjectSet`s [wmi_List0Set.wsf]

```
Sub wmi_List0(objW0)
' objW0 As WbemScripting.SWbemObject
Dim objprop ' As WbemScripting.SWbemProperty
' -- Iteration über alle Eigenschaften
For Each objprop In objW0.Properties_
    say objprop.Name & "=" & flat(objprop.Value)
Next
End Sub
```

Listing 14.17: Ausgabe aller Attribute eines einzelnen MOs. `flat()` wird verwendet, da mehrwertige Attribute möglich sind. [wmi_List0.wsf]

14.3.7 WQL-Queries im Detail

WMI unterstützt drei Typen von WQL-Anfragen:

- ▶ Anfragen nach Instanzen (*Data Queries*)
- ▶ Anfragen nach Schemainformationen (*Schema Queries*)
- ▶ Definitionen von Eventfiltern (*Event Queries*), die nach Änderungen von Klassen und Instanzen fragen

WMI-Suchanfragen

WMI_Query() ist eine Hilfsroutine, die eine WQL-Abfrage ausführt und das Ergebnis mit Hilfe der Hilfsroutine WMI_List0Set() ausgibt.

WMI_Query()

```
Sub wmi_Query(query)
Dim objServ ' As WbemScripting.SWbemServices
Dim obj0Set ' As WbemScripting.SWbemObjectSet
Set objServ = GetObject("winmgmts:")
Set obj0Set = objServ.ExecQuery(query)
WMI_List0Set obj0Set
End Sub
```

Listing 14.18: Ausführen und Ausgeben einer WQL-Anfrage

Wenn Sie Queries verwenden, sollten Sie anschließend das Windows NT EventLog überprüfen. WMI schreibt eine Fehlermeldung in das Event-Log, wenn Probleme mit Ihrer Query auftreten.



Data Queries

Eine Data Query hat die allgemeine Form

Data Queries

```
SELECT attributliste FROM class WHERE bedingung
```

Dabei ist Class ein beliebiger WMI-Klassenname. Es ist möglich, die Ergebnismenge durch die Angabe von Attributnamen und die Verwendung einer FROM-Klausel hinsichtlich der Breite und Länge einzuschränken. Andere Schlüsselwörter werden nicht unterstützt.

Beispiele sind:

WQL	Erläuterung
Select * from Win32_Service where state='running' and startmode='manual'	Alle NT-Dienste, die laufen, aber manuell gestartet wurden
Select Name, CategoryId from Win32_ComponentCategory	Name und CATID aller Komponentenkategorien

Tabelle 14.12: Beispiele für WQL-Data Queries

Windows Management Instrumentation (WMI)

Tabelle 14.12:
Beispiele für
WQL-Data
Queries
(Forts.)

WQL	Erläuterung
select IPAddress from Win32_NetworkAdapterConfiguration where IPEnabled=TRUE	Das mehrwertige Attribut IPAddress einer Netzwerkkarte, 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

WMI_WQL-Anfragen()

```
WMI_Query "select IPAddress from Win32_NetworkAdapterConfiguration
where IPEnabled=TRUE"
WMI_Query "Select * from Win32_Service where state='running' and
startmode='manual'"
WMI_Query "Select Name, CategoryId from Win32_ComponentCategory"
WMI_Query "SELECT RecordNumber, Message FROM Win32_NTLogEvent where
Logfile='Application'"
WMI_Query "SELECT * FROM __InstanceModificationEvent WITHIN 5 WHERE
TargetInstance isa 'Win32_Service' AND TargetInstance.State='Stopped'"
```

Listing 14.19: Dieses Skript führt nacheinander die vier in der Tabelle beschriebenen Abfragen aus, die mit Hilfe von `WMI_Query()` ausgegeben werden.



Wollen Sie wissen, wie viele Dateien Sie in Ihrem Dateisystem haben? Bringen Sie aber bitte etwas Zeit und genügend RAM mit, um eine Antwort zu erhalten. Sie sollten bei diesen Datenmengen auf keinen Fall alle Attribute abrufen, sondern nur ein »schlankes« Attribut; es geht ja schließlich nur um die Anzahl der Objekte. Im folgenden Listing wurde das Boolean-Attribut `compressed` verwendet.

```
Set objServ = GetObject("winmgmts:")
Set objOSet = objServ.ExecQuery("select compressed from
CIM_DataFile")
say "Anzahl der Dateien: " & objOSet.Count
```

Schema-Queries

Schema-Queries

Anfragen zum Schema haben die Form

```
SELECT attributliste FROM META_CLASS WHERE bedingung
```

wobei hier `META_CLASS` ein feststehender Ausdruck ist. Andere Schlüsselwörter werden nicht unterstützt. Mit der `WHERE`-Klausel werden das zu beobachtende Managed Object und die in dem MO zu beobachtenden Attribute definiert. Schema-Queries können mit `WMI_Query(WQL)` ausgegeben werden.

Arbeit mit der WMI-Komponente

WQL	Erläuterung
<pre>SELECT * FROM meta_class WHERE __Class = "Win32_LogicalDisk" SELECT * FROM meta_class WHERE __this ISA "Win32_LogicalDisk"</pre>	Zugriff auf eine einzelne Klasse Zugriff auf von Win32_LogicalDisk abgeleitete Klasse

Tabelle 14.13:
Beispiele für
Schema-Queries.

Event Queries

Eine Event Query bezieht sich immer auf eine Eventklasse. Mit der WHERE-Klausel werden das zu beobachtende Managed Object (MO) und die in dem MO zu beobachtenden Attribute definiert.

Event Queries

```
SELECT * FROM eventklasse WHERE bedingung
```

Alle Eventklassen sind Unterklassen eines Eventtyps. Die Eventtypen wiederum sind Unterklassen der Klasse __Event. Sie sind in der Regel an dem führenden doppelten Unterstrich und der Endung auf Event erkennbar. WMI 1.5 unterscheidet vier Typen von Events.

Ereignistyp (Oberklasse)	Ereignisklasse
__ClassOperationEvent	__ClassCreationEvent __ClassDeletionEvent __ClassModificationEvent
__ExtrinsicEvent	__SystemEvent Win32_PowerManagementEvent
__InstanceOperationEvent	__InstanceCreationEvent __InstanceDeletionEvent __InstanceModificationEvent
__NamespaceOperationEvent	__NamespaceCreationEvent __NamespaceDeletionEvent __NamespaceModificationEvent

Tabelle 14.14:
WMI-Ereignisklassen. Die Ereignistypen sind die Oberklassen zu den rechts genannten Ereignisklassen.

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%).

Event Queries unterstützen als zusätzliche SQL-Schlüsselwörter WITHIN, GROUP BY und HAVING:

Zusätzliche Schlüsselwörter

- Dabei gibt WITHIN 10 das Polling-Intervall in Sekunden an.
- GROUP bündelt eine Anzahl von Einzelevents zu einem Gesamtevent.
- HAVING dient der Definition einer Bedingung innerhalb der Gruppierung.

Tabelle 14.15:
Beispiele für Event
Queries

WQL	Erläuterung
SELECT * FROM __InstanceModificationEvent WITHIN 5 WHERE TargetInstance ISA "Win32_Service" AND TargetInstance.State="Stopped"	Alle fünf Sekunden wird geprüft, ob ein Dienst den Status <i>Stopped</i> bekommen hat.
SELECT * FROM EmailEvent GROUP WITHIN 600 HAVING NumberOfEvents > 5	Wenn innerhalb von zehn Minuten mehr als fünf E-Mail-Ereignisse auftreten, wird dieses Ereignis ausgelöst.
SELECT * FROM __InstanceCreationEvent WHERE TargetInstance ISA "Win32_NTLogEvent" AND TargetInstance.Logfile="Application" OR TargetInstance.Logfile="System"	Jeder neue Eintrag in den Ereignisprotokollen <i>System</i> und <i>Application</i> löst einen Event aus.



Sie können eine Event Query nicht auf die für die anderen beiden Query-Typen beschriebene Weise mit wmi_query() ausführen, da eine Event Query keine direkten Ergebnisse liefert. Sie können sie durch semi-synchrone Verarbeitung per ExecNotificationQuery() ausführen. Die semi-synchrone Verarbeitung wird im nächsten Abschnitt vorgestellt.

14.3.8 Semi-synchrone Verarbeitung

ExecNotificationQuery()

Die Arbeit mit ExecNotificationQuery() aus der SWbemServices-Klasse wird als halb-synchrone Verarbeitung bezeichnet, da Ereignisse nicht durch Ereignisbehandlungsroutinen, sondern durch eine spezielle Wartefunktion in Empfang genommen werden.

ExecNotificationQuery() erwartet einen *WQL Event Query-String*. Ergebnis ist ein SWbemEventSource-Objekt, das die Methode NextEvent(zeit) anbietet. NextEvent() wartet die angegebene Anzahl von Millisekunden auf ein Ereignis. Sofern ein Ereignis eintritt, wird ein SWbemObject dafür geliefert. Wenn kein Ereignis eintritt, wird der Fehler »80043001: Zeitüberschreitung« ausgelöst.

```
Dim oServ ' As WbemScripting.SWbemServices
Dim oSource ' As WbemScripting.SWbemEventSource
Dim strAbfrage ' As String
Dim objEvent ' As WbemScripting.SWbemObject
Dim objTarget ' As WbemScripting.SWbemObject
```

Arbeit mit der WMI-Komponente

```
Const Wartezeit = 10 ' Sekunden
' -- Bindung mit Privilege Security Operator
Set oServ = GetObject("winmgmts:{(security)}")
strAbfrage = "SELECT * FROM __instancecreationevent WHERE
TargetInstance ISA 'Win32_NTLogEvent'"
' -- Ausführung der Abfrage
Set oSource = oServ.ExecNotificationQuery(strAbfrage)
On Error Resume Next
' -- Warten...
Set objEvent = oSource.NextEvent(Wartezeit * 1000)
' -- Fehlerabfrage
If Err <> 0 Then
    If Err.Number = &H80043001 Then      ' wbemErrTimedout
        say "Es ist nichts passiert!"
    Else                                ' Fehler!
        say "Fehler: " & Err.Description
    End If
Else ' Es ist etwas passiert!
    Set objTarget = objEvent.TargetInstance
    say objTarget.Message
End If
```

*Listing 14.20: Semi-synchrone Abfrage von neuen Ereignisprotokolleinträgen
[wmi_Enotification_Click.wsf]*

14.3.9 Asynchrone Aufrufe mit SWbemSink

Die Klasse `SWbemSink` dient COM-Clients dazu, Benachrichtigungen von Ereignissen und im Rahmen von asynchronen WMI-Operationen zu empfangen. Asynchron bedeutet, dass WMI nach der Ausführung einer Aktion die Kontrolle an den WMI-Client zurückgibt und die Ergebnisse per COM-Ereignis meldet. Ein asynchron arbeitender WMI-Client ist der aus der Sicht von WMI ein temporärer Event Consumer.

**Asynchrone
Vorgänge**

`SWbemSink` unterstützt vier COM-Ereignisse (die nicht mit den WMI-Ereignisklassen zu verwechseln sind) und eine einsame Methode: `Cancel()` dient dem Abbruch aller mit dem `SWbemSink`-Objekt verbundenen asynchronen Operatoren. `Cancel()` hat keine Parameter.

Jedes der in der nächsten Tabelle beschriebenen Ereignisse übergibt einen Parameter `Kontext`. `Kontext` ist eine Collection vom Typ `SWbemNamedValueSet`. Diese Collection ermöglicht es, Werte beim Start der asynchronen Bearbeitung an die Ereignisbehandlungsroutinen zu übergeben. Der Sinn liegt darin, dass beim gleichzeitigen Aufruf mehrerer asynchroner Vorgänge die Ereignisbehandlungsroutine die Möglichkeit erhält, den Auslöser dieses Ereignisses zu identifizieren. Freilich ist die Angabe eines `SWbemNamedValueSet` beim Start eines asynchronen Vorgangs kein Muss.

**Ereignisse für
synchrone
Verarbeitung**

Tabelle 14.16:
Ereignisse in der
Klasse *SWbemSink*

Event	Erläuterung
OnObjectReady (Objekt, Kontext)	Dieses Ereignis wird für jede einzelne Instanz der Ergebnismenge einer asynchronen Operation ausgelöst. Sie liefert jeweils ein <i>SWbemObject</i> .
OnObjectPut (Objekt, Kontext)	Bei asynchronen Schreibzugriffen wird dieses Ereignis für jedes gespeicherte Objekt ausgelöst.
OnCompleted (HResult, Last Error, Kontext)	Dieses Ereignis zeigt an, dass eine asynchrone Operation beendet ist. <i>HResult</i> liefert im Erfolgsfall eine 0, im Fehlerfall eine Fehlernummer. <i>LastError</i> ist ein Zeiger auf ein <i>SWbemLastError</i> -Objekt.
OnProgress (Anzahl, Aktuell, Nachricht, Kontext)	Dieses Ereignis zeigt den Fortschritt einer Operation an. Dabei wird die Gesamtzahl der abzuarbeitenden Operationen und die aktuell bereits abgearbeitete Anzahl übergeben sowie eine Nachricht, die den Fortschritt beschreibt. Dieses Ereignis wird nur ausgelöst, wenn beim Start der asynchronen Operation das Flag <i>wbemFlagSendStatus</i> (= 128) angegeben wurde.

```

Dim objWO ' As WbemScripting.SWbemObject
Dim objServ ' As WbemScripting.SWbemServices
' -- Dieses Ereignis wird für jedes Objekt gefeuert
Sub objWO_OnObjectReady(objObject, objAsyncContext)
    say (objObject.Name)
End Sub
' -- Dieses Ereignis wird nur am Ende gefeuert
Sub objWO_OnCompleted(HResult, objErrorObject, objAsyncContext)
    MsgBox "Alle Daten wurden ausgegeben!"
End Sub
' -- Hauptprogramm
Set objServ= GetObject("winmgmts:")
Set objWO= WScript.CreateObject("WbemScripting.SWbemSink","objWO_")
' -- asynchrone Ausführung starten
objServ.InstancesOfAsync objWO, "Win32_process"
msgbox "Skript wartet..."

```

Listing 14.21: WSH-Beispiel für einen asynchronen Aufruf (Liste der Prozesse ausgeben)
[_wmi_asynchronerAufruf.vbs]

Event Correlation

Ereignisse zusammenfassen

Whistler (Windows XP/Windows.NET) unterstützt die Aggregation von Events. So können ähnliche Ereignisse innerhalb eines bestimmten Zeitintervalls zu einem Ereignis zusammengefasst werden.

14.4 WMI ODBC-Treiber

Der WMI ODBC-Treiber ermöglicht die Ausführung von Datenbankabfragen via ODBC auf WMI-Providern. Der ODBC-Treiber unterstützt allerdings – wie die WMI Queries auch – nur den Lesezugriff mit dem SQL-Befehl Select. Außerdem unterstützt der Treiber nur ODBC 2.0 und daher kein Unicode. Der WMI ODBC-Treiber wird auch unter den Namen »WBEM ODBC Treiber« und »WMI ODBC Adapter« geführt.

Ein OLE DB-Provider für WMI ist noch nicht verfügbar. Microsoft hat einen solchen Provider aber für die nächste WMI-Hauptversion (2.0) angekündigt.

WMI via ODBC

OLE DB-Unterstützung

Der WMI ODBC-Treiber läuft nicht unter Windows 95/98/ME. Weder im NT4-WMI-Add-on noch unter Windows2000 wird der Treiber automatisch installiert. Sie installieren ihn durch ein separates Setup-Programm, das Sie auf jeder Windows2000 Server-CD im Verzeichnis \VALUEADD\MSFT\MGMT\WBEMODBC finden. Dieses Setup installiert den Treiber unter dem Namen »WBEM ODBC DRIVER« in der Version 1.50.1084.01 (vgl. ODBC-Einstellungen in der Systemsteuerung bzw. im Ordner VERWALTUNG unter Windows2000).



Abbildung 14.11:
Anzeige der installierten ODBC-Treiber

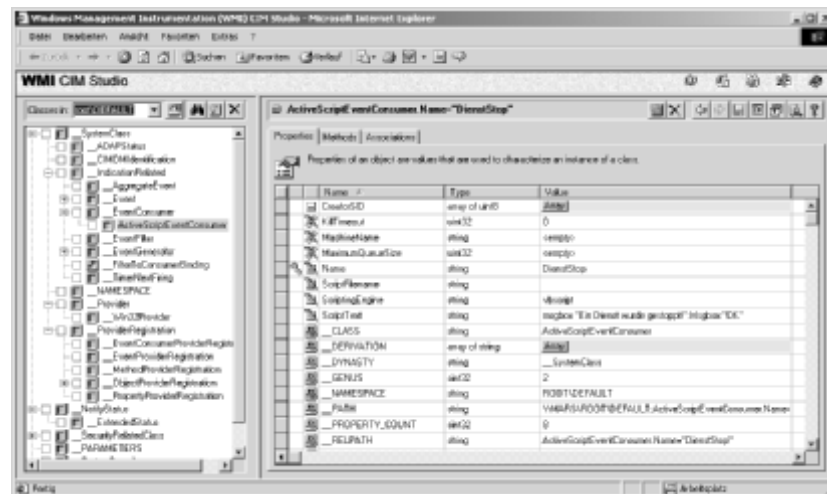
14.5 ActiveScriptEventConsumer

Bisher wurde in diesem Kapitel schon gezeigt, wie Sie ein Skript schreiben können, das auf Veränderungen in Managed Objects wartet. WMI stellt auf Basis der WMI-Ereignismechanismen noch einen eleganteren Weg bereit. Die Klasse ActiveScriptEventConsumer ist ein permanenter Event Consumer,

**Skript starten
beim Eintreten
von WMI-Ereignissen**

der in der Lage ist, beim Auftreten eines bestimmten Ereignisses ein Skript zu starten. In einer Instanz von `ActiveScriptEventConsumer` wird entweder direkt eine Folge von Skriptbefehlen oder ein Verweis auf ein WSH-Skript hinterlegt.

Abbildung 14.12:
Eine Instanz der
Klasse `ActiveScriptEventConsumer`
im
CIM Studio



Filter Wie üblich in WMI wird das zu überwachende Managed Object in Form einer Event Query spezifiziert. Die Event Query wird in einer Instanz von `__EventFilter` hinterlegt. Die Bindung einer Instanz von `ActiveScriptEventConsumer` an eine Instanz von `__EventFilter` kann mit dem WMI Event Registration Tool erfolgen.

Vorteil Der große Vorteil gegenüber anderen Methoden ist, dass Sie für die Reaktion auf ein Ereignis kein Skript im Hintergrund warten lassen müssen. Die Klasse `ActiveScriptEventConsumer` ermöglicht es vielmehr, ein Skript genau dann zu starten, wenn es wirklich benötigt wird.



Die Klasse `ActiveScriptEventConsumer` befindet sich im Namespace `/ROOT/DEFAULT`, nicht wie sonst üblich im Namespace `/ROOT/CIMV2`. Die Klasse ist aber im Standard nicht im CIM-Repository installiert. Mit der nachfolgend genannten Zeile nehmen Sie die Information des zugehörigen MOF-Files in das CIM Repository auf:

```
mofcomp.exe scrcons.mof
```

Das MOF-File `SCRCONS.MOF` gehört zum Standardinstallationsumfang von WMI und liegt im Verzeichnis `/WINNT/SYSTEM32/WBEM`. Erst danach können Sie die Klasse `ActiveScriptEventConsumer` verwenden.

ActiveScriptEventConsumer

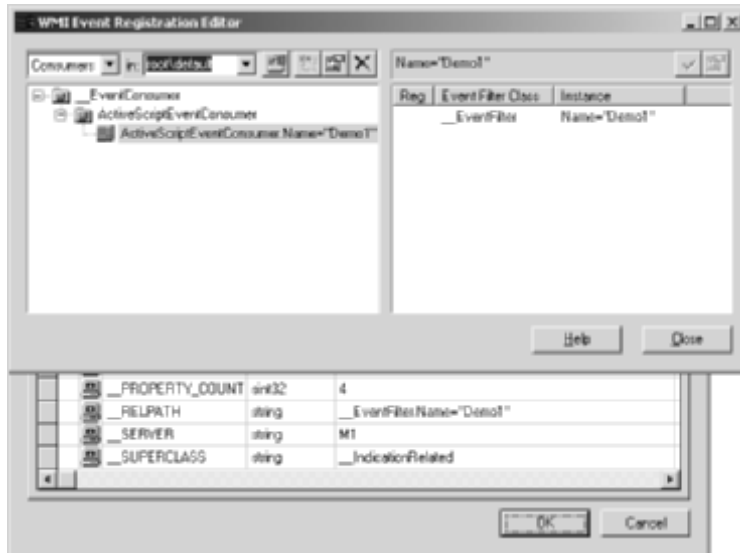


Abbildung 14.13:
Bindung einer
Instanz von Active-
ScriptEventConsu-
mer an einen Filter

14.5.1 WMI-ADSI-Integration

Die Funktionalitäten von WMI und ADSI (vgl. Kapitel 13) überschneiden sich zum Teil. Verzeichnisobjekte wie Computer, Benutzer, Gruppen und NT-Dienste können sowohl durch WMI als auch durch ADSI verwaltet werden. Die zur Verfügung stehenden Funktionalitäten sind aber keineswegs gleich, so dass innerhalb von Automatisierungslösungen oft beide Komponenten zum Einsatz kommen.

WMI versus ADSI

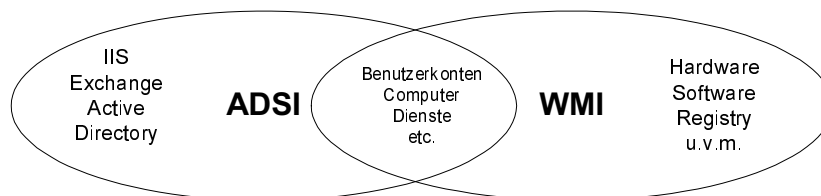


Abbildung 14.14:
Schnittmenge
zwischen ADSI
und WMI

Beide Komponenten besitzen keine gemeinsamen COM-Schnittstellen, so dass ein polymorpher Zugriff nicht möglich ist. Zwar nutzen beide Komponenten Moniker zur Identifikation; jedoch unterscheiden sich WMI-Pfade sehr stark von ADSI-Pfaden.

Unterschiede

WMI ADSI Extension

Hilfreich ist eine Komponente, die die Brücke zwischen ADSI und WMI schlägt. Diese Komponente heißt *WMI ADSI Extension*. Der Brückenschlag erfolgt zwischen der Active Directory-Klasse `Computer` und der WMI-Klasse `Win32_ComputerSystem`.

Brückenschlag

ADSI-Extension

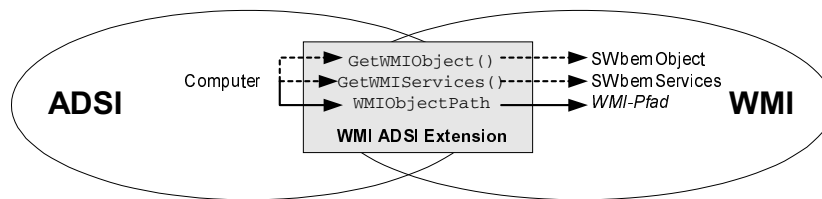
Die *WMI ADSI Extension* ist eine ADSI Namespace Extension für die Klasse *Computer* im LDAP-Namespace. Weitere Informationen zur Arbeitsweise von ADSI Namespace Extensions finden Sie in Kapitel 13).

Durch diese Extension wird die *Computer*-Klasse um ein Attribut und zwei Methoden erweitert. Alle drei Methoden haben keine Parameter.

Neue Mitglieder

- ▶ *WMIObjectPath* liefert den WMI-Pfad zu dem gegebenen ADS-Computer-Objekt.
- ▶ *GetWMIObject()* liefert eine Instanz der WMI-Klasse *Win32_ComputerSystem* zu dem gegebenen ADS-Computer-Objekt. Die Instanz der WMI-Klasse wird wie üblich durch eine Instanz der Meta-Klasse *WbemScripting.SWbemObject* gekapselt.
- ▶ *GetWMIServices()* liefert eine Instanz von *WbemScripting.SWbemServices* für das gegebene ADS-Computer-Objekt. Damit ist ein Zugriff auf in der Objekthierarchie untergeordnete WMI-Objekte möglich.

Abbildung 14.15:
Brückenschlag
zwischen ADSI und
WMI



Dieser Brückenschlag funktioniert nur für *Computer*-Objekte im Active Directory. *Computer*-Objekte aus dem WinNT-Provider bekommen diese zusätzlichen Mitglieder nicht. Hier bleibt nichts anderes übrig, als die Pfade von Hand umzusetzen. Auch unterstützt die *WMI ADSI Extension* nicht den umgekehrten Weg von einem *Win32_ComputerSystem* zu einem AD-Computer-Objekt.

Beispiel

In dem folgenden Skript wird zunächst ein *Computer*-Objekt via ADSI gebunden. Danach werden die drei oben vorgestellten Mitglieder verwendet.

```
Dim obj ' As IADs
Dim objWO ' As WbemScripting.SWbemObject
Dim objServ ' As WbemScripting.SWbemServices
' -- Zugriff auf ein Computer-Objekt via ADSI
Set obj = GetObject("LDAP://sonne2000/CN=MARS,CN=Computers,DC=IT-Visions,DC=DE")
' -- Ausgabe der Pfade
say "ADSI-Pfad: " & obj.AdsPath
say "WMI-Pfad: " & obj.WMIObjectPath
' -- Ermittlung eines Zeigers auf das MO für diesen Computer
```

Weitere WMI-Beispiele

```
Set objWO = obj.GetWMIObject
say "Systemtyp: " + objWO.Systemtype
say "Speicher: " & objWO.TotalPhysicalMemory
' --- Ermittlung eines Zeigers auf ein WMI-Service-Objekt
Set objServ = obj.GetWMIServices
Set objWO = objServ.Get("Win32_LogicalDisk.DeviceID=\"C:\"")
say "Freier Speicher auf C: " + objWO.FreeSpace
```

Listing 14.22: Nutzung der WMI ADSI Extension [WMI_ADSIEX.wsf]

```
ADSI-Pfad: LDAP://sonne2000/CN=MARS,CN=Computers,DC=IT-Visions,DC=DE
WMI-Pfad: WINMGMTS:{impersonationLevel=impersonate}! //MARS/root/
cimv2:Win32_ComputerSystem.Name="MARS"
Systemtyp: X86-based PC
Speicher: 267952128
Freier Speicher auf C: 764121088
```

Listing 14.23: Beispiel für die Ausgabe des vorherigen Skripts

14.6 Weitere WMI-Beispiele

Dieses Kapitel zeigt Ihnen noch einige ausgewählte Beispiele für den Einsatz von WMI.

Abmelden, Herunterfahren und Neustart

Mit der Methode `Reboot()` in der Klasse `Win32_OperatingSystem` kann ein System heruntergefahren werden. Wenn Sie den genauen Namen einer Instanz nicht kennen, 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 eine Instanz geben kann. Bitte beachten Sie die notwendige Angabe des Privilegs *RemoteShutdown*. Mit `Win32shutdown(0,0)` wird der aktuelle Benutzer abgemeldet, mit `Shutdown()` der Rechner heruntergefahren.

Reboot

```
' -- Bindung an Menge der Betriebssysteme
Set OSet =
GetObject("winmgmts:{impersonationLevel=impersonate,(RemoteShutdown)}
//Sonne2000").ExecQuery("select * from Win32_OperatingSystem where
Primary=true")
' -- Neustart "aller" BS (ist aber immer nur eins!)
For Each o In OSet
    say "Betriebssysteme: " & o.Name
    'o.Win32shutdown 0, 0 ' Abmelden
    'o.Shutdown ' Herunterfahren
    o.Reboot ' Neustarten
    say "Aktion wurde ausgelöst!"
Next
```

Listing 14.24: Reboot eines Systems [WMI_Reboot.wsf]

Performance-Daten

Leistungsdaten WMI ermöglicht über den *Performance Counters Provider* Zugriff auf zahlreiche Leistungsdaten des Windows-Systems. Die Klassen beginnen mit der Zeichenfolge Win32_PerfRawData. Wenn Sie diese Klassen nicht finden, starten Sie den WMI-Dienst einmalig manuell an der Kommandozeile mit Winmgmt /resyncperf.

```
Set OSet = GetObject("winmgmts://
mars\root\cimv2").InstancesOf("Win32_PerfRawData_PerfProc_Process")
For Each p In OSet
    say p.Name & ": " & p.WorkingSet / 1024 & " KB"
Next
```

Listing 14.25: Speichernutzung der aktiven Prozesse [wmi_speichernutzung.wsf]

```
Set OSet =
GetObject("winmgmts:\\mars\root\cimv2").InstancesOf("Win32_PerfRawData
_ASP_ActiveServerPages")
For Each o In OSet
    say "Anfragen Gesamt: " & o.Requeststotal
    say "davon erfolgreich: " & o.RequestsSucceeded
    say "Aktuelle Sitzungen: " & o.SessionsCurrent
    say "Abgelaufene Sitzungen: " & o.SessionsTimedOut
Next
```

Listing 14.26: Performance-Daten über Active Server Pages [WMI_ASP.wsf]

Erzeugung einer Freigabe

Freigabe anlegen Mit ADSI kann eine Freigabe erzeugt werden, jedoch ohne die Berechtigungen auf einer Freigabe festzulegen. In WMI gibt es diese Beschränkung nicht.

```
' - Service-Objekt ermitteln
Set objServ = GetObject("winmgmts:")
' - Binden an ObjectSet
Set objOSet = objServ.InstancesOf("Win32_Share")
' - Ausgabe des ObjectSets
For Each o In objOSet
    say o.Name & ":" & o.Path
Next
```

Listing 14.27: Liste aller Freigaben auf einem System [wmi_shares.wsf]

Erstellung der NT-Sicherheitsbeschreibung

Aufwendigste Aufgabe bei der Erzeugung einer Freigabe ist die Zusammensetzung eines korrekt aufgebauten Security Descriptors (zu den Grundlagen der Windows-Sicherheit siehe Kapitel 13.11.5. Das Objektmodell zum Aufbau eines SDs ist in WMI ähnlich, aber im Detail anders als in der *ADsSecurity-Komponente*. Ein Unterschied liegt darin, dass es keine Klasse zur Kapselung einer Access Control List (ACL) gibt. Eine ACL ist in WMI nur ein Array von

Weitere WMI-Beispiele

Win32_ACE-Objekten. Ein anderer Unterschied ist, dass der Trustee beim Aufbau einer ACE nicht als String übergeben werden kann, sondern eine Instanz von Win32_Trustee erwartet wird. Beim Aufbau einer ACE und eines Win32_Trustee-Objekts unterstützen Sie die im folgenden Listing implementierten Funktionen MakeACE() und MakeTrustee(). Es gibt nur vier Rechte (*lesen* = 1179817, *ändern* = 1245462, *Vollzugriff* = 2032127, *kein Recht* = 1), wobei eine Kombination von Lesen und Ändern möglich wird, indem zwei getrennte ACEs für den Trustee angelegt werden. MakeTrustee() muss für die Well Known Security Principal eine Fallunterscheidung durchführen, wobei im nachstehenden Quelltext nur »Jeder« berücksichtigt wird. Im nachfolgenden Beispiel wird eine neue Freigabe für C:\TEMP angelegt. Für normale Benutzer verwendet MakeTrustee() das Active Directory Service Interface (ADSI), um den Security Identifier (SID) eines Benutzers zu ermitteln.

```
' -- AccessMasks für Freigaben
Const SHARE_READ = 1179817
Const SHARE_CHANGE = 1245462
Const SHARE_FULL = 2032127
Const SHARE_NONE = 1
' -- Bindung
Set objServ =
GetObject("WINMGMTS:{impersonationLevel=impersonate,(Security)}!\mars
\ROOT\CIMV2")
' ----- Erzeugung eines neuen SDs
' -- Neue Instanz von Win32_SecurityDescriptor
Set objSDClass = objServ.Get("Win32_SecurityDescriptor")
Set oSD = objSDClass.SpawnInstance_()
oSD.Properties_.Item("ControlFlags") = 4
' -- ACEs erzeugen
Set oACE1 = MakeACE(SHARE_NONE, 3, 0, MakeTrustee("mars", "Everyone"))
Set oACE2 = MakeACE(SHARE_READ, 3, 0, MakeTrustee("mars", "HS"))
Set oACE3 = MakeACE(SHARE_CHANGE, 3, 0, MakeTrustee("mars", "HS"))
' -- ACL zusammensetzen als Array von ACEs
dACL = Array(oACE1, oACE2, oACE3)
' -- ACL an SD zuweisen
oSD.Properties_.Item("DACL") = dACL
' ----- Erzeugung der Freigabe
' -- Neue Instanz von WIN32_Share
Set oFreigabe = objServ.Get("Win32_Share")
Set objInParam =
oFreigabe.Methods_("Create").InParameters.SpawnInstance_()
' -- Attribute setzen
objInParam.Properties_.Item("Name") = "temp-freigabe8"
objInParam.Properties_.Item("Path") = "C:\temp"
objInParam.Properties_.Item("Type") = 0
objInParam.Properties_.Item("Description") = "Per Skript erzeugt!"
objInParam.Properties_.Item("Access") = oSD
objInParam.Properties_.Item("MaximumAllowed") = 10 _
```

Windows Management Instrumentation (WMI)

```
'Standard ist 'max allowed'
' objInParam.Properties_.Item("Password") = "Password" 'optional
' -- Anlegen der Freigabe
Set objOutParams = oFreigabe.ExecMethod_("Create", objInParam)
' -- Ergebnisauswertung
Select Case objOutParams.returnValue
    Case 0: say "Freigabe wurde angelegt!"
    Case 22: say "Fehler: Freigabe existierte schon!"
    Case Else: say "Fehler: #" & objOutParams.returnValue
End Select

MakeTrustee() ' ### Erzeugung eines WIN32_Trustee-Objekts
Function MakeTrustee(strDomain, strName)
Dim objTrustee ' As WbemScripting.SWbemObject
Dim objUser 'As IADsUser
Dim Sid
If UCase(strName) = "EVERYONE" Then 'Sonderfall Jeder
    Sid = Array(1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0)
Else ' SID via ADSI ermitteln
    Set objUser = GetObject("WinNT://" & strDomain & "/" & strName)
    Sid = objUser.Get("ObjectSID")
End If
Set objTrustee =
GetObject("Winmgmts:{impersonationlevel=impersonate}!root/
cimv2:Win32_Trustee").SpawnInstance_
objTrustee.domain = strDomain
objTrustee.Name = strName
objTrustee.Properties_.Item("SID") = Sid
Set MakeTrustee = objTrustee
End Function

MakeACE() ' ### Erzeugung eines WIN32_ACE-Objekts
Function MakeACE(accessmask, AceFlags, AceType, objTrustee)
Dim objACE ' As WbemScripting.SWbemObject
Set objACE =
GetObject("Winmgmts:{impersonationlevel=impersonate}!root/
cimv2:Win32_Ace").SpawnInstance_
objACE.Properties_.Item("AccessMask") = accessmask
objACE.Properties_.Item("AceFlags") = AceFlags
objACE.Properties_.Item("AceType") = AceType
objACE.Properties_.Item("Trustee") = objTrustee
Set MakeACE = objACE
End Function
```

*Listing 14.28: Erzeugung einer neuen Freigabe inkl. Sicherheitseinstellungen
[WMI_CreateShare.wsf]*

14.7 WMI-Werkzeuge

Microsoft liefert für das *Windows Management Instrumentation (WMI)* einige Tools, die mit dem WMI-SDK installiert werden. Dieses ist auf der CD enthalten [CD:/sdk/wmi]. *WBEMTest* und *MofCom* sind allerdings Bestandteile des WMI-Kerns.



14.7.1 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* (vgl. Anhang A) 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

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.

WMI Object Browser versus Object Browser

Der WMI Object Browser besteht aus HTML-Seiten mit DHTML-Skripten 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.

Implementierung

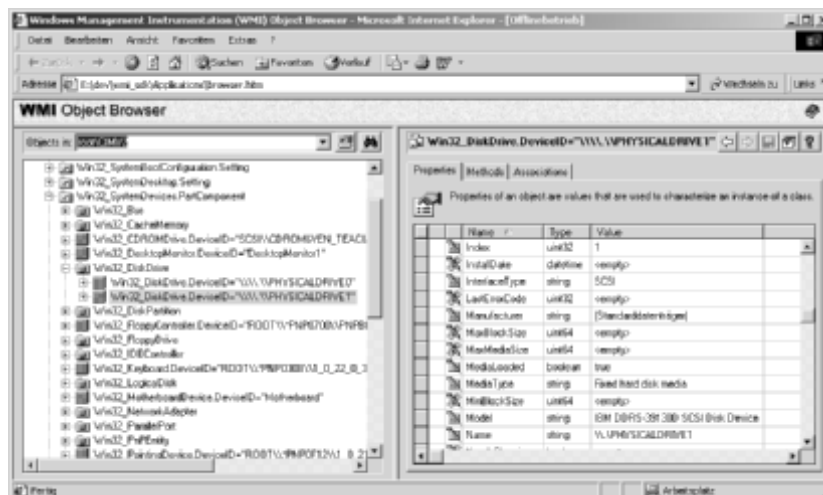


Abbildung 14.16: Der WMI Object Browser zeigt an, dass der Computer zwei Festplatten besitzt, wobei die ausgewählte zweite Festplatte eine IBM DDRS-Festplatte mit SCSI-Schnittstelle ist.

Tabelle 14.17:
Symbole im Object
Viewer

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.
	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.

Funktionsüberblick

Der Object Viewer ermöglicht folgende Funktionen:

- Hilfe** ► 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.
- Attribute** ► Anzeige und Veränderung der Attribute. Veränderungen können direkt in der Attributtabelle vorgenommen werden. Änderungen müssen explizit persistent gemacht werden (Diskettensymbol).
- Methoden** ► 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.
- Objektbeziehungen** ► Darstellung der Beziehungen der Instanz zu anderen Instanzen in Form eines Graphen (Registerkarte ASSOCIATIONS, vgl. Abbildung 14.18).

Abbildung 14.17:
Die Registerkarte
»Associations«
ASSOCIATIONS
im Object Viewer
des WMI Object
Browsers zeigt die
WMI-Instanz einer
Festplatte, die vier
Partitionen enthält.



14.7.2 WMI CIM Studio

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 Browser: 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.

Das CIM Studio umfasst auch einen Großteil der Funktionalität des WMI Object Browser, 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).

Vererbungshierarchie in studio.htm

Vergleich zum WMI Object Browser

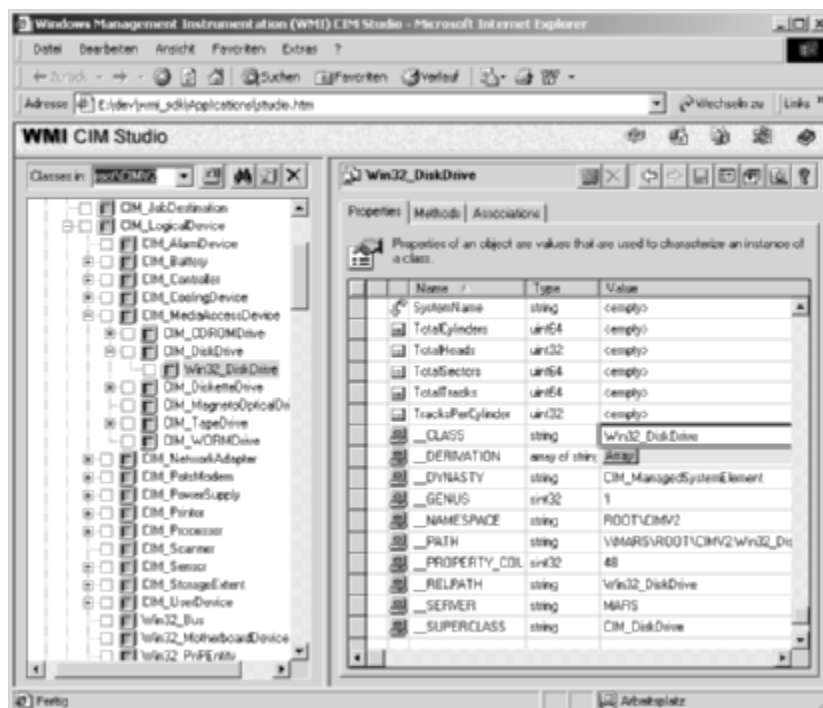
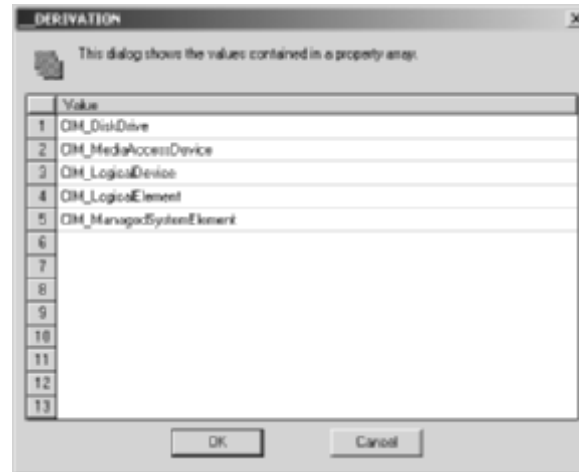


Abbildung 14.18:
Die Klasse Win32_DiskDrive im CIM Studio. Die Klasse ist eine Unterklasse von CIM_DiskDrive, die wiederum Unterklasse von CIM_MediaAccess Drive ist.

Abbildung 14.19:
Vererbungshierarchie der Klasse
Win32_DiskDrive



Funktionsüberblick

- Attribute** Die Registerkarte PROPERTIES (vgl. Abbildung 14.19) 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.
- Methoden** 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.
- Assoziationen** Unter ASSOCIATIONS erscheint ein Graph, der anzeigt, welche Unterklassen die Klasse enthalten kann.

Tabelle 14.18:
Die wichtigsten
Symbole im Class
Viewer

Symbol	Erläuterung
	Erzeugen neuer Instanzen dieser Klasse
	Zeigt die Instanzen dieser Klasse (Tabellensymbol)
	Erstellen, Testen und Speichern von WQL-Abfragen
	Zeigt die Hilfeinformationen zu dieser Klasse an

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

WMI-Werkzeuge

erscheint der Object Viewer für diese Instanz mit den schon beim WMI Object Browser beschriebenen Fähigkeiten.

CSName	Description	ExecutablePath	ExecutionS	Handle	KernelModeTl	MaximumWorkingSet
MARS	hnt.exe	F:\WINNT\hnt.exe	completo	236	256672240	1413120
MARS	IE-PLORIE.EXE	F:\Programme\Internet Explorer\	completo	213	422607968	1413120
MARS	ITVisions.exe	D:\bach\vis.exe\ITVisions.exe	completo	25	400576	1413120
MARS	lsass.exe	F:\WINNT\System32\lsass.exe	completo	266	22932876	1413120
MARS	mdm.exe	F:\WINNT\System32\mdm.exe	completo	85	44060360	1413120
MARS	MSACCESS.EXE	E:\MSOffice\Office\MSACCESS	completo	264	648732832	1413120
MARS	mofc.exe	F:\WINNT\System32\mofc.exe	completo	158	1001440	1413120
MARS	mofsec.exe	F:\WINNT\System32\mofsec.exe	completo	67	580728	1413120
MARS	mimm.exe	F:\Programme\Outlook Express\	completo	308	145609232	1413120
MARS	MSTask.exe	F:\WINNT\System32\MSTask.e	completo	89	488576	1413120

Abbildung 14.20:
Listen der Instanzen der Klasse
WIN32_Process
in der Instanzen-
ansicht des
CIM Studios

Weitere Funktionen

Weiterhin bietet der Class Viewer folgende Möglichkeiten:

- ▶ Ausführen von WQL-Abfragen (vgl. Abbildung 14.22). 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)

Weitere Features des Class Viewers

Abbildung 14.21:
WQL-Abfrage nach
den aktiven Win-
word-Prozessen im
Abfragefenster des
CIM Studios



14.7.3 WMI Event Registration Tool

eventreg.htm 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 (siehe Kapitel 14.3.7) beherrschen.

Beispiel 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.

Konsument definieren

- Definieren Sie zunächst einen Consumer:
 - 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 MACHINEName 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.

WMI-Werkzeuge

- ▶ Klicken Sie auf die Klasse `__EVENTFILTER`, und wählen Sie im Kontextmenü **NEW INSTANCE**.
- ▶ Geben Sie in dem Dialogfenster einen beliebigen Namen und `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 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.

Bindung des Konsumenten an einen Filter

Damit ist die Konfiguration abgeschlossen. Alle auftretenden Ereignisse werden dann im WMI Event Viewer angezeigt.

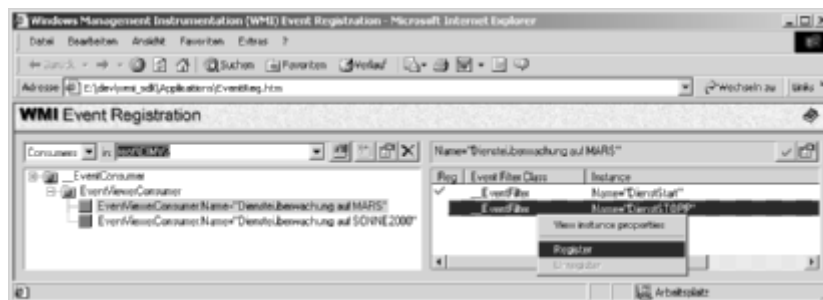


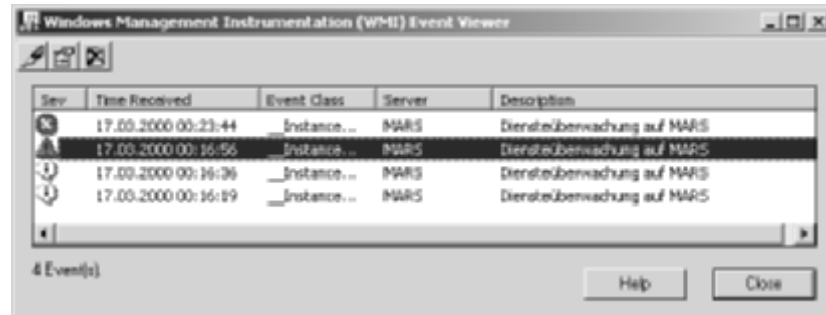
Abbildung 14.22:
Bindung einer
Instanz von
`__EventFilter` an
eine Instanz von
`EventFilterConsumer`

14.7.4 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 Event Consumer, 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 14.23:
Anzeige eines Ereignisses im Event Viewer: Durch einen Doppelklick erhalten Sie Details zu dem Ereignis. Dabei sehen Sie auch alle Attributwerte des auslösenden MOs.

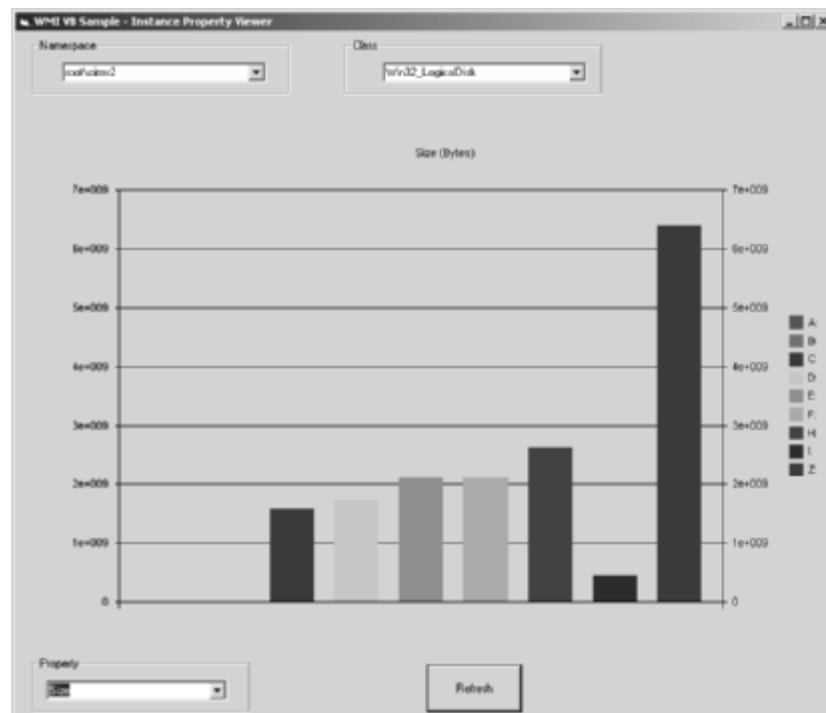


14.7.5 VBInstance

VBInstance.vbp

Microsoft liefert im WMI-SDK eine Visual Basic-Beispielanwendung, die sich als Tool eignet. Das Beispiel *VBInstance* (Verzeichnis /SCRIPTING/VB/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 14.24:
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.



14.7.6 WMI-Testprogramm

WBEMTEST.EXE gehört zum Standardinstallationsumfang (Verzeichnis %System%/WBEM) und bietet ein einfaches, wenig komfortables 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.

WBEMTest.exe



Abbildung 14.25:
WBEMTest.exe
EST.EXE

14.7.7 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 14.1). 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

14.8 Ihre Expedition in die WMI

Der Ausflug in die weite Welt von WMI ist hier leider beendet. Ihnen mitgegeben seien jedoch einige Tipps zur Selbsterforschung von WMI:

Hilfe zur Selbst-
hilfe

- Stöbern Sie im WMI Object Browser oder in CIM Studio nach der Klasse, die auf Ihr Problem passt.

Windows Management Instrumentation (WMI)

- ▶ Lesen Sie sich die Hilfe zu der Klasse durch, um zu überprüfen, ob die Klasse die von Ihnen gesuchten Fähigkeiten hat.
- ▶ Schauen Sie sich die bestehenden Instanzen dieser Klasse an. Überprüfen Sie, ob dort wirklich die Instanzen enthalten sind, die Sie erwarten.
- ▶ Betrachten Sie die Attribute der Instanzen, um mit den typischen Werten vertraut zu werden. Verwenden Sie das Hilfe-Symbol, um sich Details über die Attribute anzeigen zu lassen.
- ▶ Schauen Sie sich die Methoden und Assoziationen an.
- ▶ Probieren Sie es aus!

Danach werden Sie in der Lage sein, Ihre Problemstellung mit Hilfe von WMI zu lösen.

15 COM+ Administration Objects

Die Verwaltung des COM+-Katalogs kann durch die Komponente *COM+ Administration Objects* automatisiert werden. Dies war auch schon unter MTS 2.0 möglich mit den *MTS Administration Objects* (MTSADMIN.DLL). Beide Komponenten sind ein Meta-Objektmodell für das Objektmodell des COM+- bzw. MTS-Katalogs. Allgemeine Informationen zum MTS und zu COM+ finden Sie in Kapitel 2.

Mit der Einführung von COM+ wurden sowohl das Basis-Objektmodell als auch das Meta-Objektmodell geändert. Der COM+-Katalog wird durch eine spezielle Komponente, CATSRV.DLL realisiert. Diese Komponente ist in Form der Klasse *CatSrv.CatalogServer* im Rahmen der *System Application* selbst im COM+ konfiguriert. Die COMADMIN.DLL realisiert dazu ein Meta-Objektmodell.

COM+-Katalog automatisieren

Name und Abkürzung	COM+ Administration Objects
Name der Komponentendatei	comadmin.dll
Interner Name der Typbibliothek	COMAdmin
Helpstring der Typbibliothek	COM + 1.0 Admin Type Library
Hersteller	Microsoft
Lizenzierung	Bestandteil von Windows ab Windows 2000
Besprochene Version	2001.12.4302.12
NT4, Windows 95/98/ME	nicht verfügbar (nur MTS 2.0 Administration Objects)
Windows2000	Version 03.00.00.3422 enthalten
Whistler (Windows XP / Windows.NET)	Version 2001.12.4302.12 enthalten
Dokumentation	MSDN Library [MSL00]

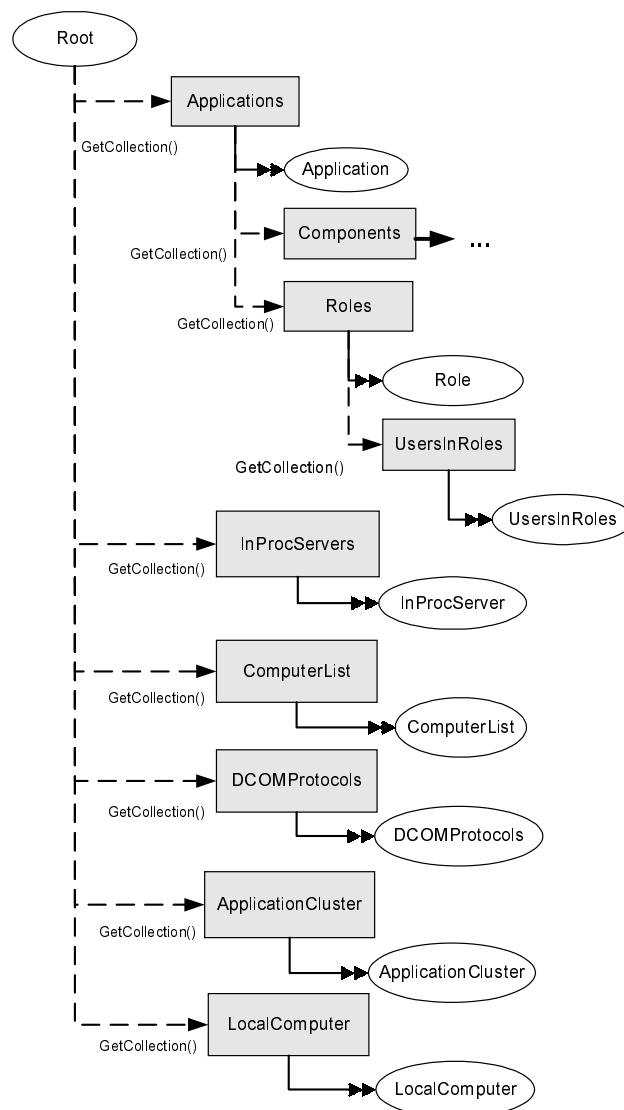
*Tabelle 15.1:
Schnellinfo COM+
Administration
Objects*

15.1 Basis-Objektmodell

Objektmodell des COM+-Katalogs

Das Basis-Objektmodell besitzt unterhalb des Root-Objekts verschiedene Collections. Die Collection Applications führt zu den konfigurierten COM+-Applications mit allen dazugehörigen Klassen und Schnittstellen. Weiterhin enthält das Root-Objekt Collections für verschiedene Informationen, z.B. die in der Management Console in der Komponentenverwaltung eingerichteten Computer (ComputerList) sowie alle in der Registry verzeichneten COM-In-process-Komponenten (InProcServers) (siehe nächste Abbildungen).

Abbildung 15.1:
Teil 1 des COM+-
Katalog-Objekt-
modells



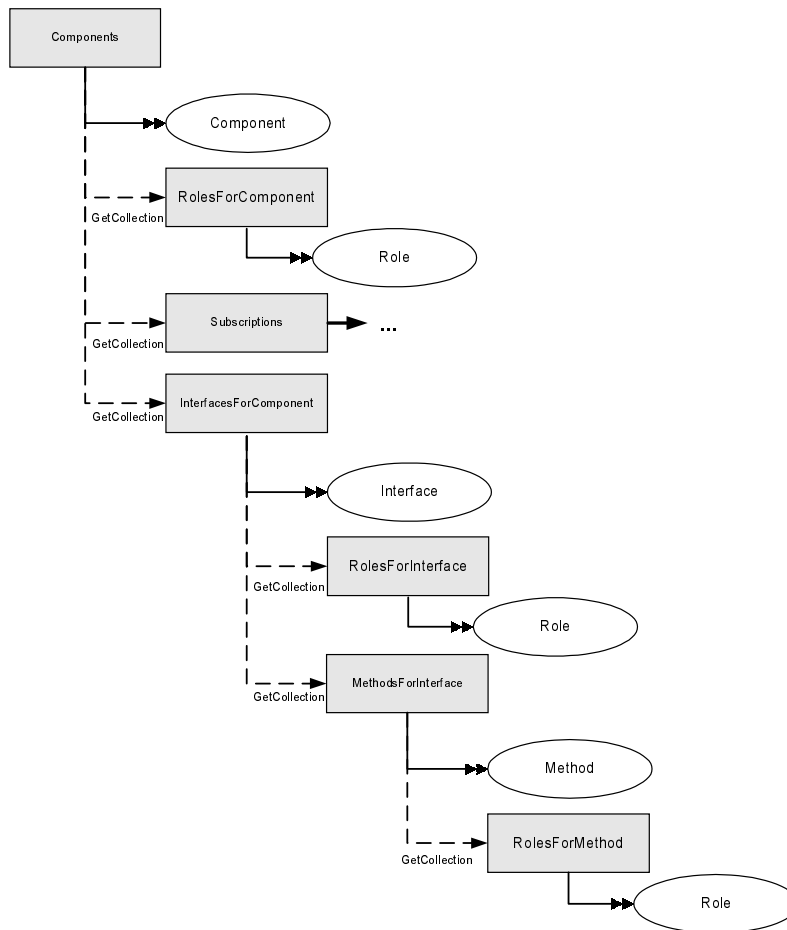


Abbildung 15.2:
Teil 2 des COM+-
Katalog-Objekt-
modells

Wenn Sie das Objektmodell oder das Komponentendienste-MMC-Snap-In genauer betrachten, werden Sie schnell einen großen MINFU feststellen: Microsoft definiert die Hierarchie aus den Begriffen COM-Anwendungen, Komponenten, Interfaces und Methoden. Wo sind die COM-Klassen? Nun, die Klassen sind da, wo Microsoft von Komponenten redet. In eine COM+-Anwendung werden einzelne Klassen eingetragen, nicht Komponenten (dies hat allerdings Auswirkungen auf die Komponente, zu der die Klasse gehört).



Jedes Element in einem COM+-Katalog besitzt eine eindeutige GUID. Der textliche Name eines Elements ist jedoch nicht zwingend eindeutig. So kann ein Anwendungsname mehrfach für verschiedene Anwendungen vergeben werden.

15.2 Meta-Objektmodell

Das Meta-Objektmodell definiert nur drei Klassen (sowie zahlreiche Konstantenlisten):

- COMAdmin-Catalog** ► COMAdminCatalog. Diese Klasse repräsentiert das Root-Objekt des COM+-Katalogs und bietet übergreifende administrative Funktionen.
- COMAdmin-CatalogObject** ► COMAdminCatalogObject ist eine Meta-Klasse für ein beliebiges Objekt aus dem COM+-Katalog.
- COMAdmin-CatalogCollection** ► COMAdminCatalogCollection ist eine Meta-Klasse für eine beliebige Collection aus dem COM+-Katalog.

Abbildung 15.3:
Typbibliothek der
COM+ Administration Objects im
VB6-Objektkatalog



Ungewöhnliche Navigation

- GetCollection()** Die Navigation in dem Meta-Objektmodell ist höchst außergewöhnlich und gewöhnungsbedürftig: Der Zugriff funktioniert nicht wie sonst üblich konsequent stufenweise über Objekte und Collections. Vielmehr werden den einzelnen Objekten untergeordnete Collections über dem Objekt übergeordnete Collections angesprochen.

Meta-Objektmodell

Beispiel: Nach einer Bindung eines `COMAdminCatalogObject` an ein `Application-Object` aus dem COM+-Katalog, ist ein direkter Zugriff von dem einzelnen `Application-Object` auf die darin installierten Klassen nicht möglich. Umständlich muss der Programmierer dazu aus der `Applications-Collection` heraus die Methode `GetCollection()` aufrufen unter Angabe der gewünschten `Collection-Klasse` (hier: `Components`) sowie des Schlüsselwerts desjenigen `Applications-Objekts`, dessen `Components-Collection` gesucht wird.

Und noch eine wichtige Ausnahme: Jede `Collection` muss zunächst mit der Methode `Populate()` gefüllt werden. Andere administrative Objektmodelle erledigen das automatisch.

Beispiel

Populate()

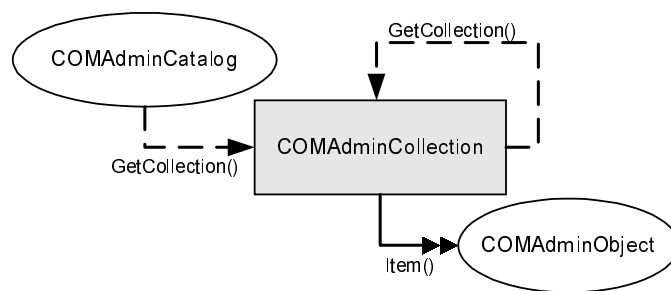


Abbildung 15.4:
COM+-Meta-
Objektmodell

Attribut	Erläuterung
Name	Name des Basisobjekts. Es wird der Name geliefert, der im Komponentendienste-Snap-In angezeigt wird. Es gibt keine explizite Möglichkeit, den Klassennamen zu ermitteln. Dieser ergibt sich aus dem Namen der Collection, in der das Objekt enthalten ist.
Key	Wert des Schlüsselattributs
Valid	Das Attribut liefert <i>True/False</i> und gibt Auskunft darüber, ob alle Attribute des Basisobjekts erfolgreich in das Meta-Objekt eingelesen werden konnten.

Tabelle 15.2:
Attribute der Meta-
klasse *COMAdmin
CatalogObject*

Methode	Erläuterung
Value(strAttribut)	Ermöglicht Lese- und Schreibzugriff auf ein Attribut des Basisobjekts
IsPropertyReadOnly (strAttribut)	Ermittelt, ob ein Attribut des Basisobjekts verändert werden kann
IsPropertyWrite Only (strAttribut)	Ermittelt, ob ein Attribut des Basisobjekts verändert werden kann

Tabelle 15.3:
Methoden der Meta-
klasse *COMAdmin
CatalogObject*

Tabelle 15.4:
Attribute der
Metaklasse
COMAdminCatalog
Collection

Attribut	Erläuterung
Count	Anzahl der Objekte in der Collection
Name	Name der Collection
AddEnabled	Liefert die Information (<i>True/False</i>), ob neue Objekte hinzugefügt werden können
RemoveEnabled	Liefert die Information (<i>True/False</i>), ob vorhandene Objekte entfernt werden können
DataStoreMajorVersion DataStoreMinorVersion	Diese Attribute liefern zusammen die Versionsnummer des Katalogs.

Tabelle 15.5:
Methoden der
Metaklasse COM-
AdminCatalog
Collection

Methode	Erläuterung
Item(index)	Liefert den Zeiger auf ein COMAdminCatalogObject aus der Collection anhand des übergebenen Index
Remove(index)	Entfernt ein Objekt auf Basis seines numerischen Index
Add()	Legt eine neue Instanz einer Basisklasse an und fügt diese der Collection hinzu. Da alle COM+ Catalog Collections homogene Collections sind, ist der Klassenname durch die Collection, auf der Add() ausgeführt wird, festgelegt. Add() liefert einen Zeiger auf die neue Instanz zurück.
Populate()	Diese Methode liest die der Basis-Collection untergeordneten Basis-Objekte in den Cache ein. Ohne Ausführung dieser Methode ist eine COMAdminCatalog Collection stets leer!
SaveChanges (Anzahl)	Speichert Änderungen im COM+-Katalog ab. Die Methode liefert über ihren Rückgabewert eine eventuelle Fehlernummer. Anzahl ist ein OUT-Parameter; d.h., dass Sie für diesen Parameter eine leere Variable übergeben müssen, die per Call by Reference von der Methode gefüllt wird. Anzahl liefert dem Aufrufer die Anzahl der vorgenommenen Änderungen.
GetCollection (CollectionName, Key)	Bindet eine COMAdminCatalogCollection an eine Basis-Collection und liefert einen Zeiger auf die betreffende Instanz von COMAdminCatalogCollection. Als Parameter muss nicht nur der Schlüsselwert der gewünschten Collection, sondern auch der Klassenname der Basis-Collection übergeben werden.

Bindung an Basis-Klassen

Bindung Ausgangspunkt der Arbeit mit dem COM+-Katalog ist stets die Instanziierung der Stammklasse.

Meta-Objektmodell

```
Dim objCAT ' As COMAdmin.COMAdminCatalog
Set objCAT = CreateObject("COMAdmin.COMAdminCatalog.1")
```

Da ein `ComAdminCatalogObject` keinerlei Verfahren für die Bindung an ein Basisobjekt bereitstellt, ist es nicht möglich, von der Stammklasse aus direkt ein Basisobjekt anzusprechen. Sie müssen ein Objekt also immer über seine Collection ansprechen.

Auch für Collections gibt es eine Einschränkung: Es ist immer nur die nächsttiefere Collection direkt erreichbar. Nicht möglich ist es, eine bestimmte Collection in der Objekthierarchie des Katalogs direkt anzusprechen. Sie müssen sich stets über die Objekthierarchie des Katalogs zu dem gewünschten Objekt vorhangeln. Vom Stammobjekt aus können Sie nur die Applications-Collection erreichen.

GetCollection()

```
Set colApps = objCAT.GetCollection("Applications")
```

Der Zugriff auf eine andere Collection führt zu einem Fehler. Im Gegensatz zu anderen Objektmodellen und Meta-Objektmodellen ist beim COM+-Katalog ein expliziter Befehl zum Füllen der Collection notwendig: `colApps.Populate()`. Danach können Sie mit `For...Each` über die Collection iterieren.

Populate()

```
say "Anzahl der COM+-Anwendungen:" & colApps.Count
For a = 0 To colApps.Count - 1
    Set objApp = colApps.Item(a)
    say a + 1 & ":" & objApp.Name & ":" & objApp.key
Next
```

Innerhalb der Collection ist ein konkretes Objekt mit `Item()` leider nur über den Index, nicht über den Namen oder die Schlüssel-GUID ansprechbar. Es gäbe theoretisch mit der Methode `PopulateByKey(array_of_guid)` eine Möglichkeit, sich die mühsame Suche über alle Elemente zu ersparen, doch diese Methode kann weder Visual Basic noch Visual Basic Script aufrufen.

Item()

Ausgabe der Objekthierarchie

Das folgende Skript steigt den Weg im COM+-Katalog eines entfernten Systems mühsam bis zu den Schnittstellen hinab.

```
Dim objCAT ' As COMAdmin.COMAdminCatalog
Dim colApps ' As COMAdmin.COMAdminCatalogCollection
Dim colComs ' As COMAdmin.COMAdminCatalogCollection
Dim colInt ' As COMAdmin.COMAdminCatalogCollection
Dim objApp ' As COMAdmin.COMAdminCatalogObject
Dim objCOM ' As COMAdmin.COMAdminCatalogObject
Dim objINT ' As COMAdmin.COMAdminCatalogObject
Dim a, c, i ' Zähler
Set objCAT = CreateObject("COMAdmin.COMAdminCatalog.1")
objCAT.Connect("sonne2000")
' Zugriff auf Applications-Collection
Set colApps = objCAT.GetCollection("Applications")
colApps.Populate
```

COM+ Administration Objects

```
' Iteration über Applications
say "Anzahl der COM+-Anwendungen:" & colApps.Count
For a = 0 To colApps.Count - 1
    Set objApp = colApps.Item(a)
    say a + 1 & ":" & objApp.name & ":" & objApp.key
    ' Zugriff auf Klassen-Collection
    Set colComs = colApps.GetCollection("Components", objApp.key)
    colComs.Populate
    ' Iteration über Klassen
    say "enthaltene Komponenten: " & colComs.Count
    For c = 0 To colComs.Count - 1
        Set objCOM = colComs.Item(c)
        say "      " & c + 1 & ":" & objCOM.name
        ' Zugriff auf Schnittstellen
        Set colInt = colComs.GetCollection("InterfacesForComponent", _
            objCOM.key)
        colInt.Populate
        ' Iteration über Schnittstellen
        For i = 0 To colInt.Count - 1
            Set objINT = colInt.Item(i)
            say "          - " & i + 1 & ":" & objINT.name
        Next
    Next
Next
say "---"
End Sub
```

Listing 15.1: Ausgabe aller Anwendungen mit ihren Klassen und Schnittstellen
[complus_liste.wsf]

15.3 Arbeit mit der COMAdminCatalog-Klasse

COMAdmin-Catalog

Das COMAdminCatalog-Objekt stellt im Wesentlichen Methoden zur Sicherung des Katalogs sowie zur Konfiguration von Anwendungen bereit.

Tabelle 15.6:
Ausgewählte
Methoden der Klasse
COMAdminCatalog

Methode	Erläuterung
GetCollection (strCollectionName)	Liefert einen Zeiger auf die angegebene Collection. Vom Root-Objekt aus können Sie nur die Applications-Collection erreichen.
Connect (strComputerName)	Stellt eine Verbindung zu einem entfernten COM+-Katalog her
ImportComponent (strApp Key, strProgID strCLSID)	Import einer bereits registrierten Klasse in eine bestimmte Anwendung

Arbeit mit der COMAdminCatalog-Klasse

Methode	Erläuterung
InstallComponent (strAppKey, strDLL, strTLB, bstrProxy)	Registrierung einer Komponente und Import aller enthaltenen Klassen in eine bestimmte Anwendung
ShutdownApplication (strAppKey)	Herunterfahren des Prozesses einer Anwendung
InstallApplication (strApplicationFile)	Installation einer Anwendung aus einer Anwendungsdatei
BackupREGDB (strBackup File)	Backup der RegDB in die angegebene Datei
RestoreREGDB (strBackup File)	Restore der RegDB aus der angegebenen Datei
StartApplication (strAppKey)	Start des Anwendungsprozesses

Tabelle 15.6:
Ausgewählte
Methoden der
Klasse COM-
AdminCatalog
(Forts.)

Bei den Methoden, die sich auf eine Anwendung beziehen, kann statt der GUID der Anwendung auch stets der Name der Anwendung angegeben werden, wobei dann das Risiko besteht, dass der Name nicht eindeutig ist.



Hinzufügen von Anwendungen

Eine COM+-Anwendung wird entweder durch Import einer bestehenden COM+-Anwendungsinstallationsdatei oder durch Neuanlegen erzeugt. Eine COM+-Anwendungsinstallationsdatei ist eine Windows Installer Datei (.MSI). Von MTS 2.0 erzeugte Dateien (.PAK) werden auch akzeptiert.

**Install
Application()**

```
Dim oCat ' As COMAdmin.COMAdminCatalog
Set oCat = CreateObject("COMAdmin.COMAdminCatalog")
oCat.InstallApplication "d:\buch\complus\scriptingapp.msi"
```

Listing 15.2: Import einer Anwendungsinstallationsdatei [complus_importapp.wsf]

Mit der Ausführung dieses Dreizeilers wird eine Anwendung »Scripting-Buch« erzeugt, die die Klassen aus den Komponenten MATHE.DLL und WINDOWSSCRIPTING.DLL enthält. Auch das Anlegen einer neuen Anwendung ist nur ein wenig mehr Arbeit.

```
Dim oCat ' As COMAdmin.COMAdminCatalog
Dim colApps ' As COMAdmin.COMAdminCatalogCollection
Dim oApp ' As COMAdmin.COMAdminCatalogObject
Set oCat = CreateObject("COMAdmin.COMAdminCatalog")
say TypeName(oCat)
Set colApps = oCat.GetCollection("Applications")
' -- Anlegen und Benennen einer neuen Anwendung
Set oApp = colApps.Add
oApp.Value("Name") = "Scripting-Buch"
```

```
colApps.SaveChanges
ID = oApp.key
say "Anwendung angelegt. ID=" & ID
say "Installation beendet!"
```

*Listing 15.3: Dieses Skript erzeugt eine leere Anwendung »Scripting-Buch«.
[complus_newapp.wsf]*



Es ist möglich, zwei Anwendungen gleichen Namens anzulegen. Sie sollten dies aber vermeiden.

Hinzufügen von Klassen

COM-Klassen können einer COM+-Anwendung auf zwei Arten hinzugefügt werden:

Import Component()

- Es können bereits registrierte COM-Klassen importiert werden. In diesem Fall wird nur die angegebene Klasse der Anwendung hinzugefügt.

```
Catalog.ImportComponent "AppKey", "ProgID"
```

Wenn Sie eine Registrierung auf Basis einer *Version Independent ProgID* vornehmen, wird dadurch die Version der Komponente installiert, die mit dieser ProgID verbunden ist. Im Komponentendienste-Snap-In wird die vollständige ProgID dieser Version angezeigt (Beispiel: Wenn Sie `ADODB.Recordset` importieren, werden Sie in der MMC `ADODB.Recordset.2.5` lesen).

Install Component()

- Es können neue COM-DLLs installiert werden, wobei dann alle enthaltenen öffentlich instanzitierbaren Klassen Teil der COM+-Anwendung werden. Es ist nachträglich möglich, einzelne Klassen wieder zu entfernen.

```
Catalog.InstallComponent "AppKey", "\\pfad\name.dll", _
"\\pfad\typelib.tlb", "\\pfad\proxystub.dll"
```

Der dritte Parameter dient der Angabe einer Typbibliotheksdatei, wenn die Typbibliothek nicht in der Komponentendatei enthalten ist. Der vierte Parameter kann den Pfad zu einer speziellen Proxy-Stub-DLL enthalten.



Microsoft empfiehlt, Klassen zu einer Anwendung durch Installieren, nicht durch Importieren hinzuzufügen, da importierte Klassen einigen Einschränkungen unterliegen. So kann COM+ für diese Klassen die Schnittstellen und Methoden nicht anzeigen.

Die nachfolgenden beiden Zeilen ergänzen das letzte Listing um die Installation bei beiden Komponenten. Ein `InstallComponent()` wird fehlschlagen, wenn die Komponenten bereits registriert sind. Sie müssen also ggf. die erzeugte COM+-Anwendung erst wieder löschen.

Arbeit mit der COMAdminCatalog-Klasse

```
' -- Installieren von Komponenten
Const pfad = "d:\buch\complus"
oCat.InstallComponent ID, pfad & "\windowsscripting.DLL", "", ""
oCat.InstallComponent ID, pfad & "\mathe.DLL", "", ""
'ocat.ImportComponent "test", "Scripting.FileSystemObject"
```

*Listing 15.4: Installation einer Komponente in einer COM+-Anwendung
[complus_newapp.wsf]*

Löschen eines Eintrags

Ein Objekt in einer Collection wird durch `Remove()`, ausgeführt auf der Collection, gelöscht. Allerdings erwartet `Remove()` nicht den Key, sondern den laufenden Index innerhalb der Collection. Da ein Objekt seinen Index auch nicht zurückliefern kann, müssen Sie die Collection sequenziell durchlaufen. Es bietet sich an, eine Hilfsroutine zu schreiben:

Remove()

```
Sub complus_DelByKeyOrName(col, name, key)
Dim obj, anz, a
' -- Einlesen der Collection
col.Populate
' -- Anzahl ermitteln
anz = col.Count - 1
a = 0
' -- Schleife
Do While a < anz
' -- Zugriff auf Objekt
Set obj = col.Item(a)
' -- Kontrollausgabe
say a + 1 & " von " & anz & ":" & obj.name & ":" & obj.key
If obj.key = key Or obj.name = name Then
' -- Löschen
col.Remove (a)
col.SaveChanges
' -- Collection aktualisieren
'col.Populate
' -- Schleifenende verkürzen
anz = anz - 1
' -- Kontrollausgabe
say a + 1 & " gelöscht!"
Else
a = a + 1
End If
Loop
End Sub
```

Listing 15.5: Löschen eines Objekts in einer Collection im COM+-Katalog

**Anwendungs-
beispiel**

Diese Routine löscht entweder anhand eines Namens oder eines Schlüssels. Bei der Verwendung des Namens werden ganz bewusst alle Anwendungen dieses Namens gelöscht. Die Routine ist insofern etwas knifflig, als beim Löschen eines Objekts in der Collection alle anderen Objekte sofort nachrücken. Die Schleifenlänge muss also nach einem Löschvorgang verkürzt werden (damit die Schleife nicht über die veränderte Obergrenze hinausläuft), und der Schleifenzähler muss verharren (da sonst das Folgeobjekt übersehen wird).

```
Set objCAT = CreateObject("COMAdmin.COMAdminCatalog.1")
Set colApps = objCAT.GetCollection("Applications")
complus_DelByKeyOrName colApps, "Scripting-Buch", 0
```

*Listing 15.6: Löschen aller Anwendungen, die »Scripting-Buch« heißen
[complus_alleloeschen.wsf]*

15.4 MTS Administration Objects

**Beispiel zu
MTS 2.0**

Es folgt ein kleines Beispiel zum alten MTS. Mit dem folgenden Code, der auch noch unter COM+ funktioniert, werden alle Packages bzw. COM+ Applications heruntergefahren. Dies ist eine nützliche Funktion, da der MTS bzw. COM+ im Rahmen der *Just-in-Time-Activation* verwendete Komponenten erst nach einigen Minuten Leerlauf freigibt und in dieser Zeit sonst kein Austausch der Komponenten-Binärdaten möglich ist.



Die MTS Administration Objects sind auch unter Windows2000 aus Kompatibilitätsgründen weiterhin verwendbar. Für Windows2000 geschriebene Anwendungen sollten aber das erweiterte COM+-Modell benutzen. Informationen zu den *MTS Administration Objects* finden Sie in der MSDN Library [MSL00].

```
Dim objCAT ' As mtsadmin.Catalog
Dim colPack ' As mtsadmin.CatalogCollection
Dim objPack ' As Object
Dim objUtil ' As mtsadmin.PackageUtil
Dim n ' Packages-Zähler
' -- Zugriff auf MTSADMIN
Set objCAT = CreateObject("MTSAdmin.Catalog.1")
' -- Zugriff auf Packages
Set colPack = objCAT.GetCollection("Packages")
Set objUtil = colPack.GetUtilInterface
colPack.Populate
' -- Iteration über alle Packages
n = colPack.Count
For i = 1 To n - 1
    Set objPack = colPack.Item(i)
```

MTS Administration Objects

```
objUtil.ShutdownPackage objPack.Key  
say objPack.Value("Name") & " " & objPack.Key & " wurde  
heruntergefahren!"  
Next
```

Listing 15.7: Herunterfahren aller MTS-Packages eines MTS [mts_shutdown.wsf]

Unter dem MTS besaßen einige Klassen noch Mehrfachschnittstellen, auf die über das Attribut `GetUtilInterface` zugegriffen werden konnte.

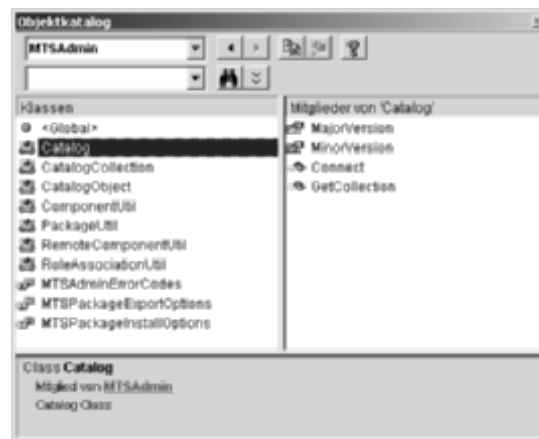


Abbildung 15.5:
Typbibliothek der
MTS Administra-
tion Objects im
VB6-Objektkatalog

16 SQL Server Distributed Management Objects (DMO) 7.0/2000

Die Komponente *Distributed Management Objects* (DMO) ist eine sehr mächtige COM-Komponente für die automatisierte Administration des Microsoft SQL Servers. Sie bildet die Funktionen des SQL Enterprise Managers nach und stand auch schon in früheren Versionen des SQL Servers zur Verfügung. DMO ist in weiten Teilen eine objektorientierte Sicht auf den Inhalt der SQL Server-Systemdatenbanken *Master* und *MSDB*.

DMO Version 7.0

DMO wurde für den SQL Server Version 7.0 stark erweitert. DMO Version 7.0 ist mit 60 Einzelklassen, 61 Collection-Klassen, 155 Interfaces und stattlichen 97 Konstantenlisten eine der umfangreichsten Automatisierungskomponenten unter den derzeit verfügbaren COM-Komponenten überhaupt. Auf Grund dieser großen Funktionsfülle kann im vorliegenden Buch nur ein kleiner Teil der Möglichkeiten besprochen werden, die DMO bietet.

SQL Server-Automatisierung

DMO für SQL Server 7.0

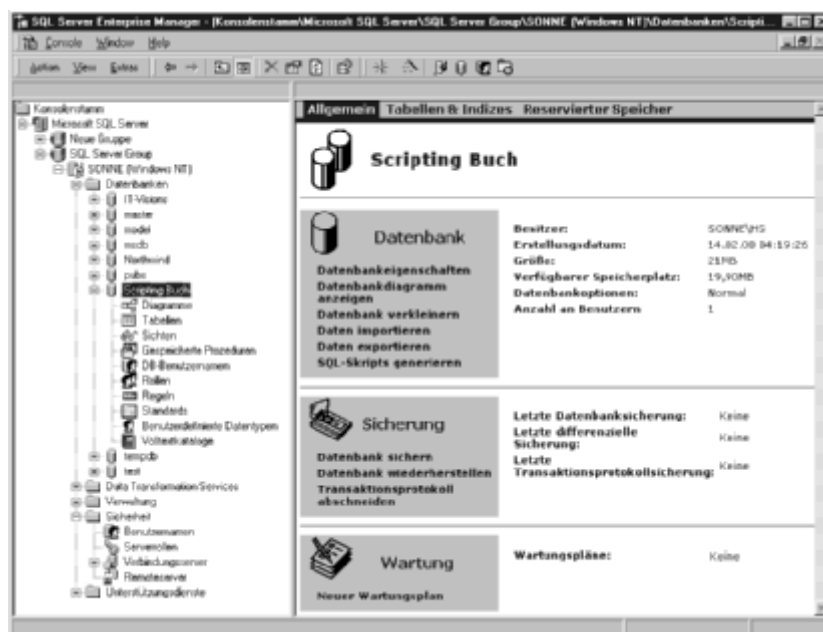


Abbildung 16.1:
Der SQL Server
Enterprise Manager

DMO Version 2000**DMO für SQL Server 2000**

Im SQL Server 2000 hat DMO einige kleinere Erweiterungen erfahren. In diesem Fall hat Microsoft die Erweiterungen konsequent in neuen Klassen abgebildet, die durch eine angehängte »2« gekennzeichnet sind, z.B. `SQLServer2`, `StoredProcedure2`, `Database2`, `Column2`. Diese neuen Klassen haben neue Attribute und Methoden, die nur beim SQL Server 2000 zur Verfügung stehen. Der SQL Server 2000 kann aber auch weiterhin mit den alten Klassennamen angesprochen werden; dann stehen lediglich die neuen Funktionen nicht zur Verfügung.

*Tabelle 16.1:
Schnellinfo SQL
Server Distributed
Management
Objects
(SQL-DMO)*

Name und Abkürzung	SQL Server Distributed Management Objects (SQL-DMO)
Name der Komponentendatei	SQLDMO.DLL (Typbibliothek: SQLDMO.RLL)
Interner Name der Typbibliothek	SQLDMO
Helpstring der Typbibliothek	Microsoft SQLDMO Object Library
Hersteller	Microsoft
Lizenzierung	Bestandteil des Microsoft SQL Servers 2000
Besprochene Version	2000.80.194.0 SQL DMO 7 war Version 7.00.623
NT4, Windows2000 und Windows 95/98/ME	Installation Microsoft SQL Server 2000 bzw. der zugehörigen Client-Tools
Dokumentation	MSDN Library [MSL00]

Dokumentation in der MSDN Library**Dokumentation**

DMO ist in der MSDN Library gut dokumentiert. Zu jeder Klasse wird das Vorgehen bei typischen Programmieraufgaben Schritt für Schritt beschrieben, und es gibt sogar sowohl in einem Gesamtmodell als auch in Ausschnitten zu jeder Klasse grafische Darstellungen des Objektmodells. Unverständlichlicherweise hat Microsoft dann wieder daran gespart, zu jedem der dargestellten Programmierabläufe auch den Programmcode mitzuliefern. Die Anzahl der Programmierbeispiele ist eher spärlich. Schade, dabei war die SQL-DMO-Dokumentation auf dem besten Weg, vom Autor dieses Buches als vorbildhaft bezeichnet zu werden.

16.1 DMO-Objektmodell

Objektmodell

Da das Objektmodell jedoch recht geradlinig ist und in vielen Bereichen der Ansicht des SQL Server Enterprise Managers entspricht, ist die Selbsterforschung dieser Komponente einfach. DMO ist kein Meta-Objektmodell und auch nicht dazu geeignet, andere Anwendungen als den SQL Server zu administrieren, auch wenn der Name DMO dies vielen suggeriert. Es gibt in

der Typbibliothek erfreulicherweise zu jeder Einzel-Klasse (nicht aber zu den Collection-Klassen) auch eine Klassendefinition (in Form einer CoClass). Die Collection-Klassen sind lediglich über Interface-Definitionen beschrieben. Mehrfachschnittstellen sind eher selten.

Gemeinsame Mitglieder

Jede DMO-Klasse verfügt über folgende Attribute:

- ▶ `Application` ist ein Verweis auf das Stammobjekt der DMO. **Application**
- ▶ `Parent` ist ein Verweis auf das jeweils übergeordnete Objekt. **Parent**
- ▶ `TypeOf` ist ein Zahlenwert, der die Klasse beschreibt. In der Konstantenliste `SQLDMO_OBJECT_TYPE` ist für jede DMO-Klasse eine eindeutige Zahl definiert. **TypeOf**
- ▶ `UserData` ist ein Zahlenwert, der vom Programmierer frei belegt werden kann. Microsoft schweigt darüber, warum ausgerechnet in diesem Objektmodell eine solche Möglichkeit geschaffen wurde. **UserData**
- ▶ Das Attribut `Properties` verweist auf eine Property-Collection mit Property-Objekten. Jedes Attribut einer bestimmten Klasse (außer den o. g. Standardattributen) ist hier eingetragen. Die Property-Klasse stellt die Attribute `Name`, `Type`, `Get`, `Set` und `Value` bereit. `Type` enthält einen numerischen Wert, der den Datentyp repräsentiert. `Get` und `Set` dienen nicht wie die ADSI-Methoden `Get()` und `Put()` der Veränderung des Attributs, sondern informieren darüber, wie das Attribut verwendet werden kann. `Set = False` bedeutet read-only, `Get = False` bedeutet write-only. Mit `Value` kann das Attribut entsprechend der Einstellung von `Get/`
`Set` gelesen und/oder verändert werden. **Property-Collection**

Laut MSDN-Dokumentation dient die `Properties-Collection` dem Zugriff von Automation Clients und den Eingabehilfen der Entwicklungsumgebungen. Dies ist jedoch falsch, da COM-Automation Clients die `IDispatch`-Methoden der vorhandenen dualen Schnittstellen nutzen und Entwicklungsumgebungen auf Typbibliotheken zurückgreifen. Es ist aber eine interessante Vision, dass Entwicklungsumgebungen sich in Zukunft solcher Collections bedienen, um in (Meta-)Objektmodellen Eingabehilfen anzubieten. So ist die `Properties-Collection` zunächst nur ein Instrument der Erweiterbarkeit zur Laufzeit. Natürlich fehlt dann noch eine `Methods-Collection`. Umgesetzt ist dies schon in WMI (siehe Kapitel 14).

Properties-Collection

SQL Server Distributed Management Objects (DMO) 7.0/2000

Abbildung 16.2:
Erster Ausschnitt
aus dem DMO-
Objektmodell

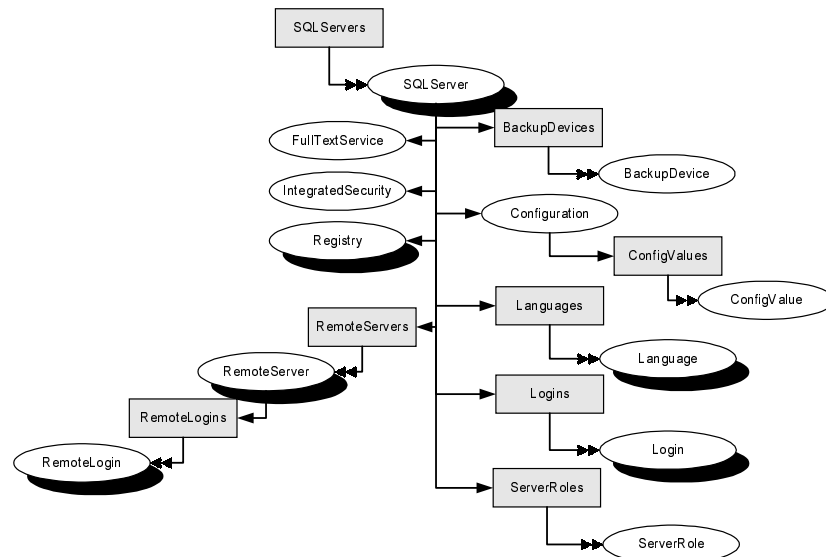
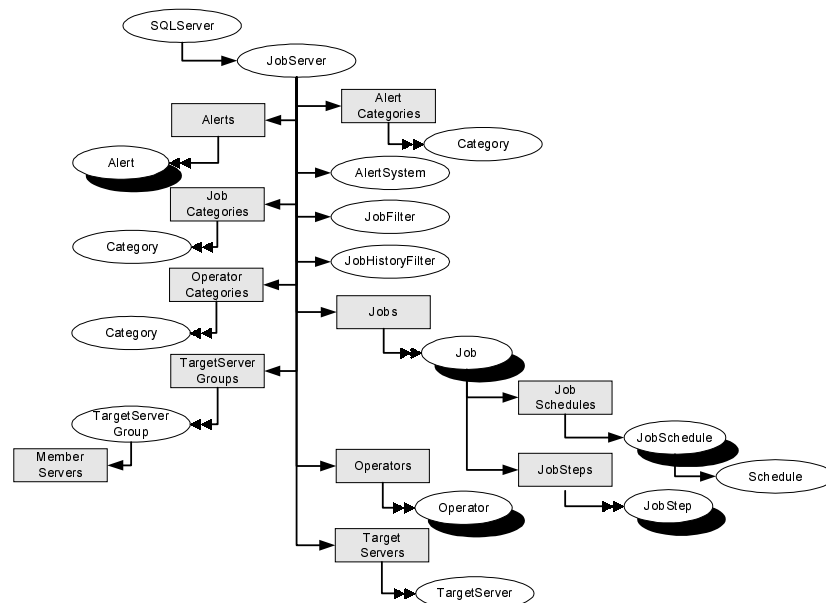


Abbildung 16.3:
Zweiter Ausschnitt
aus dem DMO-
Objektmodell



DMO-Objektmodell

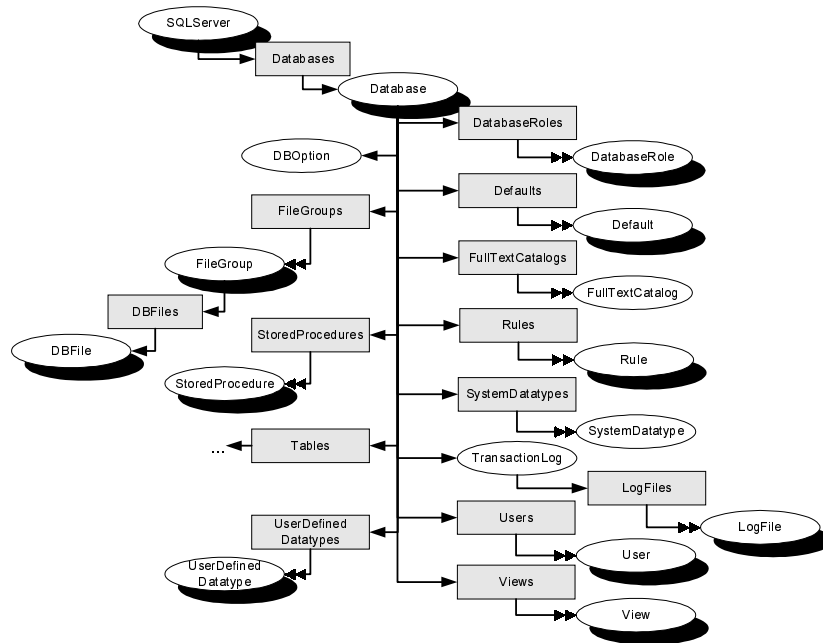


Abbildung 16.4:
Dritter Ausschnitt
aus dem DMO-
Objektmodell

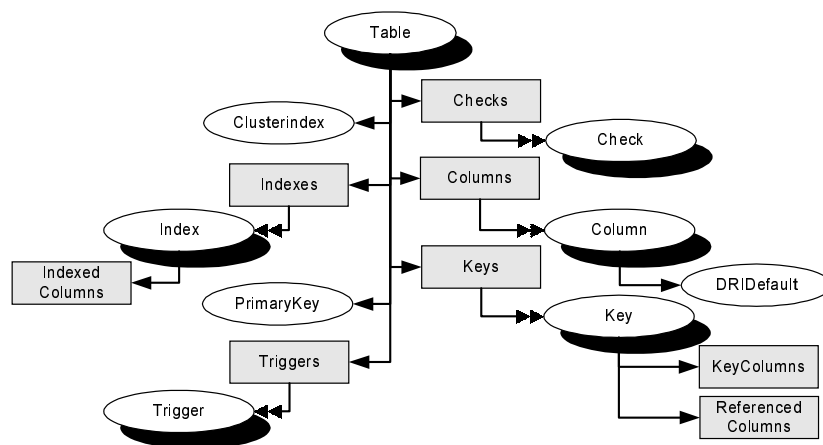


Abbildung 16.5:
Vierter Ausschnitt
aus dem DMO-
Objektmodell

Optional, aber häufig vorkommend, sind folgende Mitglieder:

- ID-Attribut** ► Viele Klassen (außer den Collection-Klassen) verfügen über ein ID-Attribut, das ein Objekt innerhalb einer Collection eindeutig identifizieren soll. Diese ID ist aber keine GUID und weder zwischen verschiedenen Collections noch innerhalb einer Collection über die Zeit eindeutig. Die ID startet in jeder Collection bei 1. Ein neues Objekt erhält die kleinste Zahl ≥ 1 , die gerade frei ist. Wird ein Objekt gelöscht und ein neues angelegt, erhält das neue Objekt die ID des alten. Dieser Mechanismus ist zweifelsohne sehr unbefriedigend. Da natürlich auch Namen wiederverwendet werden können, ist es unmöglich festzustellen, ob ein bestimmtes DMO-Objekt beim erneuten Aufruf dasselbe ist wie beim vorherigen Aufruf.
- Refresh()-Methode** ► Viele Objekte verfügen über eine Refresh()-Methode, um die Attributwerte des Objekts zu aktualisieren. Sie kennen diese Funktion aus dem Enterprise Manager, wo sie in den Kontextmenüs allgegenwärtig ist.
- Weitere Mitglieder in Collections** Collection-Klassen verfügen daneben auch noch wie üblich über die Methode item() und das Attribut count, oft auch über add() und remove(). Item() unterstützt numerische Werte und in den meisten Fällen auch alphanumerische Namen. Die Unterobjekte sind innerhalb der Collection anhand ihres Namens alphabetisch geordnet. Item() liefert bei der Angabe eines numerischen Werts n den Eintrag an der n-ten Position in dieser alphabetisch sortierten Liste. Da sich diese Position natürlich schnell ändern kann, bieten viele Collections auch die Methode ItemByID(id) an. Bitte beachten Sie jedoch die obigen Ausführungen zur unzureichenden Eindeutigkeit dieser ID. Die Suche auf Basis der ID ist jedoch immer noch besser als die auf Basis der Position.

Externe Instanziierung durch den COM-Client

- Viele instanziiertbare Klassen** Eine Besonderheit der DMO ist es, dass fast alle Objekte von außen direkt instanziiertbar sind. Die Objekterzeugung erfolgt also nicht von einem übergeordneten DMO-Objekt aus, sondern wird vom COM-Client vorgenommen. Danach muss das extern instanziierte Objekt mit der Add()-Methode der jeweiligen Collection in die Collection aufgenommen werden, da sonst keine Bindung an das Objektmodell existiert. Die den Objekten entsprechenden Einträge im SQL Server werden auch erst mit Ausführung der Add()-Methode vorgenommen. Während die Instanziierung sehr schnell ist, benötigt die Add()-Methode also vergleichsweise viel Zeit. Wenn Sie den Aufruf von Add() vergessen, dann hat das zur Konsequenz, dass die von Ihnen erzeugten Objekte nur so lange leben wie die betreffenden Objektvariablen. Sie erzeugen jedoch keine Inkonsistenzen oder Memory-Leaks.
- Nicht persistente Klassen** Obige Ausführungen gelten für DMO-Klassen, die persistenten Einträgen im SQL Server entsprechen. Es gibt auch DMO-Klassen, deren Objekte nicht zu persistenten Einträgen im SQL Server werden (z.B. Backup, Transfer). Auch hier erfolgt die Instanziierung extern. Jedoch wird das instanziierte Objekt danach nicht einer Collection hinzugefügt, sondern der Methode eines anderen Objekts als Parameter vorgeworfen (im Fall Transfer) oder aber es wird nach Ausführung eigener Methoden verworfen (im Fall Backup).

Application-Klasse



Abbildung 16.6:
Typbibliothek von
DMO Version 7.0

16.2 Application-Klasse

Die Klasse `Application` bildet das Stammobjekt der DMO, ist aber keineswegs der alleinige Einstiegspunkt in das Objektmodell. In der Regel steigt der Programmierer über die Klasse `SQLServer` ein. Leider ist die Dokumentation dieser Klasse in der MSDN Library unvollständig. Jede DMO-Klasse implementiert einen Rückverweis auf das `Application`-Objekt. `Application` dient vor allem

Application-Klasse

- ▶ zur Ausgabe von Informationen über die DMO-Komponente (s. u.)
- ▶ zur Ermittlung aller verfügbaren SQL Server im Netz (Methode `ListAvailableSQLServers()`). Diese Funktion wird allerdings nur unter NT unterstützt.
- ▶ zur Pflege einer Liste aller SQL Server, die derzeit über DMO verwendet werden (Collection `SQLServers`)
- ▶ zum Zugriff auf die in Servergruppen registrierten SQL Server. Dies bezieht sich auf die im Enterprise Manager angelegten Servergruppen und registrierten Server (Collection `ServerGroups`). DMO sieht vor, dass `ServerGroup`-Objekte selbst wieder eine `ServerGroups`-Collection enthalten. Es ist also möglich, registrierte Server in einer beliebigen Hierarchie zu verwalten.

- ▶ zum Beenden der Verbindung zu allen SQL Servern (Methode `Quit()`)
- ▶ zum Festlegen der Timeout-Zeit beim Setzen eines Locks auf ein Objekt

16.3 SQLServer-Klasse

Klasse Die `SQLServer`-Klasse stellt folgende Funktionalitäten bereit:

- ▶ Aufbau der Verbindung zu einem SQL Server (Methode `Connect()`)
- ▶ Zugriff auf Konfigurationsinformationen
- ▶ Zugriff auf die dem SQL Server untergeordneten Objekte (z.B. `Configuration`, `Replication`, `JobServer`) und `Collections` (z.B. `Databases`, `BackDevices`, `Logins`, `LinkedServers`, `ServerRoles`)
- ▶ Ausführung von Transact-SQL-Befehlen inkl. Empfang von Ergebnismengen und Transaktionssteuerung – das ist alternativ auch auf Datenbankebene möglich (Methoden `ExecuteImmediate()`, `ExecuteWithResults()`, `ExecuteWithResultsAndMessages()`)
- ▶ Starten und Stoppen des SQL Server-Dienstes (Attribut `Status`, Methoden `Start()`, `Stop()`, `Pause()`, `Continue()`)
- ▶ Abfangen von Server-Ereignissen

Verbindungsaufbau

Connect() Entsprechend den zwei Sicherheitsmodi des SQL Servers bietet auch die Methode `Connect()` zwei Verfahren an. NT-Authentifizierung wird verwendet, wenn zuvor das Attribut `LoginSecure` auf `True` gesetzt wurde. Andernfalls erwartet `Connect()` die Übergabe eines Logins und eines Kennworts für die SQL Server-Sicherheit. Wenn kein Servername angegeben wird, wird lokal ein SQL Server gesucht.

```
Connect( [ ServerName ] , [ Login ] , [ Password ] )
```

Das Listing zeigt beide Formen.

```
Dim objServ ' As SQLDMO.SQLServer
Set objServ = CreateObject("SQLDMO.SQLServer")
' --- 1. Möglichkeit: SQL Server-Sicherheit
objServ.LoginSecure = True
objServ.Connect "Sonne", "sa", "egal"
objServ.Disconnect
' -- 2. Möglichkeit: NT-Authentifizierung
objServ.LoginSecure = True
objServ.Connect "Sonne"
say "Verbunden von " & objServ.HostName & " mit Server " & _
objServ.Name & " auf " & objServ.NetName
say "ServerVersion = " & objServ.VersionMajor & "." & _
objServ.VersionMinor
```

Anlegen einer neuen Datenbank

```
say "Status = " & objServ.Status
say "ProcessID: " & objServ.ProcessID
say "ConnectionID: " & objServ.ConnectionID
say "Issysadmin: " & objServ.Issysadmin
' -- Verbindung beenden
objServ.Disconnect
```

*Listing 16.1: Verbindungsaufbau via NT- und SQL Server-Authentifizierung
[dmo_connect.wsf]*

Jede Verbindung sollte beendet werden, wenn sie nicht mehr gebraucht wird. Dazu dient `objServer.Disconnect()`.



Zugriff auf Unterobjekte

Der Zugriff auf die Collections unterhalb eines `SQLServer`-Objekts ist eine Routineaufgabe.

```
Dim objDB ' As SQLDMO.Database
Dim objBackDev ' As SQLDMO.BackupDevice
Dim objServ ' As SQLDMO.SQLServer

' -- Verbindung aufbauen
Set objServ = CreateObject("SQLDMO.SQLServer")
objServ.LoginSecure = True
objServ.Connect "Sonne", "sa", "egal"
' -- Unterobjekte auflisten
say "--- Liste der Datenbanken:"
For Each objDB In objServ.Databases
    say objDB.Name
Next
say "--- Liste der Sicherungsmedien:"
For Each objBackDev In objServ.BackupDevices
    say objBackDev.Name
Next
' usw.
' -- Verbindung beenden
objServ.Disconnect
```

*Listing 16.2: Liste der Datenbanken und Sicherungsmedien auf einem SQL Server
[dmo_server.wsf]*

16.4 Anlegen einer neuen Datenbank

DMO ist eine sehr aufmerksame Komponente, die – sofern der Benutzer keine Vorgaben macht – viele notwendige Schritte automatisch mit Standardwerten vornimmt. So ist das Minimalskript zur Erzeugung einer Datenbank (aufbauend auf einer bestehenden Verbindung) ein Dreizeiler:

**SQLDMO.
Database**

- ▶ Erzeugung einer neuen Instanz der Klasse Database

```
Set oDB = CreateObject("SQLDMO.Database")
```
- ▶ Vergabe eines Datenbanknamens (DMO denkt sich nicht selbst einen aus)

```
oDB.Name = DBNAME
```
- ▶ Anfügen der Datenbank an die Databases-Collection des gewünschten SQL Servers

```
oSQLServer.Databases.Add oDB
```

Beim Aufruf der letzten Anweisung sorgt DMO dafür, dass automatisch eine Datenbankdatei und eine Transaktionsprotokolldatei mit jeweils der Größe 1 MB (mit unlimitiertem Wachstum in 10%-Schritten) angelegt werden.



Wenn Sie Kontrolle darüber haben wollen, wie diese Dateien konfiguriert werden, dann müssen Sie vor dem `Databases.Add()` entsprechende Objekte erzeugen und an die passende Collection des Database-Objekts übergeben. Das Vorgehen ist dabei analog: Externe Instanziierung, Attribute setzen und dann mit `Add()` der Collection hinzufügen.

Beispiel zur Erzeugung einer neuen Datenbank

Datenbankdatei	Im nächsten Listing wird eine Datenbank mit folgenden Einstellungen angelegt: ▶ Es wird eine Datenbankdatei angelegt mit der Startgröße 10 MB und einem möglichen Wachstum bis auf 100 MB in Schritten zu je 5 MB. ▶ Diese Datenbankdatei wird der vom System angelegten Dateigruppe »PRIMARY« angefügt.
Transaktionsprotokolldatei	▶ Das Skript legt dann eine Transaktionsprotokolldatei an mit der Startgröße 5 MB und ohne Wachstum (<code>FileGrowth=0</code>). ▶ Da es keine Dateigruppen für Transaktionsprotokolldateien gibt, wird die Transaktionsprotokolldatei an die Collection <code>TransactionLog.LogFiles</code> des Database-Objekts angefügt.
Databases-Collection	▶ Nun fügt das Skript zunächst die Datenbank an die <code>Databases</code>-Collection an und sorgt dabei für die Erzeugung der Datenbank- und Transaktionsprotokolldateien. Eine neue Dateigruppe kann erst erstellt werden, wenn die Datenbank schon physikalisch existiert. Diese Beschränkung verbirgt der SQL Enterprise Manager beim manuellen Anlegen.
Filegroup	▶ Eine neue Datengruppe entsteht auf Basis einer Instanz von <code>SQLDMO.FileGroup</code>, die an die <code>FileGroups</code>-Collection der Datenbank angefügt wird. ▶ Die zweite Datenbankdatei hat einen Startwert von 10 MB mit einem prozentualen Wachstum um jeweils 50%, das unbegrenzt ist.

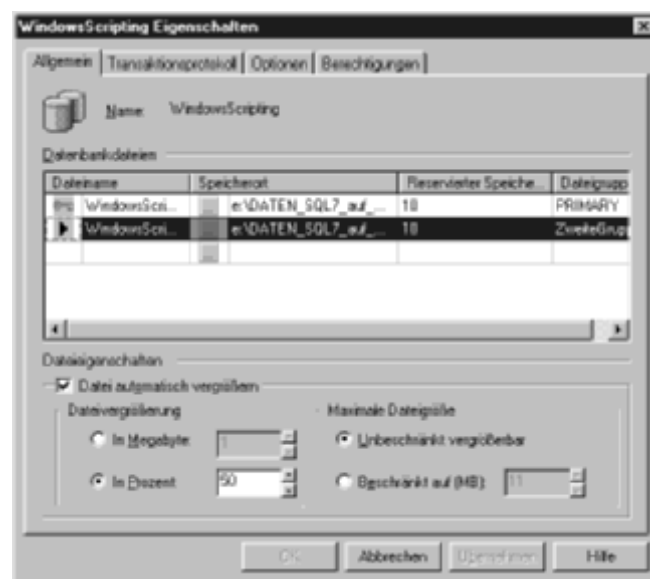
Anlegen einer neuen Datenbank

```
Dim oDB          ' As SQLDMO.Database
Dim oDBFile      ' As SQLDMO.DBFile
Dim oLogFile     ' As SQLDMO.LogFile
Dim oSQLServer   ' As SQLDMO.SQLServer
Dim oFileGroup   ' As SQLDMO.FileGroup
Const DBNAME = "WindowsScripting"
Const SQLDATA_PATH = "e:\DATEN_SQL7_auf_w2k\"
' -- Verbindung aufbauen
Set oSQLServer = CreateObject("SQLDMO.SQLServer")
oSQLServer.Connect "Sonne", "sa", "egal"
' -- Datenbank anlegen
Set oDB = CreateObject("SQLDMO.Database")
' -- Name vergeben
oDB.Name = DBNAME
' -- Erste Datenbankdatei anlegen
Set oDBFile = CreateObject("SQLDMO.DBFile")
oDBFile.Name = DBNAME & "_DB1"
oDBFile.PhysicalName = SQLDATA_PATH & DBNAME & "1.mdf"
oDBFile.PrimaryFile = True
oDBFile.Size = 10
oDBFile.FileGrowthType = 0 ' SQLDMOGrowth_MB
oDBFile.FileGrowth = 5
oDBFile.MaximumSize = 100
' -- Datenbankdatei anfügen
oDB.FileGroups("PRIMARY").DBFiles.Add oDBFile
' -- Transaktionsprotokolldatei anlegen
Set oLogFile = CreateObject("SQLDMO.LogFile")
oLogFile.Name = DBNAME & "_Log1"
oLogFile.PhysicalName = SQLDATA_PATH & DBNAME & ".ldf"
oLogFile.Size = 5
oLogFile.FileGrowth = 0
' -- Transaktionsprotokolldatei anfügen
oDB.TransactionLog.LogFiles.Add oLogFile
' -- Datenbank anfügen
say "Datenbank " & DBNAME & " wird angelegt..."
oSQLServer.Databases.Add oDB
say "Datenbank angelegt..."
' -- Dateigruppe für zweite Datenbankdatei anlegen
Set oFileGroup = CreateObject("SQLDMO.FileGroup")
oFileGroup.Name = "ZweiteGruppe"
' -- Dateigruppe anfügen
oDB.FileGroups.Add oFileGroup
' -- Zweite Datenbankdatei anlegen
Set oDBFile = CreateObject("SQLDMO.DBFile")
oDBFile.Name = DBNAME & "_DB2"
oDBFile.PhysicalName = "e:\DATEN_SQL7_auf_w2k\" & DBNAME & "2.mdf"
oDBFile.PrimaryFile = False
```

```
oDBFile.Size = 10
oDBFile.FileGrowthType = 1 'SQLDMOGrowth_Percent
oDBFile.FileGrowth = 50
oDBFile.MaximumSize = 0
' -- Datenbankdatei an "ZweiteGruppe" anfügen
oDB.FileGroups("ZweiteGruppe").DBFiles.Add oDBFile
say "Zweite DB-Datei angefügt."
```

Listing 16.3: Erzeugung einer neuen Datenbank mit expliziter Erzeugung der Datenbank- und Transaktionsprotokolldatei [dmo_neuedb.wsf]

Abbildung 16.7:
Die Arbeit des
Skripts spiegelt sich
im Eigenschaftsfens-
ter der Datenbank
»Windows-
Scripting« wider.



Nach Erstellung der Datenbank können Sie Informationen über diese erhalten, z.B. die aktuelle Datenbankgröße.

```
say "Festplattenspeicher: " & oDB.SpaceAvailableInMB & " MB" & _
" von " & oDB.Size & " MB"
say "Die Datenbank besteht aus folgenden Datenbankdateien:"
oDB.FileGroups.Refresh
For Each oFileGroup In oDB.FileGroups
    oFileGroup.DBFiles.Refresh
    For Each oDBFile In oFileGroup.DBFiles
        say " Datenbankdatei " & oDBFile.PhysicalName & " (" & _
        oDBFile.SizeInKB & " KB )"
    Next
Next
End
```

Listing 16.4: Ausgabe von Informationen über die neue Datenbank [dmo_neuedb.wsf]

Anlegen einer neuen Tabelle

Der SQL Server zählt bei `oDB.Size` das Transaktionsprotokoll mit. Das Skript sollte also bei obiger Konstellation sofort nach Erzeugung der Datenbank melden:

Festplattenspeicher: 23,80176 MB von 25 MB
Die Datenbank besteht aus folgenden Datenbankdateien:
Datenbankdatei e:\...\NeueDatenbank1.mdf (10240 KB)
Datenbankdatei e:\...\NeueDatenbank2.mdf (10240 KB)

Löschen einer Datenbank

Das Löschen einer Datenbank geht sehr schnell. Sofern Sie den Namen kennen, benötigen Sie nur die folgende Zeile:

Remove()

```
oSQLServer.Databases("DatenbankName").Remove
```

Es gibt (natürlich) keine Sicherheitsabfrage. Beim Skripten haben Sie die volle Verantwortung.



Das folgende Beispiel löscht alle Datenbanken, die mit dem Wort »Test« beginnen – aber erst nach einer selbstprogrammierten Nachfrage.

```
Dim oDB          ' As SQLDMO.Database
Dim oSQLServer   ' As SQLDMO.SQLServer

' -- Verbindung aufbauen
Set oSQLServer = CreateObject("SQLDMO.SQLServer")
oSQLServer.Connect "Sonne", "sa", "egal"

' -- Iteration über alle Datenbanken
For Each oDB In oSQLServer.Databases
    If Left(oDB.Name, 4) = "test" Then
        If MsgBox("Datenbank " & oDB.Name & " löschen?", vbYesNo, _
            "Nachfrage") = vbYes Then
            oSQLServer.Databases.Remove oDB.Name
        End If
    End If
Next
```

Listing 16.5: Selektives Löschen von Datenbanken mit Nachfrage [dmo_removedb.wsf]

16.5 Anlegen einer neuen Tabelle

Das Anlegen einer Tabelle auf Basis einer bestehenden Verbindung zu einem SQL Server vollzieht sich in folgenden Schritten:

**Vorgehen beim
Anlegen einer
neuen Tabelle**

- Instanziierung von `SQLDMO.Table`
`Set oTabelle = CreateObject("SQLDMO.Table")`

- ▶ Festlegung des Namens der neuen Tabelle
`oTabelle.Name = "ComputerListe"`
- ▶ Erzeugung mindestens eines Felds
 - ▶ Instanziierung von `SQLDMO.Column`
`Set oTabelle = CreateObject("SQLDMO.Table")`
 - ▶ Festlegung von Name und Datentyp (`dataType`)
`oSpalte.Name = "ComputerName"`
`oSpalte.dataType = "varchar"`
 - ▶ Festlegung der Länge (`length`) bei Datentypen mit variabler Länge
`oSpalte.length = 15`
 - ▶ Anfügen des Felds an das Tabellenobjekt
`oTabelle.Columns.Add oSpalte`
- ▶ Anfügen des Table-Objekts an die Tables-Collection der gewünschten Datenbank
`oSQLServer.Databases("WindowsScripting").Tables.Add oTabelle`

Feldeigenschaften Für jedes Feld der Tabelle gibt es – abhängig vom Datentyp – zahlreiche Einstellungsmöglichkeiten für die Spalten GENAUIGKEIT (`NumericPrecision`), DEZIMALSTELLEN (`NumericScale`), NULL ZULASSEN (`AllowNulls`), STANDARDWERT (`Default`), IDENTITÄT (`Identity`), ID-STARTWERT (`IdentitySeed`), ID-SCHRITTWEITE (`IdentityIncrement`) und IST ROWGUID (`IsRowGuidCol`).

Beispiel

Tabelleneigenschaften Auch für die Tabelle selbst gibt es zahlreiche Einstellungsmöglichkeiten. Im folgenden Skript wird beispielhaft die Textdateigruppe (das ist der Speicherort für die »großen« Datentypen *image*, *ntext* und *text*) auf eine andere Dateigruppe gesetzt als die übrigen Daten.

```
Dim oSQLServer ' As SQLDMO.SQLServer
Dim oTabelle   ' As SQLDMO.table
Dim oSpalte    ' As SQLDMO.Column

' -- Verbindung aufbauen
say "Verbindungsaufbau..."
Set oSQLServer = CreateObject("SQLDMO.SQLServer")
oSQLServer.Connect "Sonne", "sa", "egal"

' -- Tabelle erzeugen
say "Tabelle anlegen..."
Set oTabelle = CreateObject("SQLDMO.Table")
oTabelle.Name = "ComputerListe"
oTabelle.FileGroup = "PRIMARY"
oTabelle.TextFileGroup = "ZweiteGruppe"
```

Anlegen einer neuen Tabelle

```
say "Spalten anlegen..."

' -- Erzeugung des 1. Felds (Autowert)
Set oSpalte = CreateObject("SQLDMO.Column")
oSpalte.Name = "ComputerNr"
oSpalte.dataType = "int"
oSpalte.Identity = True
oSpalte.IdentityIncrement = 1
oSpalte.IdentitySeed = 1
oSpalte.NumericPrecision = 20
oSpalte.NumericScale = 21
oSpalte.AllowNulls = False
oTabelle.Columns.Add oSpalte

' -- Erzeugung des 2. Felds (String)
Set oSpalte = CreateObject("SQLDMO.Column")
oSpalte.Name = "ComputerName"
oSpalte.dataType = "varchar"
oSpalte.length = 15
oSpalte.AllowNulls = False
oTabelle.Columns.Add oSpalte

' -- Erzeugung des 3. Felds (Memo)
Set oSpalte = CreateObject("SQLDMO.Column")
oSpalte.Name = "Beschreibung"
oSpalte.dataType = "text"
oSpalte.AllowNulls = True
oTabelle.Columns.Add oSpalte

' -- Erzeugung des 4. Felds (Bild)
Set oSpalte = CreateObject("SQLDMO.Column")
oSpalte.Name = "Foto"
oSpalte.dataType = "image"
oSpalte.AllowNulls = True
oTabelle.Columns.Add oSpalte
' -- Anfügen der Tabelle an die gewünschte Datenbank
say "Anfügen der neuen Tabelle..."
oSQLServer.Databases("WindowsScripting").Tables.Add oTabelle
say "Tabelle wurde angefügt!"
' -- Verbindungsabbau
oSQLServer.Disconnect
```

Listing 16.6: Erzeugung einer neuen Tabelle via DMO [dmo_neueTabelle.wsf]

Das Ergebnis des obigen Skripts ist die in der nächsten Abbildung dargestellte Tabelle.

Abbildung 16.8:
Die vom Skript
erzeugte Tabelle
»ComputerListe« in
der Entwurfs-
ansicht im Enter-
prise Manager

Spaltenname	Datentyp	Größe	Genauigkeit	Dezimal	NULL zulassen	Standardwert	Identifiziert	ID-Standardwert	ID-Schrittweite	für Rowguid
ComputerID	int	4	10	0						
ComputerName	varchar	15	0	0						
Beschreibung	text	16	0	0						
Foto	image	16	0	0						

Tabelle löschen So löschen Sie diese Tabelle wieder:

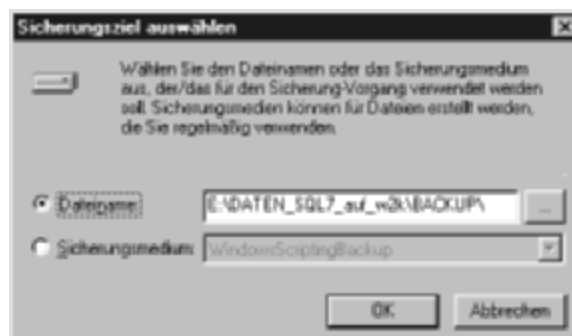
```
oSQLServer.Databases("WindowsScripting").Tables.Remove "ComputerListe"
```

16.6 Backups

Backups automatisieren

Der Erstellung eines Backups dient die Klasse `SQLDMO.Backup`. Grundsätzlich kennt der SQL Server zwei mögliche Sicherungsziele (vgl. folgende Abbildung): eine Sicherungsdatei oder ein Sicherungsmedium.

Abbildung 16.9:
Dialog zur Auswahl
der Sicherungsziele
im Enterprise
Manager



In beiden Fällen bietet der Enterprise Manager eine Mehrfachauswahl an, die DMO auch beim Skripten unterstützt.

Anlegen eines Sicherungsmediums

SQLDMO.Backup-Device

Ein Sicherungsmedium muss vor dem Start des Backups existieren. Das nächste Listing zeigt daher die skriptbasierte Erstellung eines Sicherungsmediums. Jedes Sicherungsmedium braucht als Minimaleigenschaften einen Namen, einen Typ (u. a. steht 2 für eine Festplatte und 5 für ein Magnetband) sowie den Pfad und den Dateinamen, wo die Sicherungsdatei abgelegt werden soll.

```
say "Sicherungsmedium anlegen..."
Set oDevice = CreateObject("SQLDMO.BackupDevice")
oDevice.Name = "WindowsScriptingBackup"
```

Backups

```
oDevice.Type = 2 ' SQLDMODevice_DiskDump
oDevice.PhysicalLocation = SQLDATA_PATH & "BackupMedium1.bak"
oSQLServer.BackupDevices.Add oDevice
say "fertig!"
```

Listing 16.7: Erstellen eines Sicherungsmediums [neues_Sicherungsmedium.wsf]

Backup definieren und ausführen

Ein Backup-Objekt benötigt Werte für das Attribut `Database` und zusätzlich entweder für das Attribut `Devices` oder das Attribut `Files`. Beide Attributnamen sind im Plural, was auf eine Mehrfachauswahl hindeutet. Die Attribute sind jedoch keine Collections, sondern elementare String-Attribute. Mehrfachangaben erfolgen in der Form »[Device1],[Device2],[Device3]« (analog für Files). Microsoft nennt dies eine »SQL-DMO-Multizeichenfolge«.

Notwendige Attribute

Das Backup wird durch die Methode `oBackup.SQLBackup (oSQLServer)` unter Angabe eines Objekts vom Typ `SQLDMO.SQLServer` gestartet. Die Backup-Klasse definiert drei Ereignisse, die während des Programmablaufs abgefangen werden können:

SQLBackup()

- ▶ `Complete(Message as String)` wird ausgelöst, wenn das Backup abgeschlossen wurde.
- ▶ `PercentComplete(Message as String, Percent as Long)` wird während der Abarbeitung in bestimmten Intervallen ausgelöst. Das Intervall wird definiert durch das Attribut `PercentCompleteNotification`, angegeben in Prozentwerten (Standard ist 10%).
- ▶ `NextMedia(Message as String)` wird ausgelöst, wenn ein physikalisches Medium voll ist und gewechselt werden muss.

```
' -- Sicherung definieren
say "Sicherung definieren..."
Set oBackup = CreateObject("SQLDMO.Backup")
' -- notwendige Attribute
oBackup.Database = "Northwind"
oBackup.Devices = "WindowsScriptingBackup"
' Alternative:
' oBackup.Files = "[e:\sql1.bak],[e:\sql2.bak]"
' -- zusätzliche Attribute
oBackup.Action = SQLDMOBackup_Database ' vollständiges Backup
oBackup.BackupSetName = "WS_FullBackup"
oBackup.BackupSetDescription = "WS_FullBackup " & Now
' -- Backup ausführen
say "Backup wird gestartet..."
oBackup.SQLBackup oSQLServer
say "Backup fertig!"
```

Listing 16.8: Erstellung eines Backups [dmo_backup.wsf]

Die Tabelle zeigt die möglichen Werte für das Action-Attribut. Standard ist das vollständige Backup (SQLDMOBackup_Database).

Tabelle 16.2:
Konstanten für
Backup-Action

Symbolische Konstante	Wert	Erläuterung
SQLDMOBackup_Database	0	Vollständiges Backup
SQLDMOBackup_Differential	1	Sicherung der Änderungen seit dem letzten vollständigen Backup
SQLDMOBackup_Files	2	Sicherung bestimmter Files (spezifiziert durch DatabaseFiles bzw. DatabaseFileGroups)
SQLDMOBackup_Log	3	Sicherung des Transaktionsprotokolls

Die folgende Abbildung zeigt den Inhalt des Sicherungsmediums *WindowsScriptingBackup* nach zweimaliger Ausführung eines vollständigen Backups.

Abbildung 16.10:
Inhaltsanzeige
eines Sicherungs-
mediums im Enter-
prise Manager



17 DTS-Package Objects

Die *DTS-Package-Komponente* dient der programmgesteuerten Erzeugung, Bearbeitung und Ausführung von DTS-Paketen. Grundlegende Informationen zum Data Transformation Service (DTS) finden Sie in der Dokumentation des SQL Servers. Dieses Kapitel gibt nur einen sehr kurzen Abriss dieser umfangreichen Komponente.

Name und Abkürzung	DTS-Package Objects
Name der Komponentendatei	dtspkg.dll (Typbibliothek: dtspkg.rll)
Interner Name der Typbibliothek	DTS
Helpstring der Typbibliothek	Microsoft DTSPackage Object Library
Hersteller	Microsoft
Lizenzierung	Teil des SQL Servers 7.0
Besprochene Version	7.00.623
alle Windows-Versionen	Installation Microsoft SQL Server 7.0 bzw. der zugehörigen Client-Tools
Dokumentation	MSDN Library [MSL00]

*Tabelle 17.1:
Schnellinfo
DTS-Package-
Komponente*

Das Objektmodell ist in der MSDN Library dokumentiert [MSL00].

Objektmodell

Das Beispiel zeigt die Ausführung eines in einer Datei gespeicherten Pakets. Zur Veranschaulichung werden zunächst einige Informationen über das Paket ausgegeben, bevor der Start mit `Execute()` erfolgt. Den Ausführungsstatus liefert das Objektmodell in Form von Ereignissen (`onStart()`, `onProgress()`, `onFinish()`, `onError()`) zurück. Leider führt jeder Versuch, die Ereignisse im WSH zu binden, zu einem Fehler.

**Beispiel:
Paket starten**

```
Dim oPak ' As DTS.Package
Dim oTask ' As DTS.Task
' - COM-Objekt instanziiieren
Set oPak = CreateObject("DTS.Package")
' --- Package laden
oPak.LoadFromStorageFile "d:\buch\dts\usercopy.dts", ""
say "Paket " & oPak.Name & ", erzeugt von " & _
oPak.CreatorName & " am " & oPak.CreationDate
say "Liste der Tasks:"
For Each oTask In oPak.Tasks
    say "- " & oTask.Description
Next
say "Starte Paket..."
' Package starten
```

DTS-Package Objects

```
oPak.Execute  
say "Paket wurde ausgeführt"
```

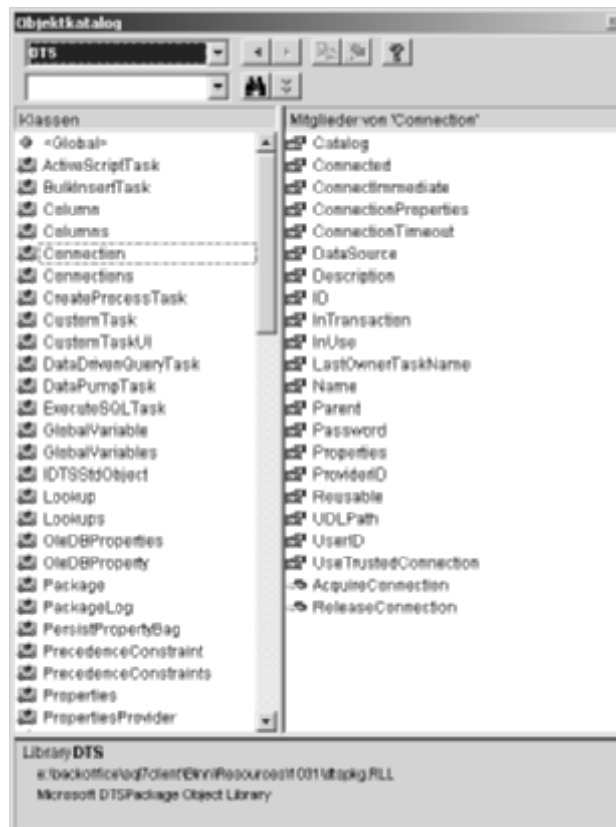
Listing 17.1: Start eines DTS-Pakets [dts_start.wsfl]

Laden vom Server

Alternativ kann ein Paket auch von einem SQL Server geladen werden:

```
LoadFromSQLServer "Sonne2000", "sa", "egal", . . . , "UserCopy"
```

Abbildung 17.1:
DTSPackage-Typ-
bibliothek im VB6-
Objektkatalog



18 Collaboration Data Objects (CDO)

Die *Collaboration Data Objects* (CDO) sind eine COM-Komponente für den Zugriff auf Messaging- und Groupwaresysteme, die MAPI (Messaging Application Programming Interface) als Zugriffsschnittstelle unterstützen.

Die große CDO-Bibliothek

Mit CDO sind insbesondere folgende Funktionalitäten realisierbar:

- ▶ Zugriff auf die Nachrichten und andere Elemente wie Kontakte, Termine und Notizen in Exchange-Postfächern, öffentlichen Ordnern und PST-Dateien
- ▶ Veränderung der Ordnerstruktur in o. g. Mailstores
- ▶ Zugriff auf den Exchange-Verzeichnisdienst und auf Persönliche Adressbücher (PAB)

Die Beispiele in diesem Kapitel basieren im Wesentlichen auf einem Persönlichen Ordner mit dem Namen »Beispiel-Ordner Scripting-Buch«. Dieser Ordner ist mit einigen Beispiel-Einträgen auf der CD enthalten [CD:/umgebung/cdo/scripting-buch_cdo-szenario.pst]. Um die Skripte ohne große Änderungen anwenden zu können, erstellen Sie ein Profil mit dem Namen »HS« und nehmen den o. g. Persönlichen Ordner in dieses Profil auf.



18.1 CDO-Grundlagen

Dieses Unterkapitel beschäftigt sich mit dem grundlegenden Aufbau von Messaging-Systemen, der Architektur von MAPI und den verfügbaren CDO-Versionen.

18.1.1 Aufbau von Messaging-Systemen

In einem Messaging-System werden üblicherweise alle Arten von Einträgen – auch Kontakte, Termine, Notizen, etc. – als Nachricht (Message) bezeichnet, da diese technisch gesehen genauso abgelegt sind wie E-Mails. Unterschieden wird zwischen Nachrichten an Menschen (person-to-person, machine-to-person) und Nachrichten an Systeme (person-to-machine, machine-to-machine). Erstere heißen *Interpersonal Messages* (IPM), letztere *Interprocess Communication Messages* (IPC). IPC-Nachrichten werden im Gegensatz zu IPM-Nachrichten in der Regel von Messaging-Clients nicht dargestellt.

IPM und IPC

Information Stores

Nachrichtenspeicher Die Nachrichten sind üblicherweise in hierarchisch angeordneten Ordnern (auch Container genannt) gespeichert. Der englische Fachbegriff dafür ist *Interpersonal Message Folder Hierarchy*. Eine Ordnerhierarchie heißt auch *Information Store* (kurz: *Infostore*). Jeder Infostore besitzt genau einen Wurzelordner.

Die CDO-Terminologie orientiert sich stark an den in Microsoft Exchange verwendeten Begriffen. Zwischen verschiedenen Messaging-Systemen gibt es einige Nuancen. So spricht man in Lotus Notes von Datenbanken. Das Konzept der Notes-Datenbank entspricht einerseits den Exchange Infostores, andererseits teilweise auch Exchange-Ordnern.

Infostores sind in CDO insbesondere Exchange-Postfächer, öffentliche Ordner auf einem Exchange Server, Gemeinsame Ordner auf einem Microsoft Mail Server und Persönliche Ordner in PST-Dateien. Keine Infostores sind dagegen Persönliche Adressbücher und der Internet Mail-Dienst, da dort keine Nachrichten gespeichert werden.

Semi-strukturierte Datenbank

Strukturiert versus semi-strukturiert Dabei muss an dieser Stelle eine grundlegende Eigenschaft von Messaging- und Groupware-Systemen wie Microsoft Exchange Server und Lotus Notes im Kontrast zu relationalen Datenbanken beachtet werden. Auch wenn Nachrichten in Ordnern Ähnlichkeiten zu Datensätzen in Tabellen aufweisen, gibt es einen wichtigen Unterschied: In relationalen Datenbanken ist eine Tabellenstruktur fest vorgegeben, und alle enthaltenen Datensätze müssen diese Struktur aufweisen. Dagegen kann jede Nachricht in einem Mailordner eigene Felder aufweisen, wobei der Anwender diese zum Teil selbst definieren kann. Dies bezeichnet man auch als semi-strukturiert im Vergleich zu den vollständig strukturierten relationalen Datenbanken.

18.1.2 Messaging Application Programming Interface (MAPI)

MAPI Das Verhältnis von CDO zu MAPI entspricht dem Verhältnis von ADO zu OLE DB: CDO macht das Programmieren von MAPI einfacher. CDO ist eine automationsfähige COM-Komponente und erfordert nicht wie MAPI die Programmierung in C/C++. CDO stellt aber nur diejenigen MAPI-Fähigkeiten bereit, die Messaging-Clients benötigen. Eigene MAPI-Provider können mit CDO nicht erstellt werden. MAPI umfasst auch ein Protokoll zur Kommunikation zwischen Messaging-Client und Messaging-Server.

MAPI-Provider MAPI unterscheidet vier Arten von Providern:

- ▶ *Message Store Provider* (auch *Infostore Provider*) ermöglichen die Verwaltung von Inhalten. Message Store Provider gibt es beispielsweise für MS Exchange Server, Lotus Notes und Mailbox-Dateien (.PST).
- ▶ *Address Book* oder *Directory Provider* ermöglichen die Verwaltung von Personen und Ressourcen.

CDO-Grundlagen

- *Message Transport Provider* ermöglichen den Transport von Daten.
- *Profile Provider* ermöglichen die Verwaltung von MAPI-Profilen mit unterschiedlichen Message Store Providern und Address Book Providern.

MAPI selbst stellt zwei eigene Dienstanbieter automatisch bereit:

PAB und PST

- das Persönliche Adressbuch (.PAB-Dateien)
- Persönliche Ordner (.PST-Dateien)

MAPI-Provider implementieren MAPI-Klassen. CDO ist zum Teil ein Meta-Objektmodell. Die meisten CDO-Klassen sind eine 1-zu-1-Abbildung von MAPI-Klassen. Die Klasse `Message` bildet jedoch eine Vielzahl verschiedener MAPI-Nachrichtenklassen ab. Darüber hinaus sind mehrere CDO-Klassen (`Infostore`, `Folder`, `Message`, `Attachment`, `MessageFilter`, `AddressList`, `AddressEntry`) Meta-Klassen auf Attributebene. Das Objektmodell eines MAPI-Providers können Sie mit dem MAPI-Explorer erkunden.

CDO ist ein Meta-Objektmodell für MAPI

MAPI Explorer

Der *MAPI Explorer* ist ein Freeware-Werkzeug der Firma InTouchSoftware [ITS01], mit dem Sie das Objektmodell eines *Message Store Providers* oder eines *Address Book Providers* erforschen können. Der MAPI Explorer verwendet ein konfiguriertes MAPI-Profil. In der ersten Spalte werden die verfügbaren Infostores und deren Ordnerhierarchie angezeigt. Die zweite Spalte zeigt die enthaltenen Nachrichten. Die dritte Spalte zeigt wahlweise alle Felder der Nachricht, des Infostores, des Ordners oder des Empfängers an.

MAPI-Explorer

Auf der Buch-CD finden Sie den MAPI Explorer unter [CD:/install/tools/MAPI/MAPI Explorer].

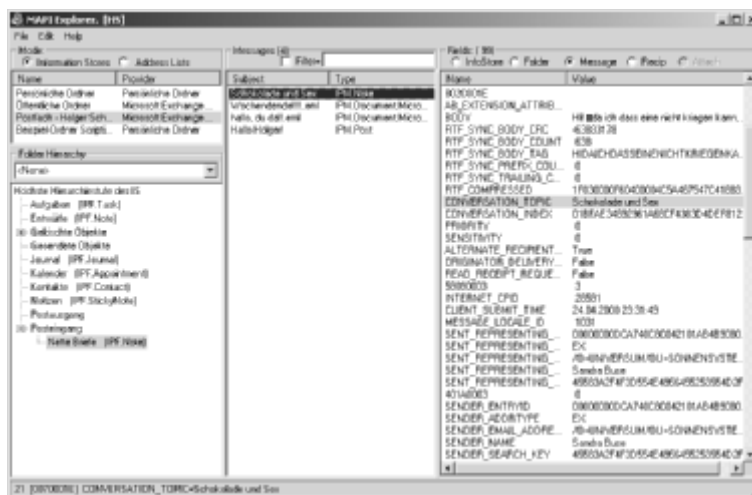


Abbildung 18.1:
Screenshot des
MAPI-Explorers
Version 2.4.0

18.1.3 CDO-Versionen

Seit 1996 hat Microsoft mehrere Versionen von CDO in Umlauf gebracht (siehe nachstehende Tabelle).

Tabelle 18.1:
CDO-Versionen
und Varianten

CDO-Version	Ausgeliefert in	Anderer Name
1.0	Exchange Server 4.0 Outlook 97 (Version 8.0)	OLE Messaging 1.0
1.1	Exchange Server 5.0 Outlook 97 ab Version 8.01	Active Messaging 1.1
1.2	Exchange Server 5.5	
1.21	Outlook 98 (Version 8.5); Outlook 2000 Version 9.0	
CDONTS	Internet Information Server 4.0	
2.0	Windows2000	CDO for Windows2000 / CDOSYS
3.0	Exchange Server 2000	

CDO-Versionen

CDO kam ursprünglich unter dem Namen »OLE Messaging 1.0« auf den Markt. Die Implementierung bestand aus einer Out-Process-Komponente mit dem Namen MDISP32.EXE. Mit der Veröffentlichung des Exchange Servers 5.5 schlug das Marketing bei Microsoft zu, so dass der Begriff *OLE* durch *Active* ersetzt wurde. Active Messaging 1.1 war der Nachfolger von OLE Messaging 1.0. Ab diesem Punkt war die Komponente eine In-process-Komponente, die implementierende hieß OLEMSG.DLL. Erst mit der Veröffentlichung des Exchange Servers 5.5 kam CDO zu dem heutigen Namen. CDONTS und CDO 2.0 sind abgespeckte Versionen von CDO zum Versand von SMTP-Nachrichten. Diese Komponenten werden in Kapitel 20 vorgestellt. Die Programmierung der Version 2000 des Exchange Servers mit CDO 3.0 ist Thema in Kapitel 19.



Die installierte CDO-Version ermitteln Sie mit folgendem Zweizeiler:

```
Set objSession = CreateObject("MAPI.Session")
MsgBox "Installierte CDO-Version: " & objSession.Version
```

Tabelle 18.2:
Schnellinfo Collaboration Data Objects (CDO)

Name und Abkürzung	Collaboration Data Objects (CDO)
Name der Komponentendatei	CDO.DLL
Interner Name der Typbibliothek	MAPI
Helpstring der Typbibliothek	Microsoft CDO 1.2 Library
Hersteller	Microsoft

CDO-Objektmodell

Lizenzierung	Bestandteil von Exchange Server und Outlook
Besprochene Version	1.21 (DLL-Version 5.5.2232.0)
Alle Windows-Versionen	Teil von Outlook oder Exchange Server; kostenloses Add-on; Alternativ ist die Einzelregistrierung möglich durch: regsvr32 CDO.DLL [CD:/install/komponenten/cdo/]
Dokumentation	MSDN Library [MSL00]

Tabelle 18.2:
Schnellinfo Col-
laboration
Data Objects
(CDO)
(Forts.)

18.2 CDO-Objektmodell

Das Objektmodell von CDO ist umfangreich und daher auf drei Grafiken aufgeteilt. Die folgende Tabelle enthält die Erläuterungen zu den einzelnen Klassen.

CDO ist in Teilen ein Meta-Objektmodell. Die Klasse `Message` ist eine Universal-Klasse für beliebige MAPI-Nachrichtenklassen. CDO ist zudem ein Meta-Objektmodell auf Attributebene, da durch die `Fields`-Collection beliebige Attribute aus den zu Grunde liegenden MAPI-Klassen verwendet werden können, ohne dass es eine Entsprechung in Form von Attributen in CDO-Klassen geben muss. Diese Eigenschaft ist insbesondere notwendig, weil die verschiedenen MAPI-Klassen benutzerdefinierte Attribute erlauben.

Meta-Objektmodell

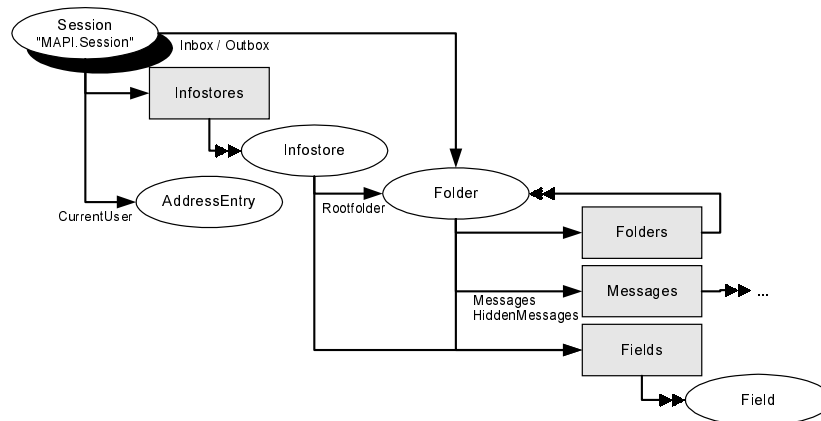


Abbildung 18.2:
Teil 1 des CDO-
Objektmodells
(Zugriff auf Mes-
sage Store-Provider)

Collaboration Data Objects (CDO)

Abbildung 18.3:
Teil 2 des CDO-
Objektmodells
(ab der Messages-
Collection)

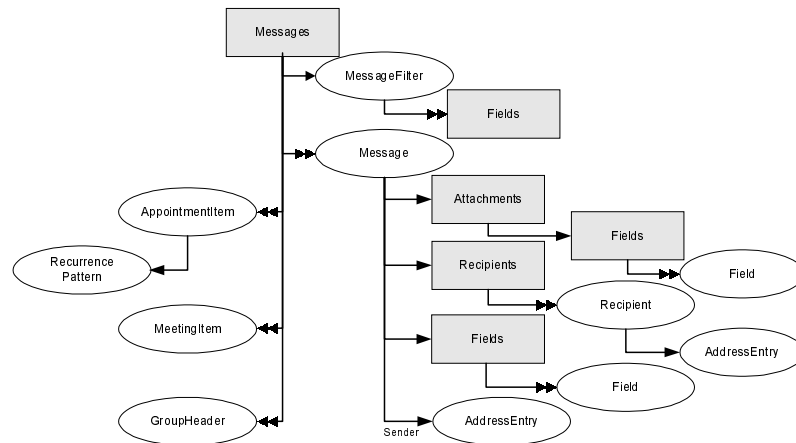


Abbildung 18.4:
Teil 3 des CDO-
Objektmodells
(Zugriff auf
AddressBook-
Provider)

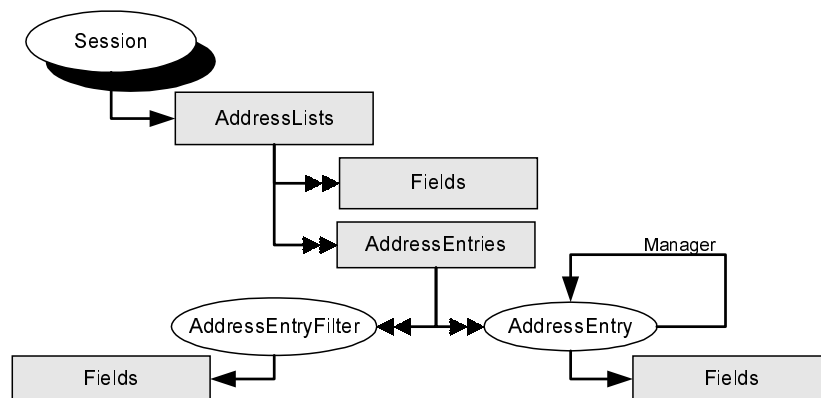


Tabelle 18.3:
Klassen der CDO-
Komponente

Einzel-Klasse / Collection- Klasse	Erläuterung
Session	Session ist das Stammobjekt und als einziges getrennt instanziiert. Es wird benötigt, um eine Sitzung zu einem Mailserver aufzubauen. Ohne diesen Sitzungsaufbau ist kein Zugriff auf die untergeordneten Objekte möglich.
InfoStore InfoStores	Ein Infostore-Objekt repräsentiert einen Infostore. Der Wurzelordner ist über das Attribut <code>RootFolder</code> erreichbar. Alle in einem MAPI-Profil konfigurierten Infostores sind über die <code>Infostores-Collection</code> erreichbar. Der Name eines Infostore-Objekts ist nicht eindeutig. Eindeutiger Bezeichner ist das Attribut <code>ID</code> . Es ist mit CDO nicht möglich, die <code>Infostores-Collection</code> zu verändern. Es können also keine Infostores hinzugefügt oder gelöscht werden.

CDO-Objektmodell

Einzel-Klasse / Collection-Klasse	Erläuterung
Folder Folders	Ein Folder-Objekt repräsentiert einen Ordner innerhalb eines Infostores. Ein Ordner enthält Nachrichten und/oder Unterordner, so dass ein hierarchischer Baum von Ordnern entstehen kann. Die enthaltenen Nachrichten werden über die Collection Messages erreicht, die Objekte der Klasse Message enthält. Für die Ordner ergibt sich eine Rekursion in dem Objektmodell: Folder-Objekte enthalten eine Folders-Collection, die wiederum Folder-Objekte enthält.
Message Messages	Ein Message-Objekt enthält die Informationen einer einzelnen Nachricht, die in einem Ordner abgelegt ist. Alle Message-Objekte sind in einer Messages-Collection angeordnet. Wenn die eindeutige Message-ID eines Message-Objekts bekannt ist, kann diese auch über die Methode GetMessage() direkt angesprochen werden.
Attachment Attachments	Ein Attachment ist der Anhang einer Nachricht. CDO unterstützt vier Typen von Anhängen: Dateien, Dateiverknüpfungen, OLE-Objekte und eingebettete Nachrichten.
Field Fields	Ein Field-Objekt ermöglicht den Zugriff auf alle MAPI-Eigenschaften eines Objekts. Nicht alle MAPI-Eigenschaften sind über die CDO-Klassendefinitionen im direkten Zugriff. Außerdem sind die MAPI-Eigenschaften einiger Objekte erweiterbar. Dazu stellen einige Klassen (AddressEntry, AddressEntryFilter, AddressList, Recipients, Attachment, Folder, InfoStore, RootFolder, Message, MessageFilter) eine Fields-Collection mit Field-Objekten bereit.
Recipient Recipients	Jede Nachricht kann mehrere Empfänger haben. Empfängerinformationen sind auf Grund dieser 1-zu-n-Beziehung in einer Recipients-Collection mit Recipient-Objekten gespeichert.
MessageFilter	Ein MessageFilter-Objekt dient der Einschränkung der Objektmenge in einer Messages-Collection.
AddressList AddressLists	Eine AddressList-Objekt enthält ein MAPI-Adressbuch. Die übergeordnete Collection AddressLists enthält alle gemäß den installierten MAPI-Providern vorhandenen Adressbücher.
AddressEntry AddressEntries	Ein AddressEntry-Objekt repräsentiert einen Eintrag in einem Adressbuch eines AddressBook Providers. Die Klasse AddressList hat eine AddressEntries-Collection.
AddressEntry Filter	Ein AddressEntryFilter-Objekt dient der Einschränkung der Objektmenge in einer AddressEntries-Collection.

Tabelle 18.3:
Klassen der
CDO-Kompo-
nente
(Forts.)

Schleichwege

Neben den im Objektmodell dargestellten Wegen entlang der Objekthierarchie gibt es noch einige »Schleichwege«, die von einem `Session`-Objekt ausgehen:

- Attribute**
 - ▶ Das Attribut `Inbox` liefert einen direkten Zeiger auf das `Folder`-Objekt für den Posteingang.
 - ▶ Das Attribut `Outbox` liefert analog dazu den Zeiger auf das `Folder`-Objekt mit dem Postausgang.
- Zugriff auf ID-Basis**
 - ▶ Die nachfolgenden Methoden liefern unter Angabe einer passenden ID ein Objekt dieses Typs:
 - ▶ `GetAddressEntry(entryID)`
 - ▶ `GetFolder(folderID)`
 - ▶ `GetInfoStore(storeID)`
 - ▶ `GetMessage(messageID,[storeID])`
 - ▶ `GetArticle(articleID)`
- Standardobjekte**
 - ▶ `GetDefaultFolder(OrdnerKonstante)` gibt einen Zeiger auf einen Standardordner zurück. Tabelle 18.4 zeigt die verfügbaren Konstanten.
 - ▶ `GetAddressList (AdressListenKonstante)` liefert einen Zeiger auf eine bestimmte Adressliste. Die Konstanten stehen in Tabelle 18.5.

Tabelle 18.4:
Konstanten für
CDO-Standard-
ordner

Standardordner	Symbolische Konstante	Wert
Kalender	<code>CdoDefaultFolderCalendar</code>	0
Posteingang	<code>CdoDefaultFolderInbox</code>	1
Postausgang	<code>CdoDefaultFolderOutbox</code>	2
Gesendete Objekte	<code>CdoDefaultFolderSentItems</code>	3
Gelöschte Objekte	<code>CdoDefaultFolderDeletedItems</code>	4
Kontakte	<code>CdoDefaultFolderContacts</code>	5
Journal	<code>CdoDefaultFolderJournal</code>	6
Notizen	<code>CdoDefaultFolderNotes</code>	7
Aufgaben	<code>CdoDefaultFolderTasks</code>	8

Tabelle 18.5:
Konstanten für
CDO-Standard-
adressbücher

Adressbuch	Symbolische Konstante	Wert
Global Address List (GAL)	<code>CdoAddressListGAL</code>	0
Persönliches Adressbuch	<code>CdoAddressListPAB</code>	1
GAL und Persönliches Adressbuch	<code>ConstCdoAddressListTotal</code>	2

18.3 CDO- und MAPI-Basiskonzepte

Dieses Kapitel stellt einige übergreifende Konzepte von CDO im Detail vor.

18.3.1 MAPI-Identifikationsnummern

Alle Einträge verfügen in MAPI über eine eindeutige Identifikations-Nummer (ID), die dem Programmierer über CDO offengelegt wird. Diese wird in CDO über das String-Attribut `ID` in Form einer Hexadezimalzahl bereitgestellt. Diese ID wird vergeben, wenn das Objekt erzeugt wird, und ist unveränderbar über die Lebensdauer des Objekts. Collections verfügen nicht über eine ID, da sie stets über ein übergeordnetes Objekt angesprochen werden. Auch die `Session`-Klasse stellt keine ID bereit.

MAPI-IDs

Ein Vergleich auf die Identität zweier Objekte kann in CDO niemals über die Objektzeiger geführt werden.



```
if 01 = 02 then ...
```

Bei jeder Beschaffung eines Zeigers auf einen MAPI-Eintrag wird jedes Mal ein CDO-Objekt angelegt. Ein Vergleich von Objektzeigern ergibt also nur dann Identität, wenn ein Zeiger zuvor mit `Set Z1 = Z2` dupliziert wurde. Zur Prüfung auf Identität ist auch ein direkter Vergleich zweier `ID`-Attribute nicht empfehlenswert.

```
if 01.ID = 02 then ...
```

Es ist in MAPI möglich, dass die zu verschiedenen Zeitpunkten ermittelten IDs einen Teil enthalten, der die exakte Gleichheit auf Binärebene verhindert. Der Vergleich kann korrekt nur über die von verschiedenen CDO-Klassen bereitgestellten Vergleichsmethoden unter dem Namen `IsSameAs()` erfolgen.

IsSameAs()

```
if 01.IsSameAs(02) then ...
```

Allgemein können zwei MAPI-IDs über die Methode `CompareID` im `Session`-Objekt verglichen werden.

```
objSession.CompareIDs(ID1, ID2)
```

Auf Strings basierende Objektnamen spielen in CDO zur Identifikation nur eine untergeordnete Rolle, da MAPI darauf ausgelegt ist, mit einer sehr großen Anzahl von Objekten zu arbeiten, bei denen die für den Benutzer sichtbaren Namen sich wiederholen dürfen.



Bitte beachten Sie: Die ID ist eine weltweit eindeutige Bezeichnung für diesen Ordner. Sofern in Beispielen in diesem Kapitel IDs genannt sind, werden diese auf Ihrem System anders sein!

18.3.2 Gemeinsame Attribute

Attribute Folgende Attribute werden von allen CDO-Klassen (außer von Collection-Klassen und der Session-Klasse) bereitgestellt:

- ▶ **Application:** Application ist nicht wie in anderen Objektmodellen ein Zeiger auf das Stammobjekt, sondern stets nur der String »Collaboration Data Objects«. Dieses Attribut ist eine Hinterlassenschaft aus der Zeit, als CDO noch eine Out-Process-Komponente war.
- ▶ **Class:** Jede MAPI-Klasse hat eine ID, die jede Instanz der Klasse über das Attribut Class zurückliefern kann. So ist feststellbar, zu welcher Klasse ein Objekt gehört. Die IDs zeigt Tabelle 18.6.
- ▶ **ID:** Dieses Attribut ist ein eindeutiger Identifikator für jede Instanz.
- ▶ **Parent:** Parent ist stets der Zeiger auf das übergeordnete Objekt.
- ▶ **Session:** Session ist stets der Zeiger auf das Stammobjekt.

IsSameAs()

Außerdem bieten alle Klassen die bereits erwähnte Methode IsSameAs() an.

Die Session-Klasse ist eine Ausnahme: Da eine ID nicht notwendig ist und ein übergeordnetes Objekt nicht existieren kann, verfügt sie aus der obigen Attributliste nur über die Attribute Application und Class.

Tabelle 18.6:
MAPI-Klassen

MAPI-Klassenname	Klassen-ID	Konstante für Klassen-ID
Session	0	CdoSession
InfoStore	1	CdoInfoStore
Folder	2	CdoFolder
Message	3	CdoMsg
Recipient	4	CdoRecipient
Attachment	5	CdoAttachment
Field	6	CdoField
AddressList	7	CdoAddressList
AddressEntry	8	CdoAddressEntry
AddressFilter	9	CdoAddressFilter
MessageFilter	10	CdoMessageFilter
InfoStores	14	CdoInfoStores

MAPI-Klassenname	Klassen-ID	Konstante für Klassen-ID
Folders	15	CdoFolders
Messages	16	CdoMessages
Recipients	17	CdoRecipients
Attachments	18	CdoAttachments
Fields	19	CdoFields
AddressLists	20	CdoAddressLists
AddressEntries	21	CdoAddressEntries
GroupHeader	25	CdoGroupHeader
Appointment	26	CdoAppointment
MeetingItem	27	CdoMeetingItem
RecurrencePattern	28	CdoRecurrencePattern
Exceptions	29	CdoExceptions
Exception	30	CdoException
MeetingPlanner	31	CdoMeetingPlanner
HiddenMessages	32	CdoHiddenMessages
MsgClassTotal	33	CdoMsgClassTotal

Tabelle 18.6:
MAPI-Klassen
(Forts.)

18.3.3 Kleine und große Auflistungen

CDO unterscheidet bei den Collections zwei Typen: kleine und große Collections. Große Collections sind solche, die eine große Menge von Objekten enthalten können.

CDO-Auflistungen

Im Gegensatz zu den kleinen Collections stellt CDO für große Collections nicht sicher, dass das `Count`-Attribut immer die korrekte Anzahl von enthaltenen Objekten anzeigt. Es ist daher nicht empfehlenswert, mit einer `For...Next`-Schleife über eine große Collection zu iterieren. Kleine Collections sind: `AddressLists`, `Attachments`, `Fields`, `InfoStores` und `Recipients`. Große Collections sind `AddressEntries`, `Folders` und `Messages`. Große Collections unterstützen gegenüber Standard-Collections zusätzliche Attribute für die Navigation innerhalb der Collection (siehe folgende Tabelle).

Wichtiger Hinweis zum `Item`-Attribut: In CDO-Collections starten die Index-Werte bei 1!



Tabelle 18.7:
Zusätzliche Attribute der CDOCollections im Vergleich zu Standard-Collections

Methode	Erläuterung
Add()	Instanziert ein neues Objekt der in dieser Collection enthaltenen Klasse. Add liefert einen Zeiger auf das neue Objekt zurück. Je nach Collection-Klasse ist optional die Angabe von Parametern möglich, mit denen die Instanz initialisiert werden soll. Diese Werte können aber auch nachträglich gesetzt werden.
Delete()	Löscht alle enthaltenen Elemente
GetFirst()	Liefert einen Verweis auf das erste Element in der Collection
GetLast()	Liefert einen Verweis auf das letzte Element in der Collection
GetNext()	Liefert einen Verweis auf das nächste Element in der Collection
GetPrevious()	Liefert einen Verweis auf das vorhergehende Element in der Collection
Sort ([SortOrder], [PropTag])	Sortiert die Collection. Sofern kein Property Tag für ein Attribut angegeben wird, wird die für jede Collection definierte Standardsortiereigenschaft verwendet (z.B. TimeReceived bei der Messages-Collection).

18.3.4 Erweiterbarkeit der Klassen

Erweiterbarkeit Auch in CDO ist ein Mechanismus notwendig, um über die in den CDO-Klassen definierten Attribute hinaus auf eine Vielzahl von Eigenschaften der Einträge in den MAPI-Providern zugreifen zu können. Ein Messaging-System kennt in der Regel eine Vielzahl unterschiedlicher Klassen von Einträgen mit zahlreichen Unterklassen.

Fields Die Erweiterbarkeit wird in CDO durch eine Fields-Collection hergestellt, die in vielen CDO-Klassen (Infostore, Folder, Message, Attachment, Message Filter, AddressList, AddressEntry) vorhanden ist. Provider können zulassen, dass der Anwender selbst benutzerdefinierte Felder hinzufügen kann.

Grundsätzlich unterscheidet MAPI vom System automatisch angelegte *Standardfelder (predefined properties)* und vom Anwender *zusätzlich definierte Felder (custom named properties)*. Die Standardfelder können in CDO weiter unterteilt werden in diejenigen, die in der Klassendefinition enthalten sind und daher im direkten Zugriff stehen, und diejenigen, die nur über die Fields-Collection erreicht werden können. Aus der Sicht von CDO gibt es also insgesamt drei Typen von Feldern.

18.3.5 Property-Tags

Alle Eigenschaften eines Objekts besitzen in MAPI eine eindeutige Field-ID (auch Property-Tag) genannt. Für den Zugriff auf Standardattribute im direkten Zugriff werden diese freilich nicht benötigt. Der Zugriff erfolgt wie üblich über

```
objekt.Attributname
```

Standardattribute, die nicht im direkten Zugriff sind, werden über die Fields-Collection mit Hilfe des korrekten Property-Tags angesprochen:

```
objekt.Fields.Item(property-Tag)
```

Dies kann verkürzt werden auf die Form

```
objekt.Fields(property-Tag)
```

Natürlich können auch die Standardattribute, die im direkten Zugriff sind, auf diese Weise angesprochen werden.

Sie können jedoch nicht – wie sonst (z.B. in ADO) üblich – auch den Begriff `.Fields` weglassen und einfach mit `objekt(property-Tag)` zugreifen.

Diese Form führt zu unerwarteten Ergebnissen, z.B. gibt ein Message-Objekt dann unabhängig vom angegebenen Property-Tag stets den Betreff zurück.

Field-ID



MAPI kennt insgesamt 803 verschiedene Standardattribute, die als symbolische Konstanten in der CDO-Typbibliothek definiert sind. Zur Verwendung in Ihren Skripten finden Sie eine Textdatei mit den passenden Konstantendefinitionen in VB-Form auf der Buch-CD [CD:/konstantenlisten/cdo/cdo_property_tags.txt].

Standardattribute

Property-Tags sind Zahlen, keine Strings. Für Standardattribute haben sie stets einen Wert von größer oder gleich 65536. So kann bei der Verwendung von `Fields.Item(i)` der Zugriff auf einen Indexwert ($0 < i < 65536$) vom Zugriff auf ein Property-Tag unterschieden werden. Damit liegt gleichzeitig das theoretische Maximum der Anzahl von Attributen pro MAPI-Klasse bei 65535.

Wertebereich

Bei einem Message-Objekt steht beispielsweise die ID 3604510 für den Betreff, 3735616 für die Sendezeit, 203358238 für den Verzeichnispfad zum Absender usw.

Beispiel

Ein Belastungs-Test, bei dem einer Nachricht benutzerdefinierte Felder hinzugefügt wurden, endete bei einem öffentlichen Ordner eines Exchange Servers mit 32667 Feldern mit Fehler 800700E (`OUT_OF_MEMORY`) und bei einem Persönlichen Ordner nach 32765 Feldern mit 8004010E (`NOT_ENOUGH_RESOURCES`) – obwohl der Speicherplatz auf den Fest-

Stress-Test

platten in beiden Fällen ausgereicht hätte. Die .PST-Datei war gerade erst 1,5 MB groß geworden.

List_allfields()

Die folgende Routine `list_allfields()` kann angewendet werden, um alle Einträge einer bestimmten `Fields-Collection` aufzulisten.

```
' ### Hilfsroutine: Macht aus einem Array eine CSV-String
Function flat(var) ' as String
On Error Resume Next
Dim i ' As Integer
If IsArray(var) Then ' Array flachklopfen
    flat = ""
    For i = LBound(var) To UBound(var)
        CSVadd flat, var(i)
    Next
Else ' War kein Array
    flat = var
End If
End Function !!!

' ### Hilfsroutine: Liste aller Felder
Sub list_allfields(fields) ' fields as MAPI.Fields
Dim objFI ' As Mapi.Field
say "-----Feldanzahl: " & fields.Count
For Each objFI In fields
    say objFI.Index & ". Feld: " & objFI.Name & "(" & _
objFI.ID & ") = " & flat(objFI.Value) & " (" & objFI.Type & ")"
Next
say "-----"
End Sub
```

Listing 18.1: Listing 5.209: Hilfsroutine zur Auflistung aller Felder in einer `Fields-Collection` [list_allfields.wsf]

Mehrwertige Felder

Die Anwendung der Standardroutine `flat()` auf dem Feldwert ist notwendig, da es an einigen Stellen (z.B. bei Kontakten) mehrwertige Attributwerte gibt, die in Form eines Arrays zurückgeliefert werden.

Die Funktion gibt Index, Feldnamen, Field-ID, Wert und auch den zugehörigen Datentyp aus. MAPI kennt die in der nachfolgenden Konstantendefinitionsliste genannten dreizehn Datentypen, die den Visual Basic-Datentypen sehr ähnlich sind.

```
CONST vbEmpty = 0
CONST vbNull = 1
CONST vbInteger = 2
CONST vbLong = 3
```


CDO-Anmeldung

```
CONST vbSingle = 4
CONST vbDouble = 5
CONST vbCurrency = 6
CONST vbDate = 7
CONST vbString = 8
CONST vbBoolean = 11
CONST vbDataObject = 13
CONST vbBlob = 65
CONST vbArray = 8192
```

Listing 18.2: Konstantenliste für MAPI-Feldtypen (*mapiFieldType*)

MAPI belegt den Feldnamen für die Standardfelder (*predefined MAPI properties*) in der Regel nicht. Benutzerdefinierten Feldern sollten Sie jedoch einen Namen geben.

Keine Feldnamen

18.4 CDO-Anmeldung

Voraussetzung für den Zugriff auf MAPI-Mailstores ist der Aufbau einer MAPI-Sitzung. Eine MAPI-Sitzung nach Instanziierung eines *Sessions*-Objekts wird mit Hilfe der *Logon()*-Methode erzeugt.

```
objSession.Logon( [profileName] [,profilePassword] [,showDialog]
[,newSession] [,parentWindow] [,NoMail] [,ProfileInfo] )
```

Ohne den Aufruf von *Logon()* kommt es beim Zugriff auf untergeordnete Objekte zu dem Fehler *MAPI_E_NOT_INITIALISED*.

Parameter	Erläuterung
ProfileName	Name eines bestehenden MAPI-Profiles
ProfilePassword	Kennwort. Ohne Angabe wird ein Leerstring verwendet.
ShowDialog	Mit diesem Parameter wird spezifiziert, ob Dialogboxen gezeigt werden oder nicht. Standardwert ist <i>True</i> . Wenn dieser Wert auf <i>False</i> gesetzt wurde, <i>Logon()</i> aber nicht alle nötigen Informationen hat, um eine Anmeldung durchzuführen, wird die Sitzung nicht aufgebaut.
NewSession	Dieser Parameter ermöglicht es, eine bestehende MAPI-Verbindung mitzubedenutzen. Wenn dieser Parameter auf <i>True</i> gesetzt wird, sucht CDO nach einer bestehenden MAPI-Verbindung. Wenn keine Sitzung gefunden wird, wird eine neue Sitzung aufgebaut. Wenn der Parameter <i>False</i> ist, wird immer eine neue Sitzung aufgebaut. Standardwert ist <i>True</i> .

Tabelle 18.8:
Parameter der
Logon()-Methode
(*Session*-Klasse)

Tabelle 18.8:
Parameter der
Logon()-
Methode
(Session-
Klasse)
(Forts.)

Parameter	Erläuterung
ParentWindow	Hier kann ein Window-Handle angegeben werden für das Fenster, auf dem eventuell erscheinende Dialogboxen angezeigt werden. 0 bedeutet, dass die Dialogboxen modal sind. -1 setzt das derzeit aktive Fenster als Vaterfenster.
NoMail	Normalerweise steht der CDO-Sitzung der MAPI-Spooler für den Versand und Empfang von Nachrichten zur Verfügung. Dies kann mit der Angabe von <i>True</i> für diesen Parameter unterdrückt werden.
ProfileInfo	Kann für den Aufbau einer Verbindung zu einem Microsoft Exchange Server dazu verwendet werden, ein Profil dynamisch zu erzeugen oder eine anonyme Sitzung zu dem Server aufzubauen.

Logon() Alle Parameter sind optional. Es gibt verschiedene Möglichkeiten, Logon() zu verwenden:

- Ohne Angabe irgendwelcher Parameter erhält der Anwender die Dialogbox »PROFIL AUSWÄHLEN« und kann selbst festlegen, mit welchem Profil er sich anmelden will.

```
objSession.Logon
```

Anmeldung mit Profil

- Um die Anmeldung mit einem bestimmten bestehenden Profil durchzuführen, muss der Profilname angegeben werden.

```
objSession.Logon "HS"
```

Sofern das Profil so konfiguriert wurde, dass ein Passwort erwartet wird, muss dieses angegeben werden. Wird es dennoch nicht angegeben, erscheint die Dialogbox »ANMELDEN«.

```
objSession.Logon "HS", "egal"
```

- Wenn ein Profil nicht zur Verfügung steht oder die Namen vorhandener Profile nicht bekannt sind, kann auch ein temporäres Profil dynamisch erzeugt werden. Dazu erwartet CDO als Parameter *ProfileInfo* einen String, der den Namen des Exchange Servers und den Namen des Postfachs enthält, wobei die Angaben durch einen Zeilenvorschub (vbLf oder chr(10)) getrennt sein müssen.

```
dynProfil = "Sonne" & vbLf & "Martina Koppmann"
```

```
objSession.Logon , , , , , dynProfil
```

Anonymer Zugriff

- Sofern der Exchange Server den anonymen Zugriff auf den öffentlichen Informationsspeicher zulässt, kann die Anmeldung mit CDO anonym erfolgen. Dazu muss *ProfileInfo* den Distinguished Name des Exchange-Server-Konfigurationsordners enthalten und durch zwei Zeilenvorschübe getrennt den String »anon«.

CDO-Anmeldung

```
ProfileInfo = "/o=Universum/ou=Sonnensystem/cn=Configuration/  
cn=Sonne" & vbLf & vbLf & "anon"  
objSession.Logon , , , , , ProfileInfo
```

Mit den übrigen Parametern kann der Sitzungsaufbau weiter konfiguriert werden. Wichtig ist insbesondere der Parameter `showDialog`. Es gibt drei verschiedene Dialogboxen, die CDO während des Verbindungsaufbaus anzeigen kann:

Dialogboxen

- ▶ Die Dialogbox »PROFIL WÄHLEN«. Diese erscheint, wenn kein Profilname und kein `ProfileInfo` angegeben wurde und eine bestehende Sitzung nicht existiert oder nicht wiederverwendet werden darf (`newSession = True`).
- ▶ Die Dialogbox »ANMELDUNG«. Diese erscheint, wenn in dem Profil die Verwendung der Netzwerksicherheit deaktiviert und im Parameter `ProfilePassword` kein Wert angegeben wurde.
- ▶ Eine Dialogbox zur Konfiguration des Exchange-Server-Dienstes erscheint, wenn die in einem `ProfileString` angegebenen Informationen falsch oder unvollständig waren.

Bei der unbeaufsichtigten Ausführung eines Programms mit CDO-Zugriff muss der `showDialog` unbedingt auf `False` gesetzt werden, da eine Reaktion auf die Dialogboxen durch den Benutzer nicht möglich ist. Beim Aufbau der Sitzung fehlende Informationen führen dann dazu, dass die Sitzung nicht aufgebaut werden kann.



Da die Methode `Logon()` keine Fehlermeldung an den COM-Client liefert, muss nach einem Verbindungsaufbau zunächst geprüft werden, ob ein Zugriff auf Objekte des Servers wirklich möglich ist. Eine einfache Möglichkeit ist dabei, den Zugriff auf den Posteingang (Inbox) zu versuchen. Dies leistet die folgende Funktion:

Keine Fehlermeldung

```
' ### Überprüft, ob eine Sitzung aufgebaut ist  
Function check_logon(objSes) ' objSes as MAPI.Session  
Dim objIN ' As Mapi.folder  
On Error Resume Next  
' -- Zugriff auf den Posteingang  
Set objIN = objSes.Inbox  
If Err.Number <> 0 Then  
    check_logon = False  
Else  
    check_logon = True  
End If  
End Function
```

Listing 18.3: Überprüft, ob eine Sitzung aufgebaut ist [check_logon.wsf]

Das folgende Listing zeigt ein Beispiel mit manueller Profilauswahl.

Collaboration Data Objects (CDO)

```
Dim objSession ' AS MAPI.Session
' -- Instanziierung
Set objSession = CreateObject("MAPI.Session")
' -- Sitzungsaufbau
objSession.Logon
If check_logon(objSession) Then
    MsgBox "Anmeldung war OK!"
Else
    MsgBox "Anmeldung ist fehlgeschlagen!"
End If
' -- Sitzungsabbau
objSession.Logoff
```

Listing 18.4: Sitzungsaufbau mit Profilauswahl [cdo_logon_auswahl.wsf]

Im nächsten Skript soll die Anmeldung unbeaufsichtigt erfolgen. Dabei wird auch das Passwort angegeben, da die Verwendung der Netzwerksicherheit deaktiviert ist.

```
Dim objSession ' AS MAPI.Session
' -- Instanziierung
Set objSession = CreateObject("MAPI.Session")
' -- Sitzungsaufbau
objSession.Logon "HS", "egal", False
If check_logon(objSession) Then
    MsgBox "Anmeldung war OK!"
Else
    MsgBox "Anmeldung ist fehlgeschlagen!"
End If
' -- Sitzungsabbau
objSession.Logoff
```

Listing 18.5: Sitzungsaufbau mit dezidiertem Profil [cdo_logon_profil.wsf]



In jedem Fall – egal, ob der Sitzungsaufbau erfolgreich war oder nicht – sollten Sie vor dem Programmende die `Logoff()`-Methode aufrufen!

Weitere Mitglieder der Klasse »Session«

Die weiteren Mitglieder der Klasse `Session` zeigt der folgende Screenshot.

Zugriff auf Infostores



Abbildung 18.5:
Mitglieder der
Klasse »Session«
(VB6-Objekt-
katalog)

18.5 Zugriff auf Infostores

Die Infostores sind in einer Collection angeordnet.

```
Set objSession = CreateObject("MAPI.Session")
objSession.Logon "HS"
say "Verfügbare Infostores:"
' -- Rekursion über alle Infostores
For Each objIS In objSession.InfoStores
    say objIS.Name & ":" & objIS.ProviderName & ":" & objIS.ID
Next
```

Listing 18.6: Liste aller Infostores [cdo_infostores.vbs]

Ergebnis dieser Routine ist eine Liste aller Infostores. In diesem beispielhaften MAPI-Profil sind insgesamt vier Infostores verfügbar. Die ersten beiden sind Teil des Exchange-Server-Dienstes. Infostore 3 und 4 sind jeweils *Persönliche Ordner*, die einen individuellen Namen bekommen haben.

Collaboration Data Objects (CDO)

Verfügbare Infostores:

```
1: Öffentliche Ordner:Microsoft Exchange Server:
00000000...E4E4500
2: Postfach - Holger Schwichtenberg:Microsoft Exchange Server:
0000000038A1BB1005E5101AA1BB08002B2A56C2000...E45002F
3: Beispiel-Ordner Scripting-Buch:Persönlicher Ordner:
0000000038A1BB1005E5101AA1BB08002B2A5...737400
4: Mailverkehr www.IT-Visions.de:Persönlicher Ordner:
0000000038A1BB1005E5101AA1BB08002B2A56C2...2E70737400
```

Listing 18.7: Beispielausgabe des [cdo_infostores.wsfl]-Skripts

Direkter Zugriff über den Namen

Sofern der exakte Name eines Infostores bekannt ist, kann dieser über die Infostores-Collection direkt angesprungen werden:

```
objSession.InfoStores("Beispiel-Ordner Scripting-Buch")
```



Bitte beachten Sie: Die Namen der Infostores und der Ordner sind **case-sensitive**. Sie können zwar mit Outlook keine zwei Ordner erstellen, deren Namen sich nur hinsichtlich der Groß-/Kleinschreibung unterscheiden. Beim Zugriff auf den Ordner macht CDO aber diesen Unterschied.

GetInfoStore()

Wenn Ihnen die ID des Infostores bekannt ist, können Sie auch GetInfoStore(storeID) verwenden.

Vereinfachte Anmeldung

Im Folgenden wird häufig die Funktion cdo_login_is() verwendet, die eine Anmeldung unter einem anzugebenden Profil ausführt und den Zeiger auf einen bestimmten Infostore liefert. Dadurch werden die folgenden Skripte etwas übersichtlicher.

```
Function cdo_login_is(Profile, infostore)
Dim objSession ' As Mapi.Session
Set objSession = CreateObject("MAPI.Session")
objSession.Logon "HS"
Set cdo_login_is = objSession.InfoStores(infostore)
End Function
```

Listing 18.8: Hilfsroutine: Logon und Navigation zu einem bestimmten Infostore
[cdo_login_is.wsfl]

RootFolder

Jeder Infostore besitzt genau einen Startordner. Das Attribut RootFolder liefert einen Zeiger darauf.

18.6 Zugriff auf Ordner

Nach dem Sitzungsaufbau ist die Navigation zu einem Ordner üblicherweise der nächste Schritt. Um zu einem bestimmten Ordner zu gelangen, gibt es folgende Möglichkeiten:

Wege zu einem Ordner

- ▶ schrittweises Herabsteigen im Baum über den Wurzelordner des betreffenden Infostores und die darunterliegenden Folders-Collections
- ▶ objSession.GetFolder(FolderID) mit der eindeutigen ID des Ordners
- ▶ objSession.GetDefaultFolder(OrdnerKonstante) mit der entsprechenden Konstante für einen der Standardordner
- ▶ über objSession.Inbox oder objSession.Outbox direkt zum Posteingang oder -ausgang
- ▶ Das erste Beispiel zeigt den Weg über das Inbox-Attribut:

```
Set objSession = CreateObject("MAPI.Session")
objSession.Logon "HS"
Set objF1 = objSession.Inbox
say "Dieser Ordner heißt " & objF1.Name & " und hat die ID " & _
objF1.ID
```

Listing 18.9: Zugriff auf den Posteingang [cdo_inbox.wsf]

Dies führt etwa zu folgender Ausgabe:

```
Dieser Ordner heißt Posteingang und hat die ID
0000000024AFAE61A0C6D311930D4854E829DF500100817E638237C5D31193084854E8
29DF500000000003F50000
```

Listing 18.10: Ausgabe von Skript cdo_inbox.wsf

Direkter Zugriff

Wenn die ID bekannt ist, kann ein Ordner über die ID angesprungen werden.

GetFolder()

```
Set objF2 =
objSession.GetFolder("0000000024AFAE61A0C6D311930D4854E829DF500100817E
638237C5D31193084854E829DF50000000003F50000")
say "Dieser Ordner heißt " & objF1.Name & " und hat die ID " & _
objF1.ID
```

Listing 18.11: Bindung eines Ordners über eine Folder-ID [cdo_inbox.wsf]

Wenn es Probleme mit dem Auffinden des Ordners gibt, können Sie als zweiten Parameter bei GetFolder() die ID des Infostores, in dem der Ordner liegt, mit angeben, z.B.

```
Set objF2 = objSession.GetFolder("000000...002B050000", _
"00000000038...500")
```

Listing 18.12: Verwendung von GetFolder() unter Angabe der Store-ID [cdo_getfolder.wsf]

Rekursiver Abstieg

Rekursive Ausgabe der Ordner

Wenn der gewünschte Ordner kein Standardordner und die Ordner-ID unbekannt ist, dann besteht nur die Möglichkeit der rekursiven Suche innerhalb eines Infostores, ausgehend vom Wurzelordner des Infostores.

`RekList_Folders()` ist eine nützliche Hilfsroutine, um alle Ordner unterhalb eines Startordners auszugeben. Die Angabe des Attributs `ebene` dient lediglich der Einrückung. Beim Aufruf sollte hier eine 0 übergeben werden. `RekList_Folders()` ist eine rekursive Funktion, wobei die Ebene jeweils um eins erhöht wird, um vor der Ausgabe des Ordnersnamens entsprechend viele Leerzeichen mit der eingebauten VB-Funktion `Space(Leerzeichenanzahl)` zu erzeugen.

```
Sub RekList_Folders(objStartFO, ebene)
Dim objF ' As MAPI.folder
If Not objStartFO Is Nothing Then
    ' eingerückte Ausgabe
    say Space(ebene) & objStartFO.Name
    ' Rekursion über alle Unterordner
    For Each objF In objStartFO.Folders
        RekList_Folders objF, ebene + 1
    Next
End If
End Sub
```

Listing 18.13: Hilfsroutine zur Rekursion über alle Unterordner

Das Beispiel nutzt `RekList_Folders()`, um die komplette Ordnerhierarchie eines Persönlichen Ordners auszugeben.

```
' === Rekursive Liste der Ordner
Dim objSession ' As Mapi.Session
Dim objIS ' As Mapi.infostore
Dim objF ' As Mapi.folder
Set objSession = CreateObject("MAPI.Session")
objSession.Logon "HS"

say "--- Postfach-Ordnerliste"
Set objIS = objSession.InfoStores("Postfach - Holger Schwichtenberg")
RekList_Folders objIS.RootFolder, 0
```

Listing 18.14: Test von `RekList_Folders()` [CDO_Ordner_Liste.wsf]

Beispiel-Ausgabe

Die Ausgabe sieht etwa so aus:

```
Höchste Ebene des Persönlichen Ordners
Addison-Wesley
    Technik
        Vertrag
            Engines
```


Zugriff auf Ordner

Gelöschte Objekte
Hosts
Komponenten
ADSI
CDO
FSO

Listing 18.15: Ausgabe von CDO_Ordner_Liste.wsf

Der Ordner »Höchste Ebene des Persönlichen Ordners« ist der Rootfolder, der keine Nachrichten enthalten kann, sondern nur Unterordner. Dieser Name wird von dem Infostore-Provider (hier: Persönlicher Ordner-Dienst) selbst vergeben. Der Rootfolder heißt bei Exchange-Postfächern »Höchste Hierarchiestufe des IS« und bei öffentlichen Ordnern »IPM_Subtree«.

Suche nach einem Ordner

CDO stellt keine Funktion bereit, nach einem Ordnernamen rekursiv in allen Unterordnern zu suchen. Eine solche Funktion ist aber einfach zu schreiben. RecSearch_Folder() gibt ausgehend von einem Startordner einen Zeiger auf den ersten Ordner zurück, der den gewünschten Namen trägt.

Rekursive Suche

```
Function RecSearch_Folder(objStartFO, OrdnerName)
Dim objF ' As MAPI.folder
If Not objStartFO Is Nothing Then
    ' -- Vergleich ausführen
    If objStartFO.Name = OrdnerName Then
        Set RecSearch_Folder = objStartFO
    Else
        ' -- Rekursion über alle Unterordner
        For Each objF In objStartFO.Folders
            Set RecSearch_Folder = recsearch_folder(objF, OrdnerName)
            ' -- Abbruchbedingung der Rekursion
            If Not RecSearch_Folder Is Nothing Then Exit Function
        Next
        Set RecSearch_Folder = Nothing
    End If
End If
End Function
```

Listing 18.16: Hilfsroutine zur rekursiven Suche über eine Ordnerhierarchie

Diese Routine können Sie so verwenden:

```
' === Ordnersuche
Dim objSession ' As Mapi.Session
Dim objIS ' As Mapi.infostore
Dim objF ' As Mapi.folder
Set objSession = CreateObject("MAPI.Session")
objSession.Logon "HS"
```

Collaboration Data Objects (CDO)

```
say "--- Ordnerliste aus einer PST-Datei"
Set objIS = objSession.InfoStores("Beispiel-Ordner Scripting-Buch")
RecList_Folders objIS.RootFolder, 0
Set objF = recsearch_Folder(objIS.RootFolder, "Feedback")
list_messages objF
End
Set objF = recsearch_Folder(objIS.RootFolder, "CDO-Beispiele")
If objF Is Nothing Then
    say "Ordner wurde nicht gefunden!"
Else
    say "Ordner " & objF.Name & " wurde gefunden. ID=" & objF.id
End If
```

*Listing 18.17: Beispiel zur Verwendung von RecSearch_Folder()
[CDO_OrdnerSuche.wsf]*

Sortierung einer Ordnerliste

Sort() In Abhängigkeit von den Fähigkeiten eines Providers kann eine Ordnerliste mit Hilfe der Methode Sort() einer Folders-Collection nach verschiedenen Kriterien sortiert werden.

```
Set objIS = objSession.InfoStores("Beispiel-Ordner Scripting-Buch")
Set objF = RecSearch_Folder(objIS.RootFolder, "Komponenten")
Set objFF = objF.Folders
objFF.Sort 2 ' Absteigende Sortierung
For Each f In objFF
    say f.Name
Next
```

Listing 18.18: Sortierung einer Ordnerliste [cdo_folder_sort.wsf]



Leider unterstützt der Exchange Server-Provider lediglich die aufsteigende Sortierung bzw. in einigen Fällen auch gar keine Sortierung (z.B. kann die aus den Ordnern »Favoriten« und »Alle Öffentlichen Ordner« bestehende erste Ebene einer Hierarchie von öffentlichen Ordnern nicht sortiert werden). Das obige Beispiel mit einer absteigenden Sortierreihenfolge funktioniert nur auf einem Persönlichen Ordner. Wäre objFF eine Folders-Collection innerhalb einer Hierarchie von öffentlichen Ordnern eines Exchange-Postfachs, würde MAPI den Fehler 80040117 melden: »Too complex«.

Die Sortierung erfolgt immer in dem CDO-Objekt, nicht physikalisch in der Messaging-Datenbank. Daher muss im obigen Beispiel ein objFF als Zwischenobjekt benutzt werden. Folgender Code wäre wirkungslos:

```
objF.Folders.Sort 2
For Each f In objF.Folders
    say f.Name
Next
```

Eng mit der Sortierfunktion verbunden ist ein anderes Problem: Die Semantik bei der Verwendung der `Item()`-Methode einer `Folder-Collection` ist anders als bei `Collections` üblich. `Item(name)` liefert nicht unbedingt einen Zeiger auf einen Ordner dieses Namens, sondern es wird eine Suchfunktion ausgeführt, die die Ordnerliste gemäß ihrem momentanen Sortierkriterium durchsucht und das erste Element zurückgibt, das in dem Sortierkriterium größer oder gleich dem angegebenen Wert ist. Der Zugriff über `Item()` ist also gerade dann natürlich eine absolut ungeeignete Methode, einen Ordner zu finden, wenn die Sortierung nicht möglich ist.



Anlegen und Löschen von Ordnern

Ein neuer Ordner wird über die `Add()`-Methode einer `Folders-Collection` erzeugt und über ein `Delete()` auf einem `Folder-Objekt` selbst gelöscht. Ein `Delete()` auf einer `Folders-Collection` löscht alle Unterordner, lässt aber den Oberordner bestehen. `Add()` erwartet als Parameter den Namen des neuen Ordners.

Add() und Delete()

Im nachfolgenden Skript wird zunächst ein neuer Ordner erstellt. Darunter werden zehn weitere erstellt. Dann wird die neugeschaffene Hierarchie wieder zerstört, indem zunächst einer der Unterordner, dann alle Unterordner und zum Schluss der zu Beginn angelegte Oberordner gelöscht werden. Natürlich hätte man das schneller haben können, denn das Löschen eines Ordners entfernt natürlich auch alle darunterliegenden Ordner.

```
Set objIS = cdo_login_is("HS", "Beispiel-Ordner Scripting-Buch")
Set objF1 = RecSearch_Folder(objIS.RootFolder, "Komponenten")
Set objFF = objF1.Folders
' -- Einen neuen Oberordner
Set objF2 = objFF.Add("Neue Komponenten")
' -- Darunter 10 neue Unterordner
For a = 1 To 10
    objF2.Folders.Add "Dummy-Komponente #" & a
Next
' -- Liste ausgeben
RekList_Folders objIS.RootFolder, 0
' -- Einen Ordner wieder löschen
Set objF3 = RecSearch_Folder(objF1, "Dummy-Komponente #7")
' -- Liste ausgeben
RekList_Folders objIS.RootFolder, 0
' -- Alle Dummy-Unterordner löschen
objF2.Folders.Delete
' -- Liste ausgeben
RekList_Folders objIS.RootFolder, 0
' -- Oberordner auch noch löschen
objF2.Delete
```

Listing 18.19: Ordner hinzufügen und löschen [cdo_folder_new.wsf]

18.7 Zugriff auf die Nachrichten

Sobald ein Zugriff auf ein `Folder`-Objekt vorhanden ist, können die enthaltenen Nachrichten auf sehr einfache Weise über die `Messages`-Collection aufgelistet werden:

```
For Each m In objF.Messages
    ...
Next
```

`list_messages()`

`List_Messages()` ist eine universelle Ausgaberroutine für alle Nachrichten, die sich in einem Ordner befinden, für den ein Zeiger übergeben wurde. Es werden nur die Betreffzeilen, der Nachrichtentext und die Nachrichtenklasse ausgegeben. Bitte beachten Sie die vorherigen Ausführungen in diesem Kapitel zur Ungenauigkeit des `Count`-Attributs.

```
Sub list_messages(objF) ' objF as MAPI.Folder
Dim objM ' As Mapi.Message
Const CdoPR_MESSAGE_CLASS = 1703966
say "Alle Nachrichten im Ordner " & objF.Name
say "(ungefähre Anzahl:" & objF.Messages.Count & ")"
For Each objM In objF.Messages
    say "-----"
    say "Nachricht:" & objM.Subject
    say "(" & objM.Fields(CdoPR_MESSAGE_CLASS) & ")"
    say "-----"
    say objM(1222)
Next
End Sub
```

Listing 18.20: Universelle Ausgaberroutine für Nachrichten in einem Ordner

Ein Anwendungsbeispiel:

```
Dim objSession ' As MAPI.Session
Dim objIS ' As MAPI.InfoStore
Dim objF ' As MAPI.folder
Set objSession = CreateObject("MAPI.Session")
objSession.Logon "HS"
Set objIS = objSession.InfoStores("Beispiel-Ordner Scripting-Buch")
Set objF = RecSearch_Folder(objIS.RootFolder, "CDO-Beispiele")
If objF Is Nothing Then
    say "Ordner wurde nicht gefunden!"
Else
    List_Messages objF
End If
```

Listing 18.21: Ausgabe des Ordners »CDO-Beispiele« mit Hilfe von `List_Messages()` [cdo_messages1.wsf]

Zugriff auf die Nachrichten

Nachrichtenklassen

Das Klassenwesen in Bezug auf Nachrichten ist komplex. MAPI kennt eine Vielzahl unterschiedlicher Nachrichtenklassen. Dabei lassen sich die Klassen noch in vier Typen einteilen:

- ▶ Nachrichtenklassen, von denen die MAPI-Clients Instanzen erstellen können, die als Nachrichten sichtbar sind (z.B. E-Mails, Kontakte, Besprechungsanfragen). Dies sind Nachrichten von Mensch zu Mensch, also Interpersonal Messages (IPMs).
- ▶ Nachrichtenklassen, von denen die MAPI-Clients Instanzen erstellen können, die aber als versteckte Nachrichten gespeichert werden (z.B. Ansichten, Formulare, Agenten)
- ▶ Nachrichtenklassen, die normalerweise nur das System versendet (Bestätigungen, Fehlermeldungen, Warnungen)
- ▶ Nachrichten, die von einem Menschen oder einer Maschine an eine Maschine gesendet wurden (IPCs)

Die Klassen, die Sie selbst mit Outlook erstellen können, zeigt die folgende Tabelle.

Nachrichtentyp	MAPI-Klassen-String
Aufgabe	IPM.Task
Aufgabenanfrage	IPM.TaskRequest
Journaleintrag	IPM.Activity
Kalendereintrag	IPM.Appointment
Besprechungsanfrage	IPM.Schedule.Meeting.Request
Kontakt	IPM.Contact
Notiz	IPM.StickyNote
Allgemeine Nachricht	IPM.Note
Nachricht in öffentlichem Ordner	IPM.Post
Dokument vom Typ x	IPM.Document.x

Tabelle 18.9:
MAPI-
Nachrichtenklassen

Durch die Verwendung von benutzerdefinierten Formularen kann eine beliebige Menge eigener Formularklassen entstehen. Diese neuen Formulare erweitern einen der obigen Grundtypen, sind also Unterklassen der obigen Klassen. Sie erhalten daher einen Namen, der auf obigem Namen basiert. Der Formularname wird durch einen Punkt getrennt angehängt.

Beispiel: IPM.Post.FeedbackEintrag

Formularklasse

Hinzu kommen Nachrichten, die nicht mit Outlook erzeugt werden können, sondern normalerweise nur vom System generiert werden. Einige Beispiele:

Systemnachrichten

Collaboration Data Objects (CDO)

- ▶ `REPORT.IP.M.Note.DR` enthält eine Zustellungsbenachrichtigung (»Delivery«).
- ▶ `REPORT.IP.M.Note.NDR` steht für eine Unzustellbarkeitsnachricht (»Nondelivery«).
- ▶ `REPORT.IP.M.Note.IPNRN` enthält eine Lesebestätigung (»Read«).
- ▶ `REPORT.IP.M.Note.IPNRN` enthält die Nachricht, dass eine Nachricht nicht gelesen wurde (»NotRead«).

CDO ermöglicht Ihnen jedoch, auch solche Nachrichten zu erzeugen!

Versteckte Nachrichten

Darüber hinaus gibt es Klassen für versteckte Nachrichten. Diese sind nicht in der `Messages-Collection` enthalten, sondern in der `Collection HiddenMessages`.

Tabelle 18.10:
Nachrichtenklassen
für versteckte
Nachrichten

Nachrichtentyp	MAPI-Klassen-String
Ansicht	<code>IPM.Microsoft.FolderDesign.NamedView</code>
Agent	<code>IPC.Microsoft.EventBinding</code> und <code>IPC.Microsoft.ICS.EventBinding</code>
Formular	<code>IPM.Microsoft.FolderDesign.FormsDescription</code>
Posteingangsregel	<code>IPM.Rule.Message</code>

CDO-Nachrichtenklassen

Das CDO-Objektmodell stellt jedoch insgesamt nur vier Nachrichtenklassen (`Message`, `AppointmentItem`, `MeetingItem`, `GroupHeader`) bereit. Dabei umfasst die CDO-Klasse `Message` die übrigen MAPI-Nachrichtenklassen. Insofern ist CDO auch ein Meta-Objektmodell. Die Unterscheidung geschieht anhand des Attributs `CdoPR_MESSAGE_CLASS` (1703966), das den MAPI-Klassen-String enthält.

Standardnachrichtenklasse

Jeder Ordner hat eine Standardnachrichtenklasse. Diese steht in dem Folder-Objekt im Attribut `CdoPR_CONTAINER_CLASS` (Property-Tag 907214878). Allerdings sind die Klassenstrings nicht ganz identisch mit den Klassenstrings der Nachrichten: Sie beginnen mit `IPF` statt mit `IPM`.

Nachrichtenfelder

Felder eines Message-Objekts

Neben den verwendeten Attributen `Subject` und `Text` verfügt eine Nachricht über eine Vielzahl weiterer Eigenschaften. Wie zuvor in den Ausführungen über den allgemeinen Aufbau des CDO-Objektmodells beschrieben sind alle Eigenschaften über die `Fields-Collection` eines `Message`-Objekts im Zugriff. Die folgende Tabelle zeigt die Felder, die im direkten Zugriff sind. Die Anzahl der Felder, die das System ablegt, ist stark von Inhalt und Verwendung der Nachricht abhängig. Als Größenordnung sei hier genannt, dass eine mit CDO angelegte Nachricht mit Betreff und einfachem Inhalt 26 Eigenschaften besitzt; Outlook 97 erzeugt beim Anlegen der gleichen Nachricht dagegen 63 Eigenschaften.

Zugriff auf die Nachrichten

Attributname	Erläuterung
FolderID	Eindeutige Identifikation (Mapi-ID) des Ordners
StoreID	Eindeutige Identifikation (Mapi-ID) des Informationsspeichers
Type	Nachrichtenklasse (MAPI Message Class) für die Nachricht
Size	Ungefähre Größe der Nachricht in Bytes
Importance	Dringlichkeit der Nachricht: Standard (Wert 1), niedrig (0) oder hoch (2)
Subject	Betreff der Nachricht
Sender	Verweis auf AddressEntry-Objekt
TimeSent	Zeitpunkt (Datum und Uhrzeit), zu dem eine Nachricht gesendet wurde
TimeReceived	Zeitpunkt (Datum und Uhrzeit), zu dem eine Nachricht empfangen wurde
Text	Text der Nachricht als Zeichenkette
Sent	Das Attribut hat den Wert <i>True</i> , wenn die Nachricht über das Messaging-System gesendet wurde.
Submitted	Das Attribut gibt den Wert <i>True</i> zurück, wenn die Nachricht dem MAPI-System übergeben wurde.
Unread	Das Attribut gibt den Wert <i>True</i> zurück, wenn die Nachricht vom aktuellen Benutzer noch nicht gelesen wurde.
Fields	Das Attribut gibt einen Zeiger auf die Fields-Collection zurück.
Recipients	Das Attribut gibt einen Zeiger auf die Recipients-Collection zurück.
Attachments	Das Attribut gibt einen Zeiger auf die Attachment-Collection zurück.
Categories	Die der Nachricht zugeordnete Kategorie
TimeCreated	Datum und Uhrzeit des ersten Speicherns der Nachricht
TimeExpired	Verfallsdatum der Nachricht
TimeLast Modified	Datum und die Uhrzeit des letzten Speicherns der Nachricht

*Tabelle 18.11:
Wichtigste Attribute
eines Message-
Objekts*

`list_fullmessages()` ist eine Verbesserung der einige Seiten zuvor gezeigten Routine `list_messages()`, da alle Attribute angezeigt werden. Dabei wird die Hilfsroutine `list_allfields()` verwendet.

**Ausgabe aller
Felder**

```
' ### Hilfsroutine: Liste aller Felder
' ### fields as MAPI.Fields
Sub list_allfields(fields) ' fields as MAPI.Fields
Dim objFI ' As Mapi.Field
say "-----Feldanzahl: " & fields.Count
```

Collaboration Data Objects (CDO)

```
For Each objFI In fields
    say objFI.Index & ". Feld: " & objFI.name & "(" & objFI.id & _
    ") = " & flat(objFI.Value) & " (" & objFI.Type & ")"
Next
say "-----"
End Sub

Sub list_fullmessages(objF)
Dim objM ' As Mapi.Message
Dim objFI ' As Mapi.Field
say "Alle Nachrichten im Ordner " & objF.Name
say "(ungefähre Anzahl:" & objF.Messages.Count & ")"
For Each objM In objF.Messages
    list_allfields objM.fields
Next
End Sub
```

Listing 18.22: Hilfsroutine: Liste aller Nachrichten mit allen Feldern [list_fullmessages.vbs]

Die Ausgabe von `list_fullmessages()`, die nachfolgend in einem Ausschnitt gezeigt ist, wird Sie enttäuschen: In keinem Fall besitzt ein Standardattribut einen textlichen MAPI-Namen.

```
3. Feld: (1703966) = IPM.Post (8)
8. Feld: (3604510) = MAPI-Konstanten b (8)
9. Feld: (3735616) = 16.01.00 03:24:40 (7)
13. Feld: (4325406) = Holger Schwichtenberg (8)
22. Feld: (203358238) = /o=UNIVERSUM/ou=SONNENSYSTEM/cn=Recipients/
cn=Holger Schwichtenberg (8)
```



Standardattribute, die nicht im direkten Zugriff sind, können also nur über das Property-Tag angesprochen werden. So steht die ID 3604510 für den Betreff, 3735616 für die Sendezeit, 203358238 für den Verzeichnispfad zum Absender usw.

Direkter Feldzugriff

Fields.Item() Sie können ein Feld über die `Fields`-Collection direkt ansprechen:

```
objM.Fields.Item(3604510)
```

oder wie üblich in der Kurzform

```
objM.Fields(3604510)
```

Änderungen von Feldwerten

Update() Sofern ein Attribut nicht schreibgeschützt ist, ist eine Veränderung der Nachrichteneigenschaften durch eine Wertzuweisung an das Attribut möglich. Die Änderung von Attributwerten muss mit der Methode `Update()` bestätigt werden – also auch hier wieder eine Analogie zur Arbeit mit Recordsets in ADO.

Zugriff auf die Nachrichten

Neue Felder anlegen

Neue Felder (und damit Eigenschaften der Nachricht) kann der Anwender jederzeit mit **Fields.Add()**

```
Fields.Add(Feldname, Datentyp)
```

anlegen. Die Änderung muss mit `objM.Update()` bestätigt werden.

```
Set objIS = cdo_login_is("HS", "Beispiel-Ordner Scripting-Buch")
Set objF = Recsearch_folder(objIS.RootFolder, "CDO-Beispiele")
' -- Iteration über alle Nachrichten
For Each objM In objF.Messages
    ' -- Neues String-Feld anlegen
    Set objFI = objM.Fields.Add("Quelle", 8)
    ' -- Wert setzen
    objFI.Value = "Buch: Windows- und BackOffice-Scripting"
    ' -- Nachricht abspeichern
    objM.Update
    ' -- Neues Feld ausgeben
    say objM.Subject & ":" & objM.Fields("Quelle")
Next
End
' -- komplette Liste ausgeben
list_fullmessages objF
```

Listing 18.23: Erzeugen eines neuen Felds auf Nachrichtenebene [cdo_messagefields_add.vbs]

Das Beispiel ergänzt alle Nachrichten um eine Quellenangabe. Die Ausgabe zeigt, dass das System selbstständig ein Property-Tag (eine negative Zahl!) vergeben hat und der bei `Add()` angegebene Name gespeichert wurde.

Erläuterung des Beispiels

49. Feld: Quelle(-2146041826) = Buch: Windows- und BackOffice-Scripting (8)

Auf dieses benutzerdefinierte Feld kann nun auch über den Namen zugegriffen werden:

```
say objM.Fields("Quelle")
```

In ADO konnte der obige Ausdruck verkürzt werden auf

```
say objM("Quelle")
```

Das ist jedoch in CDO nicht möglich, denn `objM("Name")` liefert stets nur das Subject-Attribut der Nachricht zurück. Das gilt auch dann, wenn der Feldname gar nicht existiert. Dagegen liefert `objM.Fields("Name")` den korrekten Feldwert und meldet den Fehler `MAPI_E_NOT_FOUND` (8004010f), wenn das Feld nicht existiert.



Benutzerdefinierte Felder auf Ordnerebene

Prinzipiell besteht auch die Möglichkeit, Felder auf Ordnerebene anzulegen (Fields.Add() auf einem Folder-Objekt). Der Exchange Server 5.5 und die Persönlichen Ordner unterstützen diese Möglichkeit aber nicht.

18.8 Anlegen neuer Nachrichten in Ordnern

Messages.Add() Die Methode Add() auf einer Messages-Collection ermöglicht die Erzeugung neuer Nachrichten.

```
Set objM =  
objMM.Add( [subject] [, text] [, type] [, importance] )
```

Alle Parameter der Add()-Methode sind optional; die Eigenschaften können auch später per Wertzuweisung an das gleichnamige Attribut gesetzt werden.

```
Set objIS = cdo_login_is("HS", "Beispiel-Ordner Scripting-Buch")  
Set objF = recsearch_Folder(objIS.RootFolder, "FS0-Beispiele")  
say "Anzahl der Nachrichten vorher: " & objF.Messages.Count  
' -- Neue Nachricht erzeugen  
Set objM = objF.Messages.Add  
' -- Typ: Öffentliche Nachricht  
objM.Type = "IPM.Post"  
' -- Absender setzen  
Set objM.Sender = objIS.Session.CurrentUser  
' -- Betreff, Inhalt und Wichtigkeit  
objM.Subject = "www.windows-scripting.de"  
objM.text = "Die offizielle Website zu diesem Buch"  
objM.Importance = 2 ' mapiHigh = hohe Wichtigkeit  
objM.Unread = True ' Als ungelesen markieren  
' -- Änderungen speichern  
objM.Update  
' -- Kontrollausgaben  
say "Anzahl Felder der neuen Nachricht: " & objM.Fields.Count  
say "Anzahl Nachrichten nachher: " & objF.Messages.Count  
list_fullmessages objF
```

Listing 18.24: Anlegen einer neuen Nachricht in einem Ordner in einer PST-Datei [cdo_newmessage.wsfl]

Beachtung verdienen folgende Zeilen:

► Setzen der Nachrichtenklasse

```
objM.Type = "IPM.Post"
```

Anlegen neuer Nachrichten in Ordnern

► Setzen des Absenders

```
Set objM.Sender = objSession.CurrentUser
```

Ohne diese Zuweisung würde die Nachricht ohne Absender erscheinen. Das Attribut `Sender` erwartet ein `AddressEntry`-Objekt (vgl. nächstes Unterkapitel). Damit es dem CDO-Programmierer erspart bleibt, das `AddressEntry`-Objekt des aktuell angemeldeten Benutzers zu suchen, enthält das `Session`-Objekt einen Zeiger darauf. Soll die Nachricht in Stellvertretung einer anderen Person gesendet werden, muss hier ein Zeiger auf deren `AddressEntry`-Objekt zugewiesen werden.

► Für die Wichtigkeit einer Nachricht gibt es drei mögliche Werte:

```
' *** Constant Definitions: mapiImportance
CONST mapiLow = 0
CONST mapiNormal = 1
CONST mapiHigh = 2
```

Löschen von Nachrichten

Analog zum Umgang mit Ordnern können einzelne Nachrichten durch `objM.Delete()` und alle Nachrichten durch ein `Delete()` auf einer `Messages-Collection` eines Ordners gelöscht werden.

Messages.Delete()

```
' -- Nachricht wieder löschen
objM.Delete
say "Anzahl Nachrichten jetzt: " & objF.Messages.Count
' -- Alle Nachrichten löschen
objF.Messages.Delete
say "Anzahl Nachrichten jetzt: " & objF.Messages.Count
```

Listing 18.25: Löschen von Nachrichten

Der oben erwähnte Stress-Test über die `Fields-Collection` sieht dann so aus:

```
Set objIS = cdo_login_is("HS", "StressTest")
Set objF = Recsearch_folder(objIS.RootFolder, "StressTest")
Set objM = objF.Messages.Add
objM.Subject = "Stresstest"
objM.Update
For i = 1 To 65535
    Set fi = objM.Fields.Add("F" & i, 2)
    fi.Value = 10
    If i Mod 1000 = 0 Then say i & ":" & Now
Next
```

Listing 18.26: Stress-Test für Exchange Server. Das Skript versucht, für eine Nachricht 65535 Felder anzulegen. [cdo_newfield_stresstest.wsf]



Sie sollten derartige Experimente nur auf Test-Ordern bzw. -Servern ausführen. Einige Durchläufe der obigen Routine haben gezeigt: Die Persönlichen Ordner gerieten schnell in einen problematischen Zustand, als nach dem Abbruch der Stress-Routine versucht wurde, von Outlook aus auch nur ein einziges weiteres Attribut anzulegen.

Zuordnung von Sendern und Empfängern

Keine Sender und Empfänger

Die bisher erzeugten Nachrichten haben zwei Schönheitsfehler:

- ▶ Es wird kein Absender abgespeichert.
- ▶ Mit den bisherigen Instrumenten können Nachrichten nur in öffentliche Ordner abgelegt, nicht aber per E-Mail versandt werden.

Dazu ist die Beschäftigung mit einem weiteren Teil des CDO-Objektmodells notwendig, dessen Mittelpunkt die *AddressList*- und die *AddressEntry*-Klasse bilden.

18.9 Adressbücher und Adresseinträge

Exchange und Outlook kennen drei verschiedene Typen von Adressbüchern, die alle über CDO angesprochen werden können:

Adressbuchtypen

- ▶ *Globale Adresslisten* des Exchange Servers, die der Verzeichnisdienst des Exchange Servers bereitstellt
- ▶ *Persönliche Adressbücher*, die in Dateien abgespeichert werden (analog zu den Persönlichen Ordnern)
- ▶ So genannte *Outlook Adressbücher*. Dies sind Kontakte-Ordner, die darauf konfiguriert wurden, als Adressbücher angezeigt zu werden. Dabei können sowohl Kontakte-Ordner aus Postfächern und öffentlichen Ordnern als auch aus Persönlichen Ordnern verwendet werden.

```
Set objSession = CreateObject("MAPI.Session")
objSession.Logon "HS"
say "Verfügbare Adressbücher:"
' -- Iteration über alle Adressbücher
For Each objAL In objSession.AddressLists
    say "- " & objAL.Name
Next
```

Listing 18.27: Ausgabe aller verfügbaren Adresslisten [CDO_addresslist1.wsfl]

Die Ausgabe des obigen Skripts könnte so aussehen (dabei sind die letzten drei Einträge Kontakte-Ordner):

```
Verfügbare Adressbücher:
- Globales Adressbuch (Offline)
- Persönliches Adressbuch
- Kontakte
```

Senden einer E-Mail

- Ansprechpartner für dieses Buch
- Planetensheriffs im Sonnensystem

Listing 18.28: Beispielhafte Ausgabe des obigen Skripts

Wenn Sie den auf der CD mitgelieferten Persönlichen Ordner installiert haben, dann verfügen Sie damit auch über den Kontakte-Ordner »Ansprechpartner für dieses Buch«. Nachdem Sie den Outlook-Adressbuch-Dienst Ihrem Profil hinzugefügt haben, können Sie mit folgender Routine die in dem Ordner enthaltenen Kontakte auflisten.

```
Set objSession = CreateObject("MAPI.Session")
objSession.Logon "HS"
say "Adresseinträge in einem Kontakte-Ordner:"
' -- Zugriff auf Adressbuch
Set objAL = objSession.AddressLists("Ansprechpartner für dieses Buch")
' -- Iteration über alle Einträge
For Each objAE In objAL.AddressEntries
    say "- " & objAE.Name & " (" & objAE.Address & ") " & objAE.Type
Next
```

Listing 18.29: Liste aller Adresseinträge in einem Ordner [CDO_addresslist2.wsf]

Ein AddressEntry-Objekt hat natürlich eine ID. Diese kann dazu verwendet werden, Sender und Empfänger einer Nachricht festzulegen.

Sie werden feststellen, dass CDO auf diesem Wege nur auf Einträge zugreifen kann, die über eine gültige E-Mail-Adresse verfügen. Der in dem Ordner enthaltene Eintrag »Jemand ohne E-Mail-Adresse« wird nicht aufgelistet. Für den Zugriff auf Einträge eines Kontakte-Ordners behandeln Sie den Ordner wie einen normalen Ordner und gehen über objF.Messages.

```
Set objIS = objSession.InfoStores("Beispiel-Ordner Scripting-Buch")
Set objF = recsearch_folder(objIS.RootFolder, "Ansprechpartner für dieses Buch")
list_fullmessages objF
```



18.10 Senden einer E-Mail

Das Senden einer E-Mail unterscheidet sich von der Veröffentlichung einer Nachricht in einem öffentlichen Ordner insofern, als ein oder mehrere Empfänger angegeben werden müssen. Dafür stellt das Message-Objekt eine Recipients-Collection bereit. Mit der Methode Add() wird ein neuer Empfänger angefügt.

MAPI-E-Mail versenden

Collaboration Data Objects (CDO)

```
Set objRecipient =  
objRecipColl.Add( [name] [, address] [, type] [, entryID] )
```

Ein Recipient-Objekt kann damit auf drei Arten erzeugt werden:

- ▶ durch die Angabe eines Namens, der anhand des Adressbuchs aufgelöst werden soll. Der Name wird als erstes Attribut übergeben, Adresse und EntryID bleiben leer. Es muss unbedingt im Anschluss an die Erzeugung des Objekts die Methode `Resolve()` aufgerufen werden.

```
Set objR = objRR.Add("Thomas Becker", , )  
objR.Resolve
```

- ▶ durch die Angabe einer vollständigen E-Mail-Adresse. Der Transportdienst sollte der Adresse vorangestellt werden.

```
Set objR = objRR.Add(, "SMTP:tobeko@gmx.de", , )
```

- ▶ durch die Übergabe der ID eines `AddressEntry`-Objekts. Hierbei muss nur die `EntryID` (nicht aber ein Zeiger) angegeben werden.

```
Set objR = objRR.Add(, , , objSession.CurrentUser.ID)
```



Der Parameter `Type` dient der Spezifikation, wo der Empfänger im Nachrichtenkopf eingetragen werden soll: To-Feld (`Type = 1`), CC (`2`) oder BCC (`3`).

```
' === Erzeugen einer neuen Nachricht  
Dim objF ' As MAPI.folder  
Dim objM ' As MAPI.Message  
Dim objSession ' As MAPI.Session  
Dim objR ' As MAPI.Recipient  
Dim objRR ' As MAPI.Recipients  
Const CdoTo = 1  
Const CdoCc = 2  
Const CdoBcc = 3  
Set objSession = CreateObject("MAPI.Session")  
objSession.Logon "HS"  
Set objF = objSession.Outbox  
' -- Neue Nachricht  
Set objM = objF.Messages.Add  
objM.Subject = "Star-Jurist oder Fernseh-Star?"  
objM.Text = "Na, hast Du Dich jetzt für eine der beiden Karrieren  
entschieden?"  
Set objRR = objM.Recipients  
' --1. Empfänger erzeugen (im TO)  
Set objR = objRR.Add(, "SMTP:tobeko@gmx.de", CdoTo)  
' --2. Empfänger erzeugen (im CC)  
Set objR = objRR.Add("Thomas Becker bei NTV", , CdoCc)
```

Arbeit mit Attachments

```
objR.Resolve
' --3. Empfänger erzeugen (im BCC)
Set objR = objRR.Add(, , CdoBcc, objSession.CurrentUser.ID)
If objR Is Nothing Then
    MsgBox "Name kann nicht aufgelöst werden!"
End If
' -- Speichern und Senden
objM.Update
objM.Send
say "Nachricht gesendet!"
```

Listing 18.30: Erzeugen einer neuen E-Mail-Nachricht [cdo_sendmail.wsf]

18.11 Arbeit mit Attachments

CDO unterstützt auch die Arbeit mit Nachrichtenanhängen. Ein Message-Objekt enthält eine Attachments-Collection, die für jeden Anhang ein Attachment-Objekt enthält. Die Fähigkeiten des Attachment-Objekts sind abhängig vom Typ des Anhangs. Die unterstützten Typen zeigt die folgende Tabelle. Jedes Attachment-Objekt besitzt ein Source-Attribut, das sich jedoch in seiner Bedeutung beim Anlegen und beim Auslesen von Attachments unterscheidet.

E-Mail-Anhänge

Symbolische Konstante	Wert	Inhalt von Source beim Anlegen	Inhalt von Source bei bestehenden Nachrichten
CdoFileData	1	Pfad zu einer Datei	Leer
CdoFileLink	2	Pfad zu einer Datei	Pfad zu einer Datei
CdoOLE	3	Pfad zu einer ActiveX-Dokumentendatei	Leer
CdoEmbeddedMessage	4	MessageID einer Nachricht	Zeiger auf ein Message-Objekt

Tabelle 18.12: Attachment-Typen in CDO

Nachrichtenanhänge erzeugen

Attachments werden einer Nachricht durch Ausführung der Methode Add() auf einer Attachments-Collection hinzugefügt. Add() erwartet im ersten Parameter einen Namen für das Attachment, im zweiten den Typ, im dritten einen Zahlwert als Positionsangabe und zum Schluss den Wert für Source, also in der Regel eine Pfadangabe. Wenn die Angabe der Position fehlt, werden die Attachments in der Reihenfolge des Hinzufügens in die Nachricht eingeordnet.

Senden einer Nachricht mit Attachments

```
Dim objIS ' As Mapi.infostore
Dim objF ' As Mapi.folder
Dim objM ' As MAPI.Message
```

Collaboration Data Objects (CDO)

```
Dim objA ' As MAPI.Attachment
Dim apfad ' As String ' Pfad zum Attachment
Dim n ' As Integer ' Anzahl der Attachments
Set objIS = cdo_login_is("HS", "Beispiel-Ordner Scripting-Buch")
Set objF = recsearch_Folder(objIS.RootFolder, "Ordner mit Anhängen")
Const CdoFileData = 1
' -- Nachricht erzeugen und wichtigste Attribute setzen
Set objM = objF.Messages.Add
objM.Type = "IPM.Post"
' -- Benutzer in einer Schleife nach Attachments fragen
n = 0
apfad = "d:\buch\docs\test.doc"
Do
    apfad = InputBox("Bitte Pfad zum Attachment eingeben! Leer=Ende", _
        "Anhang anfügen", apfad)
    If apfad = "" Then Exit Do
    n = n + 1
    Set objA = objM.Attachments.Add("Anhang #" & n, n, CdoFileData, _
        apfad)
    say "Anhang #" & n & ": " & objA.Source & " angefügt!"
Loop
' -- Standardattribute setzen
Set objM.Sender = objIS.Session.CurrentUser
objM.Subject = "Eine von Ihnen selbst erzeugte Nachricht mit Attachments!"
objM.Text = "Diese Nachricht enthält " & n & " Anhänge!"
' -- Änderungen speichern
objM.Update
End Sub
```

Listing 18.31: Erzeugung einer Nachricht mit Datei-Anhängen (Typ 1), wobei die Attachments durch Interaktion mit dem Benutzer festgelegt werden [cdo_newattachments.wsf]

Nachrichtenanhänge auflisten und abspeichern

Liste der Nachrichten mit ihren Anhängen

Das Auflisten aller Attachments in einem Ordner ist eine leichte Übung. Sie müssen lediglich noch wissen, dass Anhänge der Typen 1 und 3 mit der Methode `WriteToFile()` gespeichert werden können. Bei Typ 4 (eingebettete Nachrichten) erhalten Sie über das `Source`-Attribut ein `Message`-Objekt, das Sie mit dessen Mitgliedern weiterverarbeiten können. Typ 2 (Verknüpfung) liefert nicht mehr als den Verknüpfungspfad im `Source`-Attribut. Ein Speichern ist hier natürlich nicht möglich, da die Nachricht den eigentlichen Inhalt des Anhangs ja gar nicht enthält.

```
Dim objIS ' As Mapi.infostore
Dim objF ' As Mapi.folder
Dim objM ' As MAPI.Message
Dim objA ' As MAPI.Attachment
```


Arbeit mit Attachments

```
Dim pfad ' As String ' Pfad zum Attachment
Set objIS = cdo_login_is("HS", "Beispiel-Ordner Scripting-Buch")
Set objF = recsearch_Folder(objIS.RootFolder, "Ordner mit Anhängen")
' --- Iteration über alle Nachrichten
For Each objM In objF.Messages
    say "-----"
    say "Nachricht:" & objM.Subject
    say "Zahl der Attachments: " & objM.Attachments.Count
    ' --- Iteration über alle Anhänge
    For Each objA In objM.Attachments
        say "- " & objA.Name & " (" & objA.Type & ") "
        Select Case objA.Type
            Case 1 Or 3:
                pfad = "d:\buch\cdo\attachments\" & objA.Name
                objA.WriteToFile pfad
                say "Gespeichert unter " & pfad
            Case 2:
                say "Verknüpfung zu " & objA.Source
            Case 4:
                say "Eingebettete Nachricht '" & objA.Source.Subject & "'"
            End Select
        Next
    Next
Next
```

Listing 18.32: Liste der Attachments in einem Ordner [cdo_listattachments.wsf]

19 Komponenten für den Exchange Server 2000

Während die Exchange-Gemeinde beim Microsoft Exchange Server 5.5 auf einfache, COM-basierte Programmierschnittstellen lange warten musste, wartet der Exchange Server 2000 (interne Versionsnummer: Exchange Server 6.0) von Beginn an mit einer Vielfalt von Komponenten auf, die eng miteinander verzahnt sind. Zusammen bieten sie meist mehrere Wege, um zu einem Ziel zu kommen.

Vielfältige Möglichkeiten

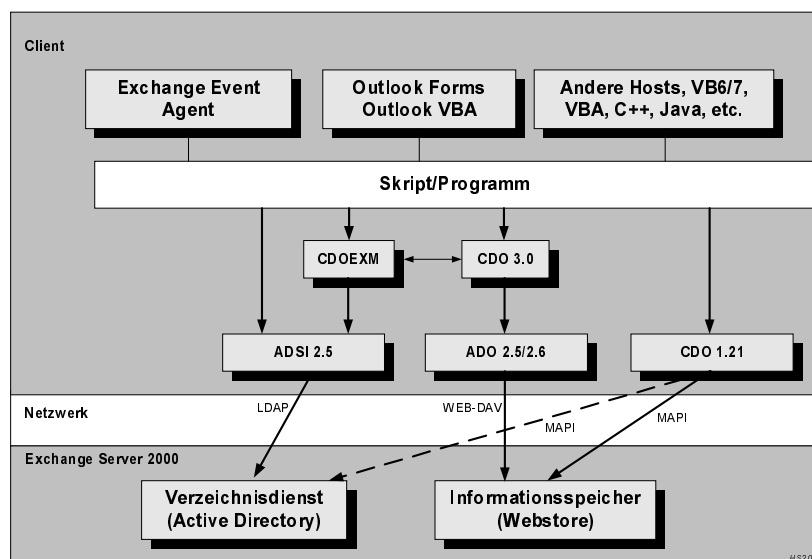


Abbildung 19.1: Überblick über die wichtigsten Komponenten zur Automatisierung von Exchange 2000

19.1 Überblick

19.1.1 Webstore

Der Informationsspeicher (Information Store) heißt im Exchange Server 2000 *Webstore* (auch: Web Storage System). Der Exchange Webstore ist eine semi-strukturierte Datenbank, in der unterschiedliche Typen von Nachrichten gespeichert werden können. Nachrichten sind nicht nur klassische E-Mails, sondern Dateien jeden beliebigen Typs. Der Webstore ist wie ein Dateisystem in Ordner- und Unterordner organisiert. Es gibt zwei Typen von Ordnern: öffentliche und private Ordner.

Web Storage System

Komponenten für den Exchange Server 2000

- ▶ *Private Ordner* sind eindeutig einem Benutzer zugeordnet und bilden das Postfach des Benutzers.
- ▶ *Öffentliche Ordner* sind Teil einer öffentlichen Ordnerhierarchie und stehen im Standard allen Benutzern zur Verfügung. Der Zugriff auf öffentliche Ordner kann jedoch auch beschränkt werden.

Verzahnung mit dem Active Directory

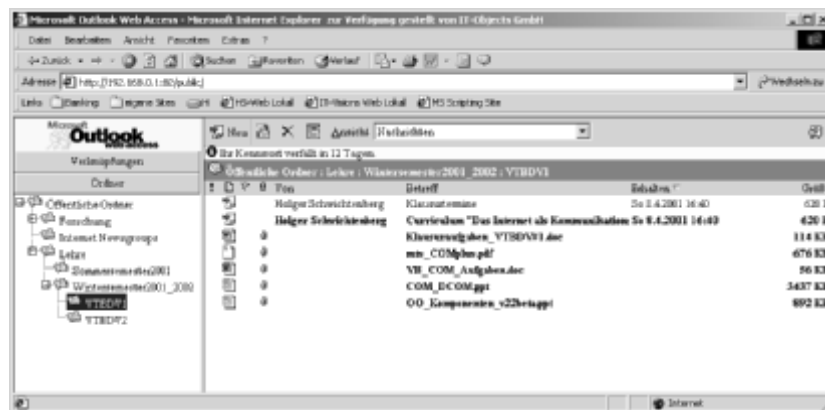
Während der Exchange 5.5-Informationsspeicher klar vom Verzeichnisdienst getrennt war, gibt es jetzt eine enge Verzahnung zwischen dem Webstore und dem Active Directory. So gibt es für jeden öffentlichen Ordner auch einen Eintrag im Active Directory, unterhalb des Containers CN=Microsoft Exchange System Objects, der sich direkt unterhalb der Wurzel des Default Naming Context des Active Directories befindet (z.B. LDAP://sonne2000/CN=Microsoft Exchange System Objects,DC=IT-Visions,DC=de)

Es gibt für den Benutzer drei Möglichkeiten, auf den Webstore zuzugreifen:

- ▶ über einen MAPI-Client (z.B. Microsoft Outlook)
- ▶ über Outlook Web Access (OWA)
- ▶ über das Windows-Dateisystem.

Die ersten beiden Möglichkeiten sind von Exchange 5.5 bekannt. Neu ist der Zugriff über das Dateisystem.

Abbildung 19.2:
Ansicht eines
Ordners via Outlook
Web Access



Laufwerk M: Zugriff über das Dateisystem

Bei der Installation des Exchange Servers wird auf dem Server ein Laufwerk M: im Dateisystem hinzugefügt. Dieses Laufwerk ist ein virtuelles Laufwerk, das ein Browsen durch den Webstore mit dem Windows Explorer (Dateisystem-Explorer) ermöglicht. Diese Funktion wird von dem so genannten *Exchange Installable File System (ExIFS)* bereitgestellt. Wenn der Laufwerksbuchstabe M: schon existiert, wird der nächsthöhere freie Laufwerksbuchstabe verwendet.

Überblick

E-Mails bekommen in der Dateisystemansicht mit dem ExIFS als Dateinamen den Betreff der Nachricht verbunden mit der Dateiextension .EML. In dieser Ansicht können – wie in einem Ordner auf der Festplatte – alle möglichen Veränderungen (Dateien hinzufügen, umbenennen, ändern, löschen; Ordner hinzufügen, umbenennen, löschen) ausgeführt werden. Über das Dateisystem besteht aber kein Zugriff auf die vielfältigen Attribute, die Exchange in einer Nachricht ablegen kann. Das Laufwerk M: kann auch freigegeben werden, so dass von entfernten Computern auf dieses Laufwerk zugegriffen werden kann.

ExIFS

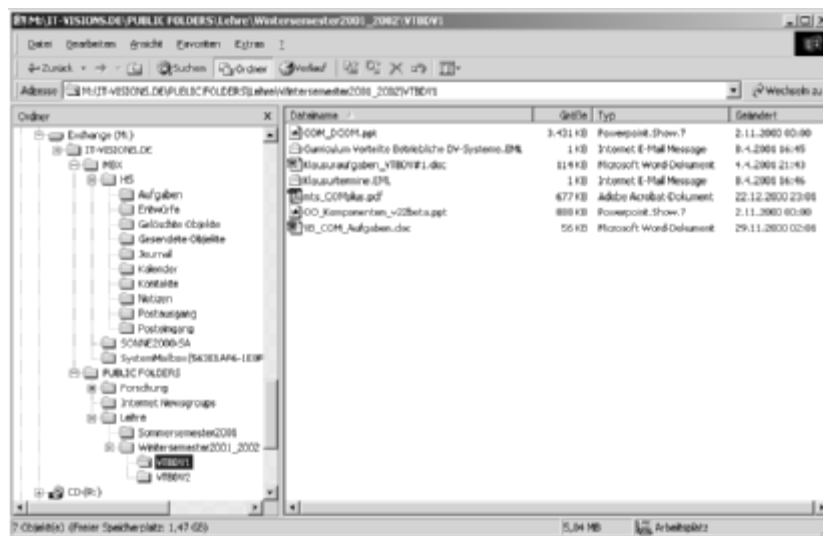


Abbildung 19.3: Ansicht des Webstores im Dateisystem. Unterhalb des Ordners »MBX« werden die Postfächer gezeigt; unterhalb von »PUBLIC FOLDERS« die öffentlichen Ordner.



Abbildung 19.4: Ansicht des Webstores via HTTP im Browser

Eine solche Freigabe kann auch eine Webfreigabe sein. Eine Webfreigabe ist ein virtuelles Verzeichnis im Microsoft Internet Information Server (IIS). Auf diese Weise können Benutzer, die keinen Zugriff auf die Microsoft-Netzwerkprotokolle haben, auf das M:-Laufwerk via HTTP zugreifen. Die Ansicht ermöglicht allerdings nur den Lesezugriff.

Administration des Webstores

Der Webstore kann über den Exchange 2000 System-Manager, Microsoft Outlook und das Dateisystem verwaltet werden. Über alle drei Wege können öffentliche Ordner angelegt, kopiert, verschoben und gelöscht werden. Nur über den System-Manager und Outlook können auch die Eigenschaften der Ordner verändert werden. Die privaten Ordner, die Teil eines Postfachs sind, können nur über Microsoft Outlook und das Dateisystem verwaltet werden.

Tabelle 19.1:
Verwaltungsfunktionen in verschiedenen Exchange 2000-Tools

	Exchange 2000 System-Manager	Outlook 2000	Outlook Web Access 2000	Dateisystem (M:-Laufwerk)	HTTP-Zugriff auf M:-Laufwerk
Ordnerhierarchie ansehen	Ja	Ja	Ja	Ja	Ja
Ordnerinhalte ansehen	Nein	Ja	Ja	Ja	Ja
Ordnerhierarchie verändern	Ja	Ja	Ja	Ja	Nein
Ordnerinhalte verändern	Nein	Ja	Ja	Ja	Nein
Ordner-eigenschaften sehen und verändern	Ja	Ja	Nein	Nein	Nein
Nachrichteneigenschaften sehen und verändern	Nein	Ja	Nein	Nein	Nein
Datenbanken verwalten	Ja	Nein	Nein	Nein	Nein

System-Manager

Der System-Manager verwaltet zusätzlich die physikalischen Exchange-Datenbanken. Eine Exchange-Datenbank liegt in Form einer .EDB-Datei auf der Festplatte vor und heißt im System-Manager »Speicher«. Eine Datenbank ist entweder ein öffentlicher Informationsspeicher für öffentliche Ordner oder ein privater Informationsspeicher (»Postfachspeicher«) für Postfächer. Mehrere Speicher werden zu einer Speichergruppe zusammengefasst, wobei eine Speichergruppe einem Verzeichnis auf der Festplatte entspricht. Jedes Dateiverzeichnis kann nur höchstens eine Exchange-Speichergruppe enthalten.

Überblick

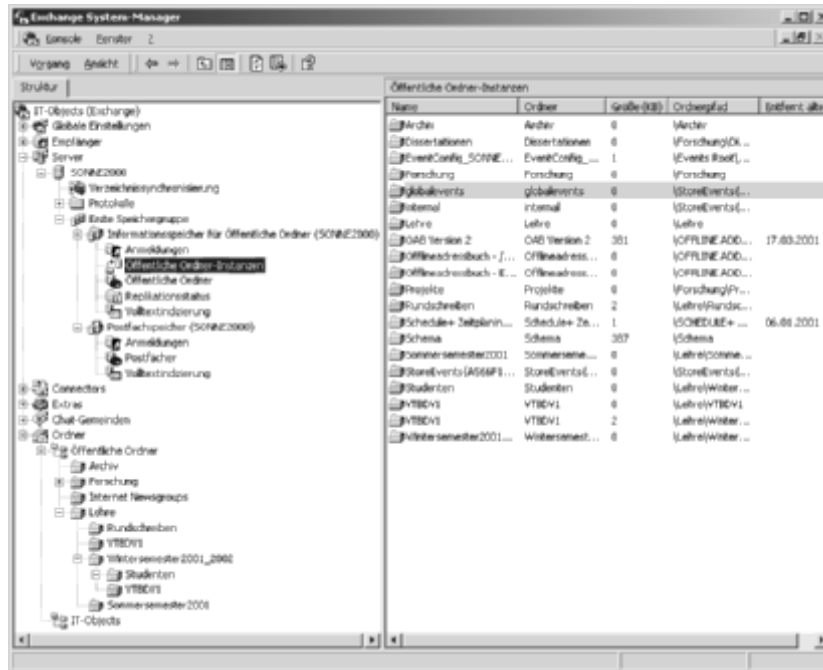


Abbildung 19.5:
Verwaltung der
Datenbanken und
öffentlichen Ordner
im Exchange Server

Im Gegensatz zum Exchange 5.5 kann es mehrere verschiedene Hierarchien von öffentlichen Ordnern geben. Jeder öffentliche Informationsspeicher kann eine Ordnerhierarchie enthalten. MAPI-Clients wie Microsoft Outlook können jedoch nur die Standard-Ordnerhierarchie mit dem Namen ÖFFENTLICHE ORDNER verwalten.

Den Zugriff auf das M:-Laufwerk konfiguriert man über virtuelle Verzeichnisse im MMC-Snap-In »Internet-Informationdienste«. Dabei sollte man die Option »Verzeichnis durchsuchen« anwählen.

Die virtuellen Verzeichnisse können auch vom Exchange System-Manager aus verwaltet werden (unter SERVER/SERVERNAME/PROTOKOLLE/HTTP). Ein virtuelles Verzeichnis ist immer im IIS eingerichtet und kann zusätzlich im Exchange System-Manager eingerichtet werden. Der System-Manager gibt bei der Einrichtung eines Verzeichnisses automatisch die nötigen Daten an den IIS weiter.

Der Unterschied zwischen den nur im IIS eingerichteten virtuellen Verzeichnissen und den zusätzlich in Exchange eingerichteten ist, dass man im ersten Fall die übliche einfache, vom IIS generierte Verzeichnisansicht erhält. Bei den zusätzlich im Exchange System-Manager eingerichteten virtuellen Verzeichnissen wird automatisch beim Zugriff via HTTP-Client die Outlook Web-Access-Oberfläche generiert und an den Client gesendet. Bei der Installation richtet der Exchange Server 2000 im IIS automatisch drei virtuelle Verzeichnisse ein: *Public*, *Exchange* und *ExAdmin*.

IIS-Manager

Virtuelle Verzeichnisse im System-Manager

Komponenten für den Exchange Server 2000

Abbildung 19.6:
Konfiguration des
HTTP-Zugriffs auf
den Webstore über
ein virtuelles
Verzeichnis

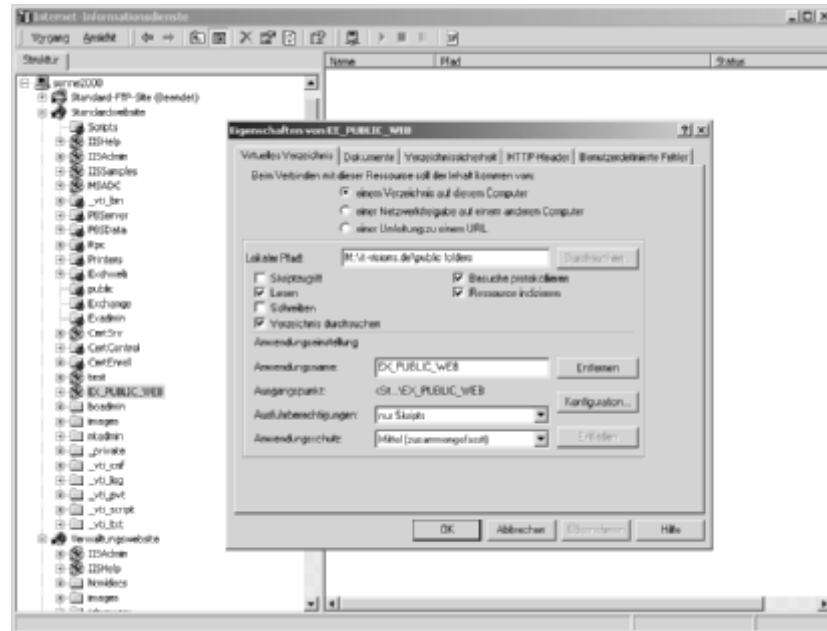
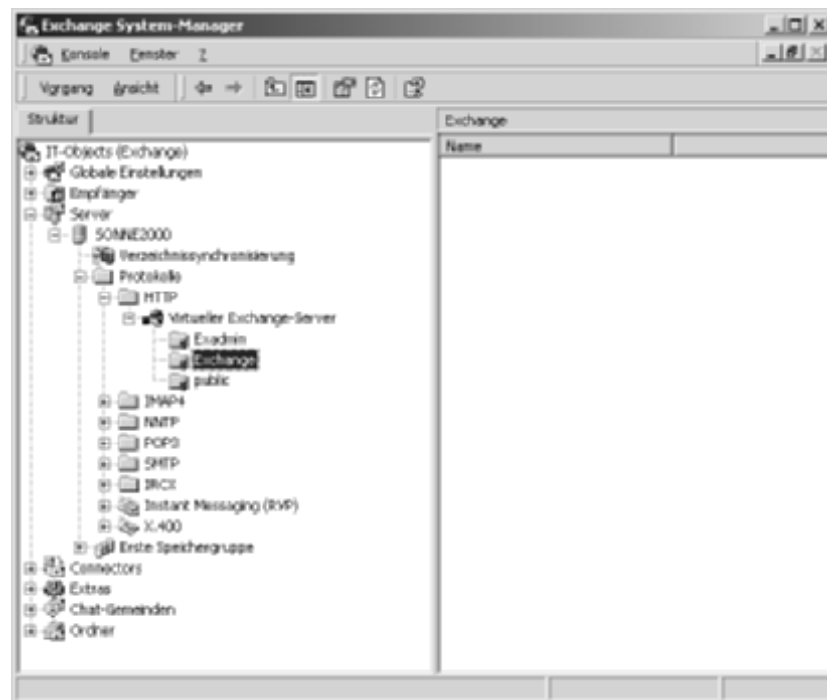


Abbildung 19.7:
Verwaltung der
virtuellen HTTP-
Verzeichnisse im
Exchange System-
Manager



Überblick

Programmierung des Webstores

Programmierer können auf den Webstore folgendermaßen zugreifen:

- ▶ MAPI (ein C++-Interface, siehe Kapitel 18)
- ▶ CDO 1.21 (COM-Automations-Komponente, siehe Kapitel 18)
- ▶ CDO 3.0 (COM-Automations-Komponente, siehe Kapitel 19)
- ▶ OLE DB (C++-Interface, siehe Kapitel 9)
- ▶ ADO (COM-Automations-Komponente, siehe Kapitel 9)

Zugriffsmöglichkeiten mit ADO

Die ActiveX Data Objects (ADO) sind eine COM-Komponente, die auf der Datenbankschnittstelle OLE DB aufbauend den Zugriff auf Datenbanken und andere Datenquellen ermöglicht. Vor Exchange 2000 hat Microsoft seinen Messaging-Server nicht in diese Architektur integriert. Für Exchange 2000 gibt es nun direkt zwei OLE DB-Provider, die es ADO ermöglichen, den Webstore ähnlich wie eine relationale Datenbank anzusprechen.

ADO 2.5

Dies ist einerseits der bereits mit Office 2000 eingeführte »Microsoft Internet Publishing Provider« (ProgID: *MSDAIP.DSO*), mit dem auf dem IIS gemappte Verzeichnisse via HTTP und WebDAV angesprochen werden können. Andererseits liefert Microsoft einen speziellen, performanteren Provider für Exchange 2000 mit Namen *EXOLEDB Provider* (ProgID: *EXOLEDB.DataSource*). Dieser OLE DB-Provider kann nur auf dem Exchange Server selbst verwendet werden. Der Provider ist nicht remoting-fähig. Er kann also nicht verwendet werden, um von Clients aus auf den Exchange Server zuzugreifen.

Collaboration Data Objects 3.0

Mit der Version des Exchange Servers hat sich auch die CDO-Komponente stark verändert. Die für Exchange 2000 entwickelte Version 3.0 der *Collaboration Data Objects* hat kaum noch etwas mit der CDO-Version 1.21 für Exchange 5.5 gemeinsam (Version 2.0 von CDO ist die mit Windows2000 ausgelieferte Light-Version von CDO 3.0, siehe Kapitel 20). CDO 3.0 greift nicht wie CDO 1.2 via MAPI auf Exchange zu, sondern bedient sich der Komponenten *Active Directory Service Interface (ADSI)* für den Zugriff auf den Verzeichnisdienst und ADO für den Zugriff auf den Information Store, der jetzt Webstore heißt.

CDO 3.0

Exchange 2000 unterstützt weiterhin MAPI, so dass auch CDO 1.2 zum Zugriff auf Exchange 2000 verwendet werden kann. Exchange 2000 unterstützt mehrere verschiedene Hierarchien von öffentlichen Ordnern. Über den Exchange System-Manager kann man beliebig viele Hierarchien anlegen. Per MAPI/CDO kann man aber nur auf die Standardhierarchie mit dem Namen »Public Folders« zugreifen. Alle anderen selbst-definierten Ordnerhierarchien bleiben MAPI/CDO verborgen.



19.1.2 Active Directory

Exchange 2000 besitzt keinen eigenen Verzeichnisdienst mehr, sondern benutzt das Windows2000-Active Directory. Die Integration ist insbesondere an folgenden Stellen im Active Directory sichtbar:

Schema- erweiterung	► Die Installation eines Exchange 2000 Servers verändert das Active Directory-Schema durch das Hinzufügen zahlreicher neuer Klassen und etlicher Attribute zu den bestehenden Klassen. So wird die ADS-Klasse <i>user</i> um zahlreiche Attribute erweitert. Ein Teil der Namen dieser Attribute beginnt mit <i>msExch</i> (z.B. <i>msExchMailboxUrl</i> , <i>msExchHideFromAddressLists</i>), andere haben dieses Präfix aber nicht (z.B. <i>homeMTA</i> , <i>otherMailbox</i>).
Empfänger- Klassen	► Die Active Directory-Klassen <i>user</i> , <i>contact</i> und <i>group</i> bilden die Grundlage für Postfächer, benutzerdefinierte Empfänger und Verteilerlisten. Die Administration dieser Objekte erfolgt über das MMC-Snap-In »Active Directory-Benutzer und -Computer«, das durch die Exchange 2000-Installation um zusätzliche Funktionen (Registerkarten und Kontextmenüeinträge) erweitert wird. Die Exchange-Empfänger sind im <i>DefaultNamingContext</i> als Erweiterung der Active Directory-Benutzer-Klassen abgelegt: ► Ein Postfach ist die Erweiterung der User-Klasse. ► Ein Verteiler ist die Erweiterung der Group-Klasse. ► Ein benutzerdefinierter Empfänger ist die Erweiterung der Contact-Klasse.
Konfigurations- daten	► Alle Exchange-Konfigurationsinformationen (z.B. Speichergruppen und Speicher) sind im Configuration Naming Context des Active Directory gespeichert. Diese Konfigurationsinformationen liegen im Untercontainer <i>Services/ Microsoft Exchange</i> (z.B. <i>LDAP://CN=Microsoft Exchange, CN=Services, CN=Configuration, DC=IT-Visions, DC=de</i>).
Öffentliche Ordner	► Für alle öffentlichen Ordner gibt es einen Eintrag im Active Directory, unterhalb des Containers <i>CN=Microsoft Exchange System Objects</i> , der sich direkt unterhalb der Wurzel des Default Naming Context des Active Directory befindet (z.B. <i>LDAP://sonne2000/CN=Microsoft Exchange System Objects, DC=IT-Visions, DC=de</i>).

Zugriff auf das Verzeichnis via ADSI

ADS und ADSI	Mit der Integration des Exchange-Verzeichnisdienstes in das Windows2000 Active Directory ist die Rolle des Active Directory Service Interface (ADSI) zur Verwaltung des Verzeichnisdienstes gestärkt. Sie greifen also in ADSI mit dem ADS-Namespace auf die Exchange Server-Konfiguration zu.

19.1.3 Weitere Komponenten

CDOEXM	Um die Verwaltung des Exchange Servers noch weiterhin zu vereinfachen, gibt es eine Komponente mit dem Namen <i>CDO for Exchange Management</i> (CDOEXM). CDOEXM setzt auf ADSI auf.

Überblick

Für den Bereich der Workflow-Funktionen bietet Exchange 2000 die Komponente *Microsoft CDO Workflow Objects for Microsoft Exchange (CDOWF)*, die hier aber nicht beschrieben ist. Die Implementierung finden Sie in CDOWF.DLL.

CDOWF

Für die Entwicklung von Ereignisbehandlungsroutinen für die neuen synchronen und asynchronen Ereignisse gibt es die Komponente EXOLEDB Library (EXOLEDB.DLL). Die EXOLEDB Library bietet auch vier Methoden zur Umwandlung von Webstore-URLs.

Events

Außerdem liefert Microsoft beim Exchange Server 2000 drei WMI-Provider mit: *ExchangeQueueProvider*, *ExchangeRoutingTableProvider* und *ExchangeClusterProvider*, die den Namespace `\ROOT\CIMV2\APPLICATIONS\EXCHANGE` bilden. Diese Provider stellen fünf WMI Managed Objects bereit: *ExchangeLink*, *ExchangeQueue*, *ExchangeConnectorState*, *ExchangeServerState* und *ExchangeClusterResource*. Für die grundsätzliche Vorgehensweise bei der Arbeit mit diesen WMI-Klassen schauen Sie bitte in das Kapitel 14.

WMI

19.1.4 Grundlegende Konzepte des Webstores

In diesem Kapitel werden zunächst einige grundlegende Konzepte des Exchange 2000 Webstores erläutert, soweit diese für die Automatisierungskomponenten relevant sind.

Felder in der Webstore-Datenbank

Der Webstore ist ebenso wie der Exchange 5.5 Informationstore eine semi-strukturierte Datenbank. Zu den verschiedenen Typen von Einträgen wird eine Vielzahl vordefinierter Attribute bereitgestellt. Die Standardattribute eines Eintrags sind im Webstore-Schema festgelegt. Für Nachrichten sind auch benutzerdefinierte Attribute möglich.

Attribute

Attribut-Identifikation über Pfade

Ein Attribut wird nicht mehr wie in MAPI über ein Property-Tag (eine Kolonne aus Zahlen und Buchstaben), sondern über einen aussagekräftigen, hierarchischen, global eindeutigen Textstring identifiziert. Der Pfad setzt sich zusammen aus einem Attributnamen und einem Namespace, in dem dieser Attributname definiert ist. Der Namespace ist ein Uniform Resource Identifier (URI), entweder in der Form eines Uniform Resource Locators (URL) oder eines Uniform Resource Names (URN). Dies ähnelt der Benennung von Elementen und Attributen in XML (vgl. Anhang A). Im Exchange Server werden auch Klassennamen und Konstanten auf diese Weise gebildet.

Benennung

URI ist der Oberbegriff für einen Textstring, der zur Identifikation von Ressourcen verwendet wird. Ein *URL* enthält im Gegensatz zu einem *URN* eine Protokollangabe.

Beispiel für einen URL: `http://www.it-visions.de`

Beispiel für einen URN: `urn:schemas:mailheader:sender`

URI, URN und URL

Namespaces Das Webstore-Schema umfasst Attribute aus mehreren Namespaces. Die verwendeten Namespaces lassen sich gemäß dem ersten Teil der Pfadangabe in drei Gruppen einteilen:

- ▶ **DAV**
Attribute aus dem WebDAV-Standard (*Web-based Distributed Authoring and Versioning*)
z.B. DAV:creationdate
- ▶ **urn**
auf anderen Internet-Standards basierende Attribute
z.B. urn:schemas:mailheader:sender
- ▶ **http://schemas.microsoft.com/exchange**
Microsoft Exchange-spezifische Attribute
z.B. http://schemas.microsoft.com/exchange/ntsecuritydescriptor

Tabelle 19.2:
Ausgewählte Web-
Dav-Attributna-
men aus der Content
Class urn:content-
classes:item

Attributname	Erläuterung
DAV:isfolder	Zeigt an, ob der Eintrag ein Ordner ist
DAV:href	URL des Eintrags
DAV:displayName	Name, mit dem der Eintrag angezeigt wird
DAV:parentname	URL des übergeordneten Eintrags
DAV:childCount	Anzahl der Unterobjekte
DAV:objectCount	Anzahl der Unterobjekte, die nicht Ordner sind
DAV:creationdate	Datum und Uhrzeit der Erzeugung
DAV:getlastmodified	Datum und Uhrzeit der letzten Veränderung
DAV:lastaccessed	Datum und Uhrzeit des letzten Zugriffs



Der Zugriff auf ein nicht vorhandenes Attribut führt nicht zu einem Fehler. Exchange liefert dafür einen Leerstring zurück.

Content Classes

**Klassen für
Exchange-Objekte**

Elemente im Exchange-Webstore sind Instanzen von Klassen. Diese Klassen werden *Content Classes* genannt. Jedes Attribut gehört zu einer so genannten Content Class. Sofern ein Attribut zu mehreren Content Classes gehören kann, wird dies durch Vererbung innerhalb der Content Classes erreicht.

Oberste Klasse der Vererbungshierarchie ist die Content Class urn:content-classes:object. Davon erbt urn:content-classes:item. Diese Klasse ist wiederum Oberklasse für urn:content-classes:folder und urn:content-classes:message.

Überblick

Typ	Content Class	ProgID
Beliebiges Element	urn:content-classes:item	CD0.Item
Ordner	urn:content-classes:folder	CD0.Folder
Nachricht	urn:content-classes:message	CD0.Message
Termin	urn:content-classes:appointment	CD0.Appointment
Dokument	urn:content-classes:document	CD0.Item
Notiz	urn:content-classes:note	CD0.Item
Aufgabe	urn:content-classes:task	CD0.Item
Kontakt	urn:content-classes:person	CD0.Person
Posteingang, Entwürfe	urn:content-classes:mailfolder	CD0.Folder
Kalender-Ordner	urn:content-classes:calendarfolder	CD0.Folder
Journal-Ordner	urn:content-classes:journalfolder	CD0.Folder
Kontakte-Ordner	urn:content-classes:contactfolder	CD0.Folder
Notizen-Ordner	urn:content-classes:notefolder	CD0.Folder

Tabelle 19.3:
Ausgewählte
Content Classes

Objektidentifikation via URL

In Exchange 2000 können Ordner und auch einzelne Nachrichten über URLs (Moniker) angesprochen werden. Als Moniker-Typ (Protokoll) wird interessanterweise FILE:// (File-URL) oder HTTP:// (HTTP-URL) benutzt.

Exchange-Moniker

File-URLs können nur auf dem Server verwendet werden. Sie beginnen immer mit dem feststehenden Ausdruck FILE://./BACKOFFICESTORAGE/. Danach folgt der Pfad genau so, wie er auch im M:-Laufwerk zu sehen ist.

File://

HTTP-URLs basieren auf virtuellen Verzeichnissen, die im Internet Information Server (IIS) bei der Installation von Exchange 2000 in der Standardwebsite eingerichtet werden. Im Standard werden die in der folgenden Tabelle genannten virtuellen Verzeichnisse eingerichtet. Sie können eigene virtuelle Verzeichnisse hinzufügen und damit andere URLs ermöglichen.

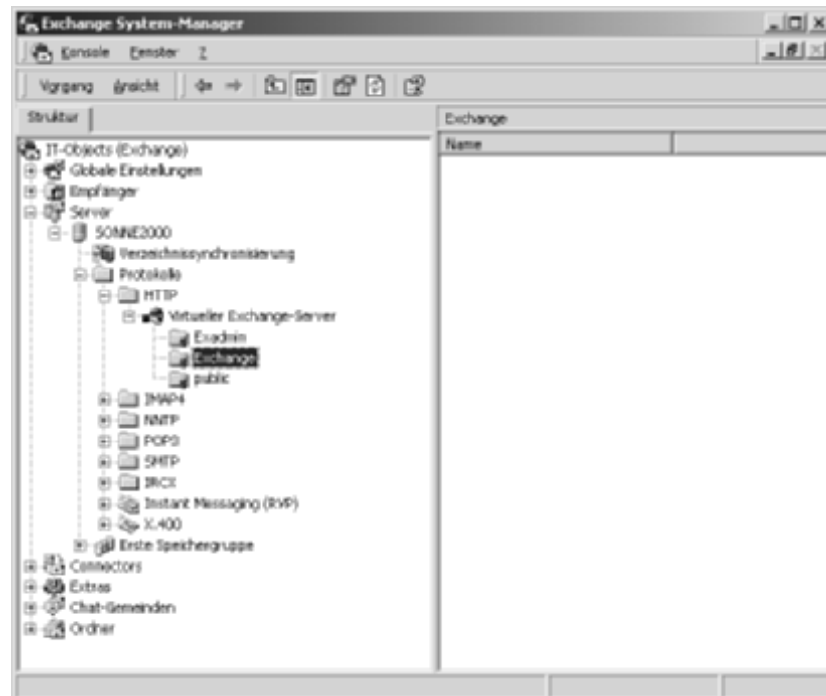
HTTP://

Ort	Virtuelles Verzeichnis	Pfad
Öffentlicher Informationsspeicher	Standardwebsite/Public	M:\DOMAIN\ PUBLIC FOLDERS
Private Informationsspeicher	Standardwebsite/Exchange	M:\IT-VISIONS.DE\MBX
Wurzel des Exchange-Speichers	Standardwebsite/ExAdmin	\\.\BACKOFFICESTORAGE

Tabelle 19.4:
Die virtuellen Standardverzeichnisse für den Exchange Webstore im IIS

Komponenten für den Exchange Server 2000

Abbildung 19.8:
Ansicht der vorkon-
figurierten virtuel-
len Verzeichnisse für
den Webstore im
Exchange 2000
System-Manager



Die folgende Tabelle zeigt Beispiele für File-URLs mit äquivalenten HTTP-URLs.

Tabelle 19.5:
Beispiele für
Webstore-Moniker

Typ	Content Class	File-URL	HTTP-URL
Öffentli- cher Ord- ner	urn: contentclas ses: folder	FILE:///BACKOFFICESTORAGE/ SONNE2000. IT-VISIONS.DE/ PUBLIC FOLDERS/ LEHRE/ SOMMERSEMESTER2001/	HTTP:// SONNE2000. IT-VISIONS.DE/ PUBLIC/ LEHRE/ SOMMERSEMESTER2001/
Nachricht	urn: contentclas ses: message	FILE:///BACKOFFICESTORAGE/ SONNE2000. IT-VISIONS.DE/ PUBLIC FOLDERS/ LEHRE/ SOMMERSEMESTER2001/ CURRICULUM.EML	HTTP:// SONNE2000. IT-VISIONS.DE/ PUBLIC/ LEHRE/ SOMMERSEMESTER2001/ CURRICULUM.EML

Überblick

Typ	Content Class	File-URL	HTTP-URL
Ordner in einem Postfach	urn: contentclas ses: folder	FILE:///BACKOFFICESTORAGE/ SONNE2000. IT-VISIONS.DE / MBX/HS/ POSTEINGANG	HTTP:// SONNE2000. IT-VISIONS.DE/ EXCHANGE/ HS/ POSTEINGANG/

Tabelle 19.9:
Beispiele für
Webstore-Moniker
(Forts.)

Bitte beachten Sie, dass bei einem deutschen Server in den Pfaden zum Teil englische, zum Teil deutsche Begriffe zu verwenden sind. So muss die Standardhierarchie der öffentlichen Ordner über /PUBLIC FOLDERS angesprochen werden; der Posteingang innerhalb eines Postfachs dagegen über /POSTEINGANG, nicht /INBOX. Outlook 2000 zeigt durchweg deutsche Namen an. Sie ermitteln den zu verwendenden Namen am besten, indem Sie den Exchange Webstore über das Dateisystem betrachten (siehe folgende Abbildung). Die dort zu findenden Namen sind auch in den URLs zu verwenden.

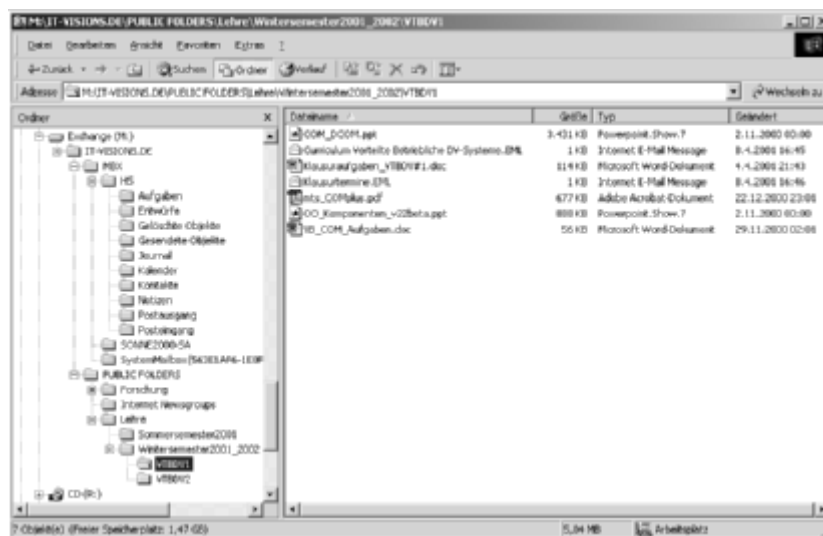


Abbildung 19.9:
Die Ansicht des
Webstores über ein
Laufwerk im Datei-
system zeigt die
Namen, die in den
Webstore-File-URLs
zu verwenden sind.

Umwandlung von URLs

Die Komponente *EXOLEDB Library* bietet eine Klasse *EXOLEDBUrlMapper* (ProgID: *EXOLEDB.UrlMapper*) mit vier Methoden zur Umwandlung von HTTP-URLs in File-URLs:

- ▶ `HttpUrlToFilePath(url)`
- ▶ `FilePathToEXOLEDBFileUrl(FilePath)`
- ▶ `EXOLEDBFileUrlToFilePath(FileURL)`
- ▶ `FilePathToHttpUrls(FilePath)`

UrlMapper

Diese Methoden werden durch das folgende Beispiel erläutert.

```
Dim mapper ' As New EXOLEDB.EXOLEDBUrlMapper
Dim FilePath, FileURL, HttpURL, url
Dim n

Set mapper = CreateObject("EXOLEDB.UrlMapper")
url = "http://192.168.0.1:82/public"

FilePath = mapper.HttpUrlToFilePath(url)
say FilePath
FileURL = mapper.FilePathToEXOLEDBFileUrl(FilePath)
say FileURL
FilePath = mapper.EXOLEDBFileUrlToFilePath(FileURL)
say FilePath

' --- Alle möglichen HTTP-URLs
HttpURL = mapper.FilePathToHttpUrls(FilePath)
n = 0
For Each url In HttpURL
    n = n + 1
    say n & ":" & url
Next
```

*Listing 19.1: Umwandlung zwischen Datei-URLs und HTTP-URLs mit der Klasse
»EXOLEDB.UrlMapper« [EX2000URLUmwandlung.wsfl]*

Das obige Listing erzeugt in der Testumgebung folgende Ausgabe:

```
HTTP-URL: http://192.168.0.1:82/public
Dateipfad: \\.\BackOfficeStorage\IT-Visions.de\Public Folders
Datei-URL (Pfad inkl. Moniker): file:///BackOfficeStorage/IT-
Visions.de/Public Folders
zurückgewandelter Dateipfad: \\.\BackOfficeStorage\IT-Visions.de\Public
Folders
```

```
Alle möglichen HTTP-URLs für diesen FilePath:
1: http://Sonne2000.IT-Visions.de:82/public
2: http://Sonne2000.IT-Visions.de:82/ex_public_web
3: https://Sonne2000.IT-Visions.de/exadmin/IT-Visions.de/
Public%20Folders
4: http://Sonne2000.IT-Visions.de:82/exadmin/IT-Visions.de/
Public%20Folders
5: https://Sonne2000.IT-Visions.de/public
6: https://Sonne2000.IT-Visions.de/ex_public_web
```


Jeder URL wird sowohl für ungesichertes HTTP als auch für HTTP over Secure Socket Layer (Moniker https://) angezeigt. Zu beachten ist, dass die Methode `FilePathToHttpUrls(FilePath)` ein Array of Strings liefert, in dem alle möglichen URLs zu diesem Webstore-Pfad enthalten sind. Dazu durchsucht die Methode die IIS-Metabase nach allen virtuellen Verzeichnissen, die auf den Webstore verweisen. Daher sind in der obigen Liste nicht nur solche virtuellen Verzeichnisse dargestellt, die nicht im Standard angelegt wurden (*public*), sondern auch zusätzlich angelegte Verzeichnisse (*ex_public_web*).

Verschiedene URLs

19.2 Exchange-Programmierung mit ADO

In Exchange 2000 kann via *ActiveX Data Objects (ADO)* sowohl lesend als auch schreibend auf im Exchange Server-Webstore gespeicherte Daten zugegriffen werden. Mehr über ADO und die zu Grunde liegende OLE DB-Architektur erfahren Sie in Kapitel 9.

19.2.1 OLE DB-Provider für den Webstore

ADO basiert auf der universellen Datenzugriffsschnittstelle OLE DB. OLE DB greift mit OLE DB-Providern auf Datenquellen zu. Microsoft liefert zwei OLE DB-Provider zum Zugriff auf den Exchange 2000-Webstore:

- ▶ den *OLE DB Provider for Internet Publishing (MSDAIPP)*, auch *Microsoft Internet Publishing Provider* genannt. MSDAIPP greift über im IIS konfigurierte virtuelle Verzeichnisse via HTTP und Web-based Distributed Authoring and Versioning (WebDav) auf den Exchange Server zu.
- ▶ den *EXOLEDB Provider*, auch *Webstore OLE DB Provider* genannt. EXOLEDB benutzt direkt das Exchange Server-API.
- ▶ Der grundsätzliche Unterschied ist, dass der EXOLEDB-Provider nur auf dem Server selbst verwendet werden kann. Der Provider ist nicht remote-fähig. Er kann also nicht verwendet werden, um von Clients aus auf den Exchange Server zuzugreifen. Außerdem unterstützt der Provider keine Impersonifizierung. Er ist aber performanter als MSDAIPP.
- ▶ File-URLs können nur mit dem EXOLEDB-Provider verwendet werden. HTTP-URLs können auf dem Server mit beiden Providern verwendet werden. Standard ist aber MSDAIPP. Soll bei einem HTTP-URL auf dem Server der EXOLEDB-Provider eingesetzt werden, muss dieser explizit spezifiziert werden.

MSDAIPP

EXOLEDB

Verwendung der URLs

Die nachstehende Tabelle zeigt den Unterschied zwischen beiden Providern.

Komponenten für den Exchange Server 2000

Abbildung 19.10:
Die OLE DB-Provi-
der für Exchange
2000

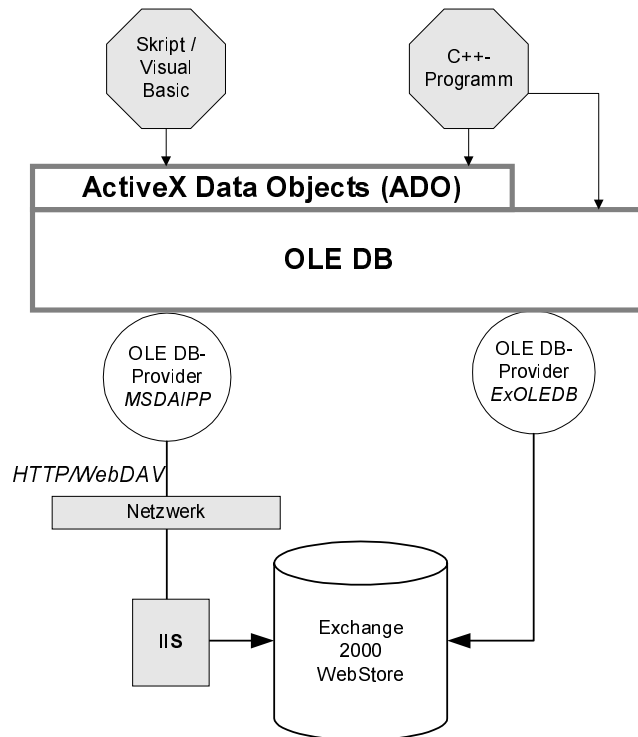


Tabelle 19.6:
Unterschiede zwi-
schen den beiden
OLE DB-Providern

Pro- vider	Kürzel	ProgID	Datei	Moniker	Verwen- dung auf Server	Verwen- dung auf Client
Micro- soft Inter- net Publi- shing Provi- der	MSDAIPP	MSDAIPP. DSO	MSDAIPP. DLL	HTTP://	Lese- und Schreib- zugriff	Lese- und Schreib- zugriff
Exchan- ge OLE DB Pro- vider	EXOLEDB	EXO- LEDB.Dat aSource	EXO- LEDB.DLL	HTTP:// und File://	Lese- und Schreib- zugriff	nicht möglich

ADO und Exchange 5.5

Es gab in der Vergangenheit auch einen OLE DB-Provider für Exchange 5.5 von der Firma Merant [MER00]. Merant hat jedoch auf Grund der geringen Nachfrage den Vertrieb des Providers eingestellt.

19.2.2 ADO-Klassen

Es wird ADO ab Version 2.5 benötigt. Aus dem ADO-Objektmodell werden für Exchange 2000 fünf Klassen verwendet:

- ▶ Record
- ▶ Recordset
- ▶ Connection
- ▶ Field
- ▶ Stream

Record und Stream wurden mit ADO 2.5 neu eingeführt.

Die Klasse `ADODB.Recordset` wird zum Zugriff auf Objektmengen verwendet. Ein Recordset-Objekt ist das Ergebnis einer SQL-Abfrage auf dem Exchange 2000-Webstore. Die Klasse `ADODB.Record` dient dagegen dem Zugriff auf einzelne Objekte. Dabei wird eine Instanz dieser Klasse über einen Moniker an einen Eintrag im Exchange Server gebunden. Es ist aber nicht so, dass ein Recordset-Objekt eine Menge von Record-Objekten darstellt.

Record versus Recordset

Beide Klassen besitzen eine `Fields`-Collection mit einzelnen `Field`-Objekten. Diese `Field`-Objekte repräsentieren die Attribute des Webstore-Elements.

Field

Ein `Connection`-Objekt dient dazu, über mehrere einzelne Objektzugriffe hinweg eine Verbindung aufzubauen. Ein `Stream` ist die serialisierte Form eines Objekts, die in dieser Form persistent im Dateisystem abgelegt werden kann.

Connection und Stream

Eine genauere Beschreibung dieser Klassen finden Sie im Kapitel 9.

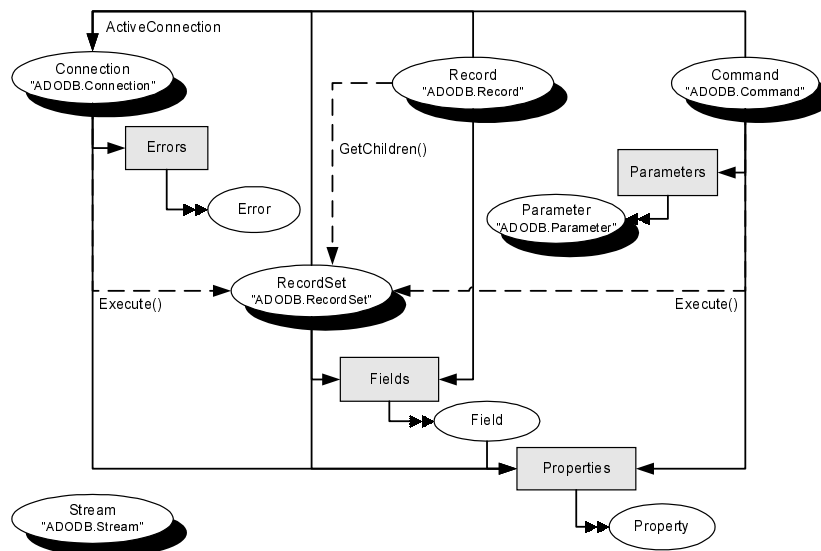


Abbildung 19.11:
ADO 2.6-Objektmodell



Der OLE DB-Provider kann auch Schreiboperationen ausführen, allerdings nur über die Methoden der Klasse `ADODB.Record` und `ADODB.Recordset`. Das Absenden von SQL-Befehlen wie `DELETE`, `UPDATE` und `INSERT INTO` via `ADODB.Command` ist nicht möglich!

Select-Syntax

19.2.3 SQL-Abfragen auf den Webstore

Die OLE DB-Provider unterstützen Abfragen in einem SQL-ähnlichen Stil (Webstore SQL). In Zusammenhang mit dem `SELECT`-Befehl werden auch die Schlüsselwörter `WHERE`, `ORDER BY`, `LIKE` und `CONTAINS` unterstützt.

```
SELECT * | Attributliste
FROM URL
[WHERE Bedingung]
[order-by-Klausel]
```

Tabellenname

Die Stelle des Tabellennamens nimmt ein File- oder HTTP-URL ein. Alle Attributnamen und der URL müssen in doppelten Anführungszeichen stehen.

Sternoperator

Der Sternoperator ist anstelle einer Attributliste möglich, liefert jedoch nur die Standardattribute und keine benutzerdefinierten Attribute zurück.

Scope

```
SELECT * | Attributliste
FROM SCOPE('SHALLOW|DEEP TRAVERSAL OF URL')
[WHERE Bedingung]
[ORDER BY Attributliste]
```

An die Stelle des URLs kann auch eine Gültigkeitsbereichsangabe rücken. Diese besteht aus dem Schlüsselwort `SCOPE` gefolgt von einem String in Klammern und in einfachen Anführungszeichen. Der Gültigkeitsbereich umfasst einen Startordner und die Angabe, ob die Suche sich nur über eine Ebene (*Shallow Traversal*) oder auch über alle Unterebenen (*Deep Traversal*) erstrecken soll. Ohne Angabe ist der Standard *Shallow Traversal*.



Deep Traversal wird für die MAPI-kompatible Standardordnerhierarchie `ÖFFENTLICHE ORDNER` nicht unterstützt. Hier bleibt nur die Möglichkeit des rekursiven Abstiegs (vgl. Kapitel 18). Postfächer und selbstangelegte öffentliche Ordnerhierarchien unterstützen allerdings sehr wohl Deep Traversal.

Berechnete Attribute

Berechnete Attribute

Eine Einschränkung führt häufig zu Fehlern: Einige Attribute können nicht in Bedingung oder Sortierung einbezogen werden, da sie im Webstore nicht gespeichert sind, sondern berechnet werden.

Exchange-Programmierung mit ADO

DAV:getetag
DAV:href
DAV:lockdiscovery
DAV:parentname
DAV:resourcetype
DAV:searchrequest
http://schemas.microsoft.com/exchange/content-href
http://schemas.microsoft.com/exchange/ntsecuritydescriptor
http://schemas.microsoft.com/exchange/oof-state
http://schemas.microsoft.com/exchange/publicfolderemailaddress
http://schemas.microsoft.com/repl/repl-uid
http://schemas.microsoft.com/repl/resourcetag
urn:schemas:contacts:proxyaddresses
urn:schemas:httpmail:htmldescription
urn:schemas:httpmail:subject
urn:schemas:mailheader:subject
urn:schemas-microsoft-com:exch-data:baseschema
urn:schemas-microsoft-com:exch-data:schema-collection-ref

Listing 19.2: Liste der berechneten Attribute in Exchange 2000

Standardattribute

ADO liefert bei einer Anfrage immer mehr als die explizit angeforderten Attribute zurück. Auf jeden Fall werden immer die nachstehend genannten Attribute zurückgegeben. Dabei sind diese Attributnamen im Unterschied zu den explizit angeforderten Attributnamen groß geschrieben.

Standardattribute

RESOURCE_PARSENAME
RESOURCE_PARENTNAME
RESOURCE_ABSOLUTEPARSENAME
RESOURCE_ISHIDDEN
RESOURCE_ISREADONLY
RESOURCE_CONTENTTYPE
RESOURCE_CONTENTCLASS
RESOURCE_CONTENTLANGUAGE
RESOURCE_CREATIONTIME
RESOURCE_LASTACCESSTIME
RESOURCE_LASTWRITETIME
RESOURCE_STREAMSIZE
RESOURCE_ISCOLLECTION
RESOURCE_ISSTRUCTUREDDOCUMENT
DEFAULT_DOCUMENT
RESOURCE_DISPLAYNAME
RESOURCE_ISROOT
RESOURCE_ISMARKEDFOROFFLINE

Webstore-SQL-Beispiele

order by Das erste Beispiel ist eine einfache Abfrage der Unterelemente eines Ordners aus der Standardhierarchie ÖFFENTLICHE ORDNER. Gefragt wird nach der Content Class und dem Namen des Elements, wobei die Ausgabe nach letzterem sortiert wird.

```
SELECT "urn:schemas:mailheader:content-class" ,"DAV:displayname"  
FROM "http://sonne2000.IT-Visions.de/public/Lehre"  
ORDER BY "DAV:displayname"
```

Listing 19.3: Beispiel für eine SQL-Anfrage an den Exchange 2000-Webstore

Shallow Traversal Die folgende Abfrage ist äquivalent, da *Shallow Traversal* der Standard ist.

```
SELECT "DAV:href", "DAV:displayname"  
FROM scope('shallow traversal of " file:///./BackOfficeStorage/  
Sonne2000.IT-Visions.de/  
public folders"')
```

Listing 19.4: Beispiel für eine SQL-Anfrage an den Exchange 2000-Webstore

where Das dritte Beispiel schließt zusätzlich eine Bedingung ein: Nur nicht-versteckte Ordner werden ausgegeben.

```
SELECT "DAV:href", "DAV:displayname"  
FROM scope('shallow traversal of " file:///./BackOfficeStorage/  
Sonne2000.IT-Visions.de/  
public folders"')
```

Listing 19.5: Beispiel für eine SQL-Anfrage an den Exchange 2000-Webstore

Deep Traversal Deep Traversal ist nur über selbstdefinierte öffentliche Ordnerhierarchien möglich. Die Anfrage richtet sich an die Ordnerhierarchie »Standort Essen«. Im HTTP-Pfad kann dazu nicht das virtuelle Verzeichnis Public verwendet werden. Das virtuelle Verzeichnis EXADMIN entspricht der Wurzel des M:-Laufwerks.

```
SELECT "urn:schemas:mailheader:content-class",  
"DAV:href","DAV:displayname"  
FROM scope ('deep traversal of  
"http://sonne2000.IT-Visions.de/exadmin/IT-Objects.de/  
Standort Essen"')
```

Listing 19.6: Beispiel für eine SQL-Anfrage an den Exchange 2000-Webstore

19.2.4 Hilfsroutinen für die ADO-Exchange-Programmierung

Dieses Kapitel enthält einige nützliche Hilfsroutinen, die in den Programmbeispielen in den folgenden Kapiteln verwendet werden.

Hilfsroutinen zum Öffnen der Objekte

Recordset versus Record

Die beiden folgenden Hilfsroutinen dienen dazu, ein Record- bzw. ein Recordset-Objekt auf Basis eines übergebenen Pfads bzw. einer übergebenen Suchanfrage zurückzugeben.

```
' ### Liefert einen Zeiger auf ein einzelnes Objekt
Function get_ExRec(url)
Dim rec ' As ADODB.Record
say "Öffnen des Ordners: " & url

' --- Instanziierung der Klasse
Set rec = CreateObject("ADODB.Record")

' --- Öffnen des Records zum Lesen und Schreiben
Const adModeReadWrite = 3
rec.Open url, , adModeReadWrite
say "Ordner geöffnet: " & rec.ActiveConnection
Set get_ExRec = rec
End Function
```

Listing 19.7: Hilfsroutine zum Öffnen eines einzelnen Exchange-Objekts

```
' ### Liefert einen Zeiger auf eine Objektmenge
Function get_ExRS(url, query)
Dim rs ' As ADODB.Recordset
Dim objRec ' As ADODB.Record

' --- Instanziierung der Klasse
Set rs = CreateObject("ADODB.Recordset")
' --- Erst Ordner öffnen
Set objRec = get_ExRec(url)

' --- Mit dieser Verbindung Query ausführen
rs.Open query, objRec.ActiveConnection
Set get_ExRS = rs
End Function
```

Listing 19.8: Hilfsroutine zum Öffnen einer Exchange-Objektmenge

Vorheriger Aufbau einer Verbindung

Wie immer bei der Arbeit mit ADO besteht auch die Möglichkeit, vorher eine Verbindung aufzubauen und diese dann wiederzuverwenden. Die

**ADODB.
Connection**

nachfolgende Hilfsroutine `ADOEX2K_Connect()` verwendet den MSDAIPP-Provider. Für die Verwendung von EXOLEDB auf dem Server muss nur eine Zeile geändert werden. Der Connectionstring besteht aus dem URL eines Wurzelordners.



Im Folgenden kann mit dieser Verbindung nur auf Ordner zugegriffen werden, die unterhalb des angegebenen Wurzelordners liegen. Es bietet sich daher an, einen URL auf einen Top-Level-Ordner eines öffentlichen oder privaten Informationsspeichers zu übergeben.

```
' ### Verbindungsaufbau
Function ADOEX2K_Connect(rooturl)
Dim objCon ' As ADODB.Connection

say "Verbindung zu: " & rooturl
Set objCon = CreateObject("ADODB.Connection")
objCon.Provider = "MSDAIPP.DSO"

' Bei Anwendung auf dem Server alternativ:
' objCon.Provider = "EXOLEDB.DataSource"

objCon.ConnectionString = rooturl
objCon.Open

Set ADOEX2K_Connect = objCon
End Function
```

Listing 19.9: Hilfsroutine zum expliziten Verbindungsaufbau

Hilfsroutinen zur Ausgabe

list_table() Zum Einsatz kommen in diesem Kapitel auch zwei bereits aus dem ADO-Kapitel bekannte Hilfsroutinen, um alle Felder eines Recordset- bzw. Record-Objekts auszugeben und um einen Recordset komplett auszugeben.

```
' ### Hilfsroutine: Universelle Tabellenausgabe
Sub list_table(rs)
Dim a ' Ausgabe

' --- Ausgabe der Feldnamen
a = ""
For Each f In rs.fields
' --- Zusammensetzen der Ausgabe
If a <> "" Then a = a & ";"
a = a & f.Name
Next
say a

' --- Iteration über alle Datensätze
Do While Not rs.EOF
```


Exchange-Programmierung mit ADO

```
a = ""

' --- Iteration über alle Felder
For Each f In rs.fields

    ' --- Zusammensetzen der Ausgabe
    If a <> "" Then a = a & ";"
    If Not IsNull(rs(f.Name)) Then
        a = a & CStr(rs(f.Name))
    End If
Next

' --- Ausgabe
say a
' --- nächster Datensatz
rs.MoveNext
Loop
End Sub
```

Listing 19.10: Hilfsroutine

Eine nützliche Hilfsroutine zur Erforschung von Datenquellen mit ADO ist die Ausgabe aller Felder, die ein übergebenes Record- bzw. Recordset-Objekt kennt. Die Routine kann aber auch für andere Objekttypen eingesetzt werden, die eine Fields-Collection unterstützen. Dies ist möglich durch den COM-Polymorphismus auf Basis des späten Bindens. **list_fields()**

```
' ### Ausgabe aller Felder eines (Record-)Objekts
Sub list_fields(obj)
Dim oF ' As ADODB.Field

say "Anzahl der Eigenschaften: " & obj.fields.Count

For Each oF In obj.fields
    If IsNull(oF.Value) Then
        say oF.Name & "= NULL"
    Else
        say oF.Name & "=" & flat(oF.Value)
    End If
Next

End Sub
```

Listing 19.11: Hilfsroutine

19.2.5 Zugriff auf Ordner

Mit ADO kann man sowohl auf öffentliche als auch auf private Ordner (Ordner in Postfächern) zugreifen.

Ordnerattribute auslesen

Ordnerattribute Der Zugriff auf einen Ordner beginnt mit der Instanziierung von ADODB.Record. Die Methode Open() bindet diese Instanz mit Hilfe eines Monitors an einen Ordner im Exchange Webstore. Danach kann über die Fields-Collection auf die Eigenschaften zugegriffen werden.

```
' === Beispiel für den Zugriff auf einen Ordner
Sub ADOEX2K_OrdnerAttributeLesen()
Dim rec As ADODB.Record
Dim url As String

' --- Verbindungsaufbau
' Angaben zum Ordner
Const DOMAIN = "sonne2000.IT-Visions.de"
Const WURZEL = "public"
Const ORDNER = "Lehre/Wintersemester2001_2002/VTBDV1"
' URL zusammenbauen
url = "http://" & DOMAIN & "/" & WURZEL & "/" & ORDNER
say "Zugriff auf Ordner: " & url
' Instanziierung der Klasse
Set rec = CreateObject("ADODB.Record")
' Öffnen des Records zum Lesen und Schreiben
Const adModeReadWrite = 3
rec.Open url, , adModeReadWrite
say "Ordner geöffnet!"

' --- Zugriff auf einzelne Attribute des Ordners
say "Ordnername: " & rec.fields("DAV:Displayname")
say "URL des Ordners: " & rec.fields("DAV:href")
say "Klasse: " & rec.fields("DAV:contentclass")
say "Übergeordneter Ordner: " & rec.fields("DAV:parentname")
say "Kommentar: " & rec.fields("DAV:Comment")
say "Ordnergröße: " & rec.fields("http://schemas.microsoft.com/
exchange/foldersize")

' --- Alternativ: Alle Attribute ausgeben via
' ADO-Hilfsroutine
list_fields rec
End Sub
```

Listing 19.12: Beispiel für den Zugriff auf einen Ordner
[ADOEX2K_OrdnerAttributeLesen.wsf]

Wenn man vorher eine Verbindung mit dem ADODB.Connection-Objekt aufbaut, verändert sich der Code nur leicht, wie das folgende Skript zeigt.

```
' === Beispiel für den Zugriff auf einen Ordner unter Verwendung einer
Connection
Sub ADOEX2K_OrdnerAttributeLesen2()
```

Exchange-Programmierung mit ADO

```
Dim rec ' As ADODB.Record
Dim con ' As ADODB.Connection
Dim url ' As String

' --- Verbindungsaufbau
' Angaben zum Ordner
Const DOMAIN = "sonne2000.IT-Visions.de"
Const WURZEL = "public"
Const ORDNER = "Lehre/Wintersemester2001_2002/VTBDV1"
' URL zusammenbauen
url = "http://" & DOMAIN & "/" & WURZEL & "/" & ORDNER
say "Zugriff auf Ordner: " & url
' Instanziierung der Klassen
Set con = CreateObject("ADODB.Connection")
Set rec = CreateObject("ADODB.Record")
' Verbindung aufbauen
con.Provider = "MSDAIPP.DSO"
con.ConnectionString = url
con.Open
' Record auf dieser Connection öffnen
Const adModeReadWrite = 3
rec.Open url, con, adModeReadWrite
say "Ordner geöffnet!"

' --- Zugriff auf einzelne Attribute des Ordners
say "Ordnername: " & rec.fields("DAV:Displayname")
say "URL des Ordners: " & rec.fields("DAV:href")
say "Klasse: " & rec.fields("DAV:contentclass")
say "Übergeordneter Ordner: " & rec.fields("DAV:parentname")
say "Kommentar: " & rec.fields("DAV:Comment")
say "Ordnergröße: " & rec.fields("http://schemas.microsoft.com/
exchange/foldersize")

' --- Alternativ: Alle Attribute ausgeben
' via ADO-Hilfsroutine
list_fields rec
End Sub
```

*Listing 19.13: Beispiel für den Zugriff auf einen Ordner unter Verwendung einer Connection
[ADOEX2K_OrdnerAttributeLesen2.wsfl]*

Ordnerattribute verändern

Das nächste Skript zeigt am Beispiel eines Ordners, wie die Attribute eines an ein Record-Objekt gebundenen Webstore-Eintrags verändert werden können. Wichtig ist dabei, dass Sie beim Öffnen des Records Ihre Schreibabsicht mit `adModeReadWrite` (Wert 3) bekunden. Bitte vergessen Sie auch nicht das abschließende `Update()` nach Verändern der `Fields`-Collection.

**Ordnerattribute
ändern und
Ordner kopieren**

Komponenten für den Exchange Server 2000

```
' ### Ordnerattribute ändern (Ändern des Kommentars zu einem Ordner)
Sub ADOEX2K_Ordner_AttributAenderungen()
Dim rec ' As ADODB.Record
Dim url ' As String

' --- Verbindungsaufbau
' Angaben zum Ordner
Const DOMAIN = "sonne2000.IT-Visions.de"
Const WURZEL = "public"
Const ORDNER = "Lehre/Wintersemester2001_2002/VTBDV1"
' URL zusammenbauen
url = "http://" & DOMAIN & "/" & WURZEL & "/" & ORDNER
say "Zugriff auf Ordner: " & url
' Instanziierung der Klasse
Set rec = CreateObject("ADODB.Record")
' Öffnen des Records zum Lesen und Schreiben
Const adModeReadWrite = 3
rec.Open url, , adModeReadWrite
say "Ordner geöffnet: " & rec.ActiveConnection

' --- Attribut ändern
' Zugriff auf Attribut
Dim f ' As ADODB.Field
Set f = rec.fields("DAV:comment")
' Wert vorher
say "Alter Kommentar: " & f.Value
' Wert ändern
f.Value = "TEST " & Now
rec.fields.Update
' Wert nachher
say "Neuer Kommentar: " & f.Value
End Sub
```

Listing 19.14: Ändern des Kommentars zu einem Ordner [ADOEX2K_AttributAenderungen.wsfl]

Ordnerattribute können nicht hinzugefügt werden

Keine neuen Ordnerattribute

ADO unterstützt zwar das Hinzufügen eines Feldes zu einem Record-Objekt, jedoch unterstützt Exchange nicht das Hinzufügen eines benutzerdefinierten Attributs zu einem Ordner. Es kommt daher zum Fehler, sobald die Update()-Methode des Record-Objekts aufgerufen wird.

Neue Nachrichtenattribute

Natürlich unterstützt Exchange das Anlegen von eigenen Attributen bei Nachrichten. Dies macht den Kern des semi-strukturierten Webstores von Exchange aus.

```
' === Fehler-Beispiel: Anfügen eines Attributs
Sub ADOEX2K_Ordner_NeuesAttribut()
```

Exchange-Programmierung mit ADO

```
Dim rec ' As ADODB.Record
Dim url ' As String

' --- Verbindungsaufbau
' Angaben zum Ordner
Const DOMAIN = "sonne2000.IT-Visions.de"
Const WURZEL = "public"
Const ORDNER = "Lehre/Wintersemester2001_2002/VTBDV1"
' URL zusammenbauen
url = "http://" & DOMAIN & "/" & WURZEL & "/" & ORDNER
say "Zugriff auf Ordner: " & url
' Instanziierung der Klasse
Set rec = CreateObject("ADODB.Record")
' Öffnen des Records zum Lesen und Schreiben
Const adModeReadWrite = 3
rec.Open url, , adModeReadWrite
say "Ordner geöffnet: " & rec.ActiveConnection

' --- Attribut anfügen
rec.fields.Append "Name des Erstellers", adBSTR, 100, adFldFixed, "S"

rec.fields.Update ' DAS GEHT NICHT !!!!!!!
' --- Alle Attribute ausgeben via ADO-Hilfsroutine
list_fields rec
End Sub
```

Listing 19.15: Dieses Skript bricht bei »Update()« mit einem Fehler ab, weil Exchange 2000 es nicht unterstützt, einem Ordner Attribute hinzuzufügen.

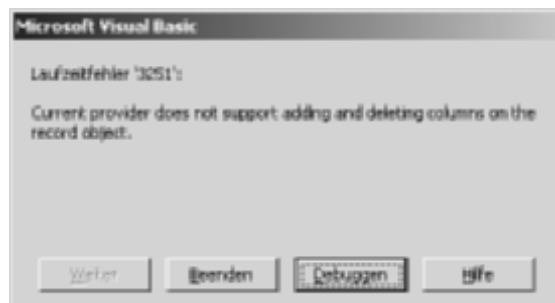


Abbildung 19.12: Fehlermeldung beim Versuch, einem Ordner ein neues Attribut hinzuzufügen

Ordner bewegen

Ordner können mit den Methoden `MoveRecord()` und `CopyRecord()` verschoben bzw. kopiert werden. Beide unterstützen jeweils zwei URL-Parameter: Im ersten Parameter kann die Quelle angegeben werden, im zweiten das Ziel. Beide Parameter müssen ein gültiger Exchange-HTTP- oder File-URL sein. Wenn der erste Parameter leer bleibt, bezieht sich die Aktion auf den aktuellen Ordner.

MoveRecord() und CopyRecord()

DeleteRecord() DeleteRecord() besitzt nur einen URL-Parameter. Ohne diesen Parameter wird der aktuelle Ordner gelöscht, mit diesem Parameter der angegebene.

```
' ### Kopieren, Verschieben und Löschen eines öffentlichen Ordners
Sub ADOEX2K_OrdnerBewegen()
Dim rec ' As ADODB.Record
Dim url ' As String
Dim root, urlquelle, urlziel1, urlziel2

' --- Verbindungsaufbau
' Angaben zum Ordner
Const DOMAIN = "sonne2000.IT-Visions.de"
Const WURZEL = "public"
Const QUELLORDNER = "Lehre/Wintersemester2001_2002/VTBDV1"
Const ZIELORDNER1 = "Lehre/Wintersemester2001_2002/VTBDV1-archiv bis
1.8.2001"
Const ZIELORDNER2 = "Archiv/VBTDV1"

' URL zusammenbauen
root = "http://" & DOMAIN & "/" & WURZEL & "/"
urlquelle = root & QUELLORDNER
urlziel1 = root & ZIELORDNER1
urlziel2 = root & ZIELORDNER2

' Instanziierung der Klasse
Set rec = CreateObject("ADODB.Record")
' Öffnen des Records zum Lesen und Schreiben
Const adModeReadWrite = 3
rec.Open urlquelle, , adModeReadWrite
say "Ordner geöffnet: " & urlquelle

' --- Kopiervorgang
rec.CopyRecord , urlziel1
say "Ordner kopiert nach " & urlziel1

' --- Verschiebevorgang
rec.MoveRecord urlziel1, urlziel2
say "Ordner verschoben nach " & urlziel2

' --- Ordner löschen
rec.DeleteRecord urlziel2
say "Ordner gelöscht: " & urlziel2
End Sub
```

*Listing 19.16: : Kopieren, Verschieben und Löschen eines öffentlichen Ordners
[ADOEX2K_OrdnerBewegen]*

Ordner anlegen

Das Anlegen eines Ordners ist sehr einfach: Bei der Methode `Open()` auf einem `Record`-Objekt muss neben dem URL und `adModeReadWrite` als zusätzlicher Parameter `adCreateCollection` angegeben werden. Der Ordner wird dann erzeugt. Wenn der Ordner schon existiert, kommt es zu einer Fehlermeldung.

**adCreate*
Collection**

```
Const adModeReadWrite = 3
Const adCreateCollection = 8192
rec.Open url, , adModeReadWrite, adCreateCollection
```

Voraussetzung ist auch, dass der übergeordnete Ordner bereits existiert. Der Ordner »Public/Lehre/Wintersemester2001_2002/Studenten« kann also nur angelegt werden, wenn »Public/Lehre/Wintersemester2001_2002« vorher existiert.

Für den neuen Eintrag sind zwei Attribute zu belegen: Einerseits muss die Content Class mit `urn:content-classes:folder` belegt sein. Andererseits unterscheidet Microsoft Outlook verschiedene Ordnertypen gemäß den enthaltenen Nachrichten. In folgendem Listing 19.17 wird ein Kontakt-Ordner (IPF.Contact) erzeugt.

```
' === Anlegen eines neuen Ordners
Sub ADOEX2K_OrdnerAnlegen()
Dim rec ' as ADODB.Record
Dim url ' As String

' --- Verbindungsaufbau
' Angaben zum Ordner
Const DOMAIN = "sonne2000.IT-Visions.de"
Const WURZEL = "public"
Const ORDNER = "Lehre/Wintersemester2001_2002/Studenten"
' URL zusammenbauen
url = "http://" & DOMAIN & "/" & WURZEL & "/" & ORDNER

' Instanziierung der Klasse
Set rec = CreateObject("ADODB.Record")

' --- Ordner erzeugen
Const adModeReadWrite = 3
Const adCreateCollection = 8192
rec.Open url, , adModeReadWrite, adCreateCollection

' --- Attribute festlegen
rec.fields("DAV:contentclass") = _
"urn:content-classes:folder"
rec.fields("http://schemas.microsoft.com/exchange/outlookfolderclass")
= "IPF.Contact"
```

```
' --- Speichern
rec.fields.Update
rec.Close
say "Ordner angelegt: " & url
End Sub
```

Listing 19.17: Anlegen eines neuen öffentlichen Ordners [ADOEX2K_OrdnerAnlegen.wsf]

19.2.6 Zugriff auf Nachrichten

Nachrichtenoperationen

Ein Ordner kann Nachrichten und/oder Unterordner enthalten. Im Folgenden werden die Operationen zum Lesen, Auflisten und Löschen von Nachrichten behandelt. Das Anlegen einer neuen Nachricht ist Thema im Kapitel 19.3.

Zugriff auf eine einzelne Nachricht

Nachrichte lesen

Eine einzelne Nachricht kann direkt per URL spezifiziert (z.B. Lehre/Wintersemester2001_2002/VTBDV1/curriculum.eml) und mit dem Record-Objekt ausgelesen werden. Im folgenden Skript werden einige Informationen auf dem Bildschirm ausgegeben. Anschließend wird die ganze Nachricht auf die Festplatte gespeichert.

```
' === Lesen einer Nachricht und Speicherung als Stream
Sub ADOEX2K_NachrichtLesen()
Dim oStrm ' As ADODB.Stream
Dim rec ' as ADODB.Record
Dim url ' As string

' --- Verbindungsaufbau
' Angaben zum Ordner
Const DOMAIN = "sonne2000.IT-Visions.de"
Const wurzel = "public"
Const ORDNER = _
"Lehre/Wintersemester2001_2002/curriculum.eml"
' URL zusammenbauen
url = "http://" & DOMAIN & "/" & wurzel & "/" & ORDNER
say "Zugriff auf Ordner: " & url
' Instanziierung der Klasse
Set rec = CreateObject("ADODB.Record")
' Öffnen des Records zum Lesen und Schreiben
Const adModeReadWrite = 3
rec.Open url, , adModeReadWrite
say "Nachricht geöffnet: " & rec.ActiveConnection

' --- Informationen über die Nachricht
say "von: " & _
rec.fields("urn:schemas:mailheader:from")
```


Exchange-Programmierung mit ADO

```
say "Betreff: " & _
    rec.fields("urn:schemas:mailheader:subject")
say "am/um: " & _
    rec.fields("DAV:creationdate")
say "Inhalt: " & _
    rec.fields("urn:schemas:httpmail:textdescription")
say " ---- Alle Felder:"
list_fields rec
' --- Speichern als Stream
Const adSaveCreateOverWrite = 2
Const adDefaultStream = -1
Set oStrm = rec.fields(adDefaultStream).Value
oStrm.SaveToFile "d:\buch\curriculum.txt", _
adSaveCreateOverWrite
End Sub
```

Listing 19.18: Lesen einer Nachricht und Speichern als Stream
[ADOEX2K_NachrichtLesen.wsf]

Die folgenden Seiten enthalten die Auflistung aller 72 Attribute einer Nachricht.

```
RESOURCE_PARSENAME=Curriculum.EML
RESOURCE_PARENTNAME=http://sonne2000.it-visions.de/public/Lehre/
Wintersemester2001_2002
RESOURCE_ABSOLUTEPARSENAME=http://sonne2000.it-visions.de/public/
Lehre/Wintersemester2001_2002/Curriculum.EML
RESOURCE_ISHIDDEN=Falsch
RESOURCE_ISREADONLY=Falsch
RESOURCE_CONTENTTYPE=urn:content-classes:message
RESOURCE_CONTENTCLASS=message/rfc822
RESOURCE_CONTENTLANGUAGE=
RESOURCE_CREATIONTIME=25.04.2001 15:12:48
RESOURCE_LASTACCESSTIME=
RESOURCE_LASTWRITETIME=25.04.2001 15:12:48
RESOURCE_STREAMSIZE=1136
RESOURCE_ISCOLLECTION=Falsch
RESOURCE_ISSTRUCTUREDDOCUMENT=Falsch
DEFAULT_DOCUMENT=
RESOURCE_DISPLAYNAME=Curriculum.EML
RESOURCE_ISROOT=
RESOURCE_ISMARKEDFOROFFLINE=Falsch
urn:schemas:httpmail:textdescription=Vorlesungen im Wintersemester:
- Verteilte Betriebliche DV-Systeme #1
- Verteilte Betriebliche DV-Systeme #2
- Technik betrieblicher Kommunikationssysteme
DAV:contentclass=urn:content-classes:message
DAV:revisionid=5gG+hnY/JUmnRPQ+79sGwAAAAABueg==
```

Komponenten für den Exchange Server 2000

urn:schemas:httpmail:fromemail=Geschaeftsfuehrer@it-objects.de
urn:schemas:mailheader:thread-topic=Curriculum
http://schemas.microsoft.com/exchange/permanenturl=http://
sonne2000.IT-Visions.de/public/-FlatUrlSpace-/
e601be86763f2549a7a513d0fbbf6c1b-6d6c/
e601be86763f2549a7a513d0fbbf6c1b-6e7a
DAV:getcontenttype=message/rfc822
DAV:id=AQEAAAAAG1sAQAAAAAbnoAAAAA
urn:schemas:httpmail:thread-topic=Curriculum
DAV:isfolder=Falsch
urn:schemas:httpmail:datereceived=25.04.2001 15:12:31
urn:schemas:mailheader:from="Holger Schwichtenberg"
<Geschaeftsfuehrer@it-visions.de>
urn:schemas:mailheader:received=by Sonne2000.IT-Visions.de id
<01C0CD9A.25F75E70@Sonne2000.IT-Visions.de>; Wed, 25 Apr 2001 17:12:31
+0200
DAV:getetag="e601be86763f2549a7a513d0fbbf6c1b0000000070a3"
http://schemas.microsoft.com/exchange/outlookmessageclass=IPM.Post
DAV:creationdate=25.04.2001 15:12:48
urn:schemas:httpmail:sendername=Holger Schwichtenberg
http://schemas.microsoft.com/exchange/
ntsecuritydescriptor=8;0;4;0;0;0;0;0;1;0;47;140;104;0;0;0;132;0;0;0;0;
0;0;0;20;0;0;0;2;0;84;0;3;0;0;0;0;16;36;0;191;15;31;0;1;5;0;0;0;0;5;
21;0;0;0;67;23;10;50;84;25;14;41;21;37;175;71;238;3;0;0;1;16;20;0;191;
15;31;0;1;1;0;0;0;0;0;5;7;0;0;0;0;16;20;0;169;14;18;0;1;1;0;0;0;0;1;
0;0;0;0;1;5;0;0;0;0;0;5;21;0;0;0;67;23;10;50;84;25;14;41;21;37;175;71;
238;3;0;0;1;5;0;0;0;0;0;5;21;0;0;0;67;23;10;50;84;25;14;41;21;37;175;7
1;1;2;0;0
urn:schemas:mailheader:message-
id=<E601BE86763F2549A7A513D0FBBF6C1B7EA8@Sonne2000.IT-Visions.de>
urn:schemas:mailheader:subject=Curriculum
DAV:autoversion=Falsch
urn:schemas:httpmail:htmldescription=<!DOCTYPE HTML PUBLIC "-//W3C//
DTD HTML 4.0 Transitional//EN">
<HTML><HEAD>
<META content="MSHTML 5.50.4208.1700" name=GENERATOR></HEAD>
<BODY>
<DIV><SPAN class=856391115-
25042001>Vorlesungen im
Wintersemester:</DIV>
<DIV>-
Verteilte
Betriebliche DV-Systeme #1</DIV>
<DIV>
<DIV>-
Verteilte
Betriebliche DV-Systeme #2</DIV>

Exchange-Programmierung mit ADO

```
<DIV><FONT face=Arial size=2><SPAN class=856391115-25042001>
<DIV><FONT face=Arial size=2><SPAN class=856391115-25042001>- Technik
betrieblicher
Kommunikationssysteme</SPAN></FONT></DIV></SPAN></FONT></DIV></SPAN></
FONT></DIV></BODY></HTML>
DAV:ishidden=Falsch
urn:schemas:mailheader:content-class=urn:content-classes:message
http://schemas.microsoft.com/mapi/proptag/0x001A001F=IPM.Post
DAV:parentname=http://sonne2000.IT-Visions.de/public/Lehre/
Wintersemester2001_2002/
urn:schemas:httpmail:subject=Curriculum
DAV:getcontentlength=1136
urn:schemas:httpmail:normalizedsubject=Curriculum
DAV:isstructureddocument=Falsch
http://schemas.microsoft.com/repl/repl-
uid=rid:e601be86763f2549a7a513d0fbbf6c1b000000006e7a
urn:schemas:httpmail:importance=1
DAV:displayname=Curriculum. EML
DAV:href=http://sonne2000.IT-Visions.de/public/Lehre/
Wintersemester2001_2002/Curriculum. EML
urn:schemas:httpmail:from="Holger Schwichtenberg"
<Geschaeftsfuehrer@it-objects.de>
DAV:isversioned=Falsch
DAV:isreadonly=Falsch
urn:schemas:mailheader:date=25.04.2001 15:12:31
DAV:uid=AQEAAAAAG56AAAAAAAAAAAAAA
DAV:revisionlabel=5gG+hnY/JUmpRPQ+79sGwAAAAABueg==
DAV:vresourceid=5gG+hnY/JUmpRPQ+79sGwAAAAABueg==
DAV:getlastmodified=25.04.2001 15:12:48
urn:schemas:httpmail:date=25.04.2001 15:12:31
urn:schemas:httpmail:fromname=Holger Schwichtenberg
urn:schemas:httpmail:submitted=Falsch
http://schemas.microsoft.com/exchange/sensitivity=0
urn:schemas:httpmail:hasattachment=Falsch
DAV:revisionuri=5gG+hnY/JUmpRPQ+79sGwAAAAABueg==
http://schemas.microsoft.com/exchange/defaultrevision=5gG+hnY/
JUmpRPQ+79sGwAAAAABueg==
DAV:iscollection=Falsch
urn:schemas:mailheader:thread-index=AcDNmiXr0vKJbeCmQLyvHq+kiLnojQ==
urn:schemas:httpmail:read=Wahr
http://schemas.microsoft.com/repl/
resourcetag=rt:e601be86763f2549a7a513d0fbbf6c1b000000006e7ae601be86763
f2549a7a513d0fbbf6c1b0000000070a3
urn:schemas:httpmail:priority=0
```

Listing 19.19: Ausgabe aller 72 Attribute einer Standard-Nachricht in einem öffentlichen Ordner

Liste aller Nachrichten

Nachrichten auflisten

Da der Ordnerinhalt eine Menge ist, muss eine SQL-Query mit einem ADODB.Recordset als Rückgabewert verwendet werden. Die Abfrage fordert drei Attribute für die in einem Ordner enthaltenen Elemente an.

```
' === Liste des Inhalts eines Ordners
Sub ADOEX2K_Ordnerinhalt()
Dim rs ' As ADODB.Recordset
Dim query ' As String
Dim url ' As String

' --- Angaben zum Ordner
Const DOMAIN = "sonne2000.IT-Visions.de"
Const WURZEL = "public"
Const ORDNER = "Lehre/Wintersemester2001_2002/VTBDV1"

' --- URL zusammenbauen
url = "http://" & DOMAIN & "/" & WURZEL & "/" & ORDNER
say "Zugriff auf Ordner: " & url

' --- QueryString
query = "select " & "urn:schemas:mailheader:content-class" & " "
query = query & ", " & "DAV:href", " & "DAV:displayname" & " "
query = query & " from scope ('shallow traversal of "
query = query & Chr(34) & url & Chr(34) & "')"
query = query & " ORDER BY " & "DAV:displayname" & " "
say "Abfrage: " & query

' --- Menge anfordern
Set rs = get_ExRS(url, query)

' --- Ausgabe des Inhalts
list_table rs
End Sub
```

Listing 19.20: Liste des Inhalts eines Ordners [ADO_EX2K_Ordnerinhalt.wsf]

GetChildren()

Eine alternative Möglichkeit ist die Verwendung der Methode GetChildren() auf einem Record-Objekt. GetChildren() ist äquivalent zu einem »SELECT * FROM ...«. Bitte beachten Sie, dass der Sternoperator immer nur die Standardattribute, niemals aber selbstdefinierte Attribute einer Webstore-Nachricht liefert. Selbstdefinierte Attribute sind nicht Teil des Schemas und müssen daher namentlich angefordert werden.

```
' === Inhalt eines Ordners auflisten unter Verwendung der
GetChildren()-Methode
Sub ADOEX2K_Ordnerinhalt2()
Dim rs ' As ADODB.Recordset
Dim Query ' As String
```

Exchange-Programmierung mit ADO

```
Dim url ' As string
Dim rec ' As ADODB.Record

' --- Angaben zum Ordner
Const DOMAIN = "sonne2000.IT-Visions.de"
Const WURZEL = "public"
Const ORDNER = "Lehre/Wintersemester2001_2002/VTBDV1"

' --- URL zusammenbauen
url = "http://" & DOMAIN & "/" & WURZEL & "/" & ORDNER
say "Zugriff auf Ordner: " & url
' Instanziierung der Klasse
Set rec = CreateObject("ADODB.Record")
' Öffnen des Records zum Lesen und Schreiben
Const adModeReadWrite = 3
rec.Open url, , adModeReadWrite
say "Ordner geöffnet: " & rec.ActiveConnection

' --- Menge anfordern
Set rs = rec.GetChildren()
' --- Ausgabe des Inhalts
list_table rs
End Sub
```

Listing 19.21: Inhalt eines Ordners auflisten unter Verwendung der »GetChildren()«-Methode [ADOEX2K_Ordnerinhalt2.wsf]

Löschen von Nachrichten

Die üblichen ADO-Funktionen sind verfügbar, z.B. die Methode `Delete()` **Delete()** für das Löschen von Datensätzen.

```
' === Alle Nachrichten aus einem Ordner löschen
Sub ADOEX2K_Ordnerinhaltloeschen()
Dim rs ' As ADODB.Recordset
Dim query ' As String
Dim url ' As string

' --- Angaben zum Ordner
Const DOMAIN = "sonne2000.IT-Visions.de"
Const WURZEL = "public"
Const ORDNER = "Lehre/Wintersemester2001_2002/VTBDV1"

' --- URL zusammenbauen
url = "http://" & DOMAIN & "/" & WURZEL & "/" & ORDNER
say "Zugriff auf Ordner: " & url

' --- QueryString
query = "select " & "urn:schemas:mailheader:content-class" & " "
query = query & ", " & "DAV:href" & ", " & "DAV:displayname" & " "
```

Komponenten für den Exchange Server 2000

```
query = query & " from scope ('shallow traversal of "
query = query & Chr(34) & url & Chr(34) & "'"
query = query & " ORDER BY ""DAV:displayname"" "
say "Abfrage: " & query

' --- Menge anfordern
Set rs = get_ExRS(url, query)

' --- Ausgabe
say "Vorheriger Inhalt:"
list_table rs
say "Löschen aller Nachrichten:"

' --- Alle löschen!
If Not rs.EOF Then rs.MoveFirst
Do Until rs.EOF
    say "Nachricht " & rs("DAV:displayname") & " wird gelöscht!"
    rs.Delete
    rs.MoveNext
Loop

' --- Ausgabe
say "Verbliebener Inhalt:"
list_table rs
End Sub
```

Listing 19.22: Löschen aller Nachrichten
Ausschnitt aus [ADOEX2K_Nachrichtenloeschen.wsfl]

Anlegen einer neuen Nachricht

Neue Nachricht Das Anlegen einer neuen Nachricht ist fast analog zum Anlegen eines neuen Ordners. Wichtig ist die Verwendung von `adCreateNonCollection` statt `adCreateCollection` als Parameter bei der `Open()`-Methode des Record-Objekts. `adCreateOverwrite` sorgt dafür, dass eine eventuell schon vorhandene Nachricht unter diesem URL überschrieben wird.

```
' === Anlegen einer neuen Nachricht
Sub ADOEX2K_NachrichtAnlegen()
Dim rec ' As ADODB.Record
Dim url ' As String

' --- Verbindungsaufbau
' Angaben zum Ordner
Const DOMAIN = "sonne2000.IT-Visions.de"
Const wurzel = "public"
Const ORDNER = "Lehre/Wintersemester2001_2002/news.eml"
' URL zusammenbauen
url = "http://" & DOMAIN & "/" & wurzel & "/" & ORDNER
```

Exchange-Programmierung mit ADO

```
' Instanziierung der Klasse
Set rec = CreateObject("ADODB.Record")

' --- Ordner erzeugen
Const adModeReadWrite = 3
Const adCreateNonCollection = 0
Const adCreateOverwrite = 67108864
rec.Open url, , adModeReadWrite, adCreateNonCollection Or
adCreateOverwrite

' --- Attribute festlegen
rec.fields("DAV:contentclass") = _
"urn:content-classes:message"
rec.fields("urn:schemas:mailheader:from") = _
"Geschaeftsfuehrer@it-visions.de"
rec.fields("urn:schemas:mailheader:subject") = _
"Neuigkeiten!"
rec.fields("urn:schemas:httpmail:textdescription") = _
"Es gibt folgende Neuigkeiten: ..."
' --- Speichern
rec.fields.Update
rec.Close
say "Nachricht wurde gespeichert unter: " & url
End Sub
```

Listing 19.23: Anlegen einer neuen Nachricht mit ADO
[ADOEX2K_NachrichtAnlegen.wsf]

Bewegen einer Nachricht

Das Kopieren, Verschieben und Löschen einer Nachricht ist völlig analog zum Kopieren, Verschieben und Löschen eines Ordners. Anders in dem folgenden Beispiel sind lediglich die URLs.

**Kopieren,
Verschieben und
Löschen**

```
' ### Kopieren, Verschieben und Löschen einer Nachricht
Sub ADOEX2K_NachrichtBewegen()
Dim rec ' As ADODB.Record
Dim url ' As String
Dim root, urlquelle, urlziel1, urlziel2

' --- Verbindungsaufbau
' Angaben zum Ordner
Const DOMAIN = "sonne2000.IT-Visions.de"
Const wurzel = "public"
Const QUELLE = "Lehre/Wintersemester2001_2002/news.eml"
Const ZIEL1 = "Lehre/Wintersemester2001_2002/news-alt.eml"
Const ZIEL2 = "Archiv/news-alt.eml"

' URL zusammenbauen
root = "http://" & DOMAIN & "/" & wurzel & "/"
```

```

urlquelle = root & QUELLE
urlziel1 = root & ZIEL1
urlziel2 = root & ZIEL2

' Instanziierung der Klasse
Set rec = CreateObject("ADODB.Record")
' Öffnen des Records zum Lesen und Schreiben
Const adModeReadWrite = 3
rec.Open urlquelle, , adModeReadWrite
say "Nachricht geöffnet: " & urlquelle

' --- Kopiervorgang
rec.CopyRecord , urlziel1
say "Nachricht kopiert nach " & urlziel1

' --- Verschiebevorgang
rec.MoveRecord urlziel1, urlziel2
say "Nachricht verschoben nach " & urlziel2

' --- Nachricht löschen
rec.DeleteRecord urlziel2
say "Nachricht gelöscht: " & urlziel2
End Sub

```

Listing 19.24: Kopieren, Verschieben und Löschen einer Nachricht
[ADOEX2K_NachrichtBewegen.wsf]

19.3 CDO 3.0 für Exchange 2000

CDO 3.0 Collaboration Data Objects Version 3.0 (CDO 3.0) ist eine völlig neue Komponente, die in nichts mehr an CDO 1.21 erinnert. CDO 3.0 ist die Erweiterung von CDO 2.0 (»CDO für Windows2000«).

CDO 3.0 ist kein durchgängiges Objektmodell, sondern eine Ansammlung einzelner kleiner Objektmodelle und überschneidet sich in vielen Funktionen mit den Möglichkeiten von ADO. CDO 3.0 hat aber dezidierte Klassen für verschiedene Exchange-Elemente (Folder, Person, Message, Appointment, Attendee, CalendarMessage, BodyPart, etc.), während in ADO alles durch die zwei Klassen Recordset und Record abgebildet wird. CDO 3.0 wird nicht automatisch installiert, da der Microsoft Exchange Server auch ohne diese Komponente funktioniert.

Tabelle 19.7:
Schnellinfo
CDO 3.0

Name und Abkürzung	Collaboration Data Objects Version 3.0
Name der Komponentendatei	CDOEX.DLL
Interner Name der Typbibliothek	CDO
Helpstring der Typbibliothek	Microsoft CDO For Exchange 2000 Library

Hersteller	Microsoft
Lizenzierung	Bestandteil von Exchange 2000
Besprochene Version	6.0.4417.0
NT4 und Windows 95/98/ME	nicht unterstützt
Windows2000	Regsvr32 cdoex.dll Die DLL liegt auf der Server-CD unter \\SETUP\\I386\\EXCHANGE\\MICROSOFT SHARED\\CDO.
Dokumentation	MSDN Library [MSL00]

Tabelle 19.7:
Schnellinfo
CDO 3.0
(Forts.)

19.3.1 Neue Konzepte in CDO 3.0

Es gibt zwei neue Konzepte in CDO 3.0, die nicht nur für CDO selbst, sondern insgesamt eine Neuerung innerhalb der automationsfähigen Komponenten darstellen:

- ▶ CDO verwendet Mehrfachschnittstellen auch für den Zugriff über COM-Automation (IDispatch).
- ▶ CDO verwendet zur Bindung an Objekte ADO. Es gibt also eine enge Verzahnung von CDO 3.0 mit ADO 2.5.

Mehrfachschnittstellen

ADO-Integration

Mehrfachschnittstellen-Konzept

CDO 3.0 arbeitet auch in spät bindenden Sprachen (z.B. VBScript und JScript) mit einem Mehrfachschnittstellen-Konzept. Einen Zeiger auf ein Objekt zu haben reicht so oft nicht mehr; es muss auch noch die richtige Schnittstelle angesprochen werden. In CDO 3.0 besitzen viele Objekte Mehrfachschnittstellen, von denen **jede einzelne** IDispatch-fähig ist. Es gibt keine einheitliche IDispatch-Schnittstelle, die eine Obermenge über alle benutzerdefinierten Schnittstellen wäre.

In Kapitel 2 hat dieses Buch erläutert, dass die ActiveX-Skriptsprachen nicht mit Mehrfachschnittstellen umgehen können, weil sie kein Instrument zum Wechsel der Schnittstelle eines COM-Objekts besitzen. Daran hat sich nichts geändert: Skriptsprachen, die COM-Automation voraussetzen, können IUnknown::QueryInterface() nicht ohne Hilfe aufrufen. Daher hat Microsoft in CDO 3.0 in den Klassen selbst einen zusätzlichen Mechanismus realisiert, um einen Zeiger auf implementierte Schnittstellen zurückzuliefern. Dieser Mechanismus ist die Methode

GetInterface()

```
GetInterface("Interfacename").
```

Zusätzlich bieten viele Klassen auch noch ein Attribut an, das direkt zu der Schnittstelle führt. Die beiden folgenden Befehlszeilen sind daher für die Klasse Message identisch:

```
' --- Schnittstellenwechsel via GetInterface()
Set oDS = oMessage.GetInterface("DataSource")
' --- Schnittstellenwechsel via Attribut
Set oDS = oMessage.DataSource
```

19.3.2 CDO 3.0-Objektmodell

Objektmodell

CDO 3.0 bietet 14 Klassen mit insgesamt 33 Schnittstellen. Die nachstehende Grafik zeigt eine der Objekthierarchien in CDO 3.0, in deren Mittelpunkt die zentrale Klasse `Message` steht. Dieses Objektmodell entspricht fast vollständig dem Objektmodell von CDO 2.0 (vgl. Kapitel 20).

*Tabelle 19.8:
Ausgewählte
Klassen in CDO 3.0*

Klasse	Erläuterung
BodyPart	Ein <i>BodyPart</i> ist ein Teil eines Nachrichtentextes oder aber ein Attachment. Besitzt nur die Standardschnittstelle <i>IBodyPart</i> .
BodyParts	Collection von <i>BodyPart</i> -Objekten
Configuration	Enthält die Konfiguration eines <i>Message</i> -Objekts (z.B. Servername, Port, etc.). Besitzt nur die Standardschnittstelle <i>IConfiguration</i> .
DropDirectory	Verwaltung von im Dateisystem gespeicherten Nachrichten. Besitzt nur die Standardschnittstelle <i>IDropDirectory</i> . <i>IDropDirectory.GetMessages</i> liefert eine <i>Messages-Collection</i> .
Message	Repräsentiert eine Nachricht.
Messages	Eine Collection von <i>Message</i> -Objekten
Person	Ein E-Mail-Empfänger im Active Directory

Abbildung 19.13:
Ausschnitt aus dem
Objektmodell von
CDO 3.0

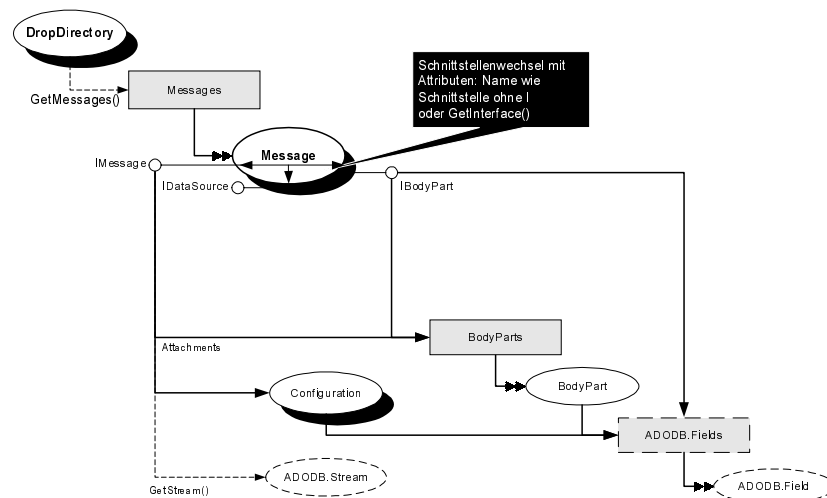




Abbildung 19.14:
Die CDO 3.0-Typ-
bibliothek im VB6-
Objektbrowser

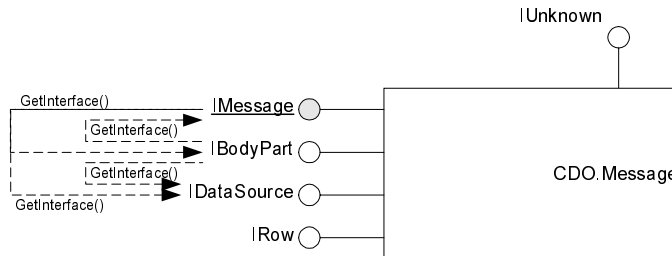
19.3.3 Arbeit mit der Klasse »Message«

Die Message-Klasse besitzt insgesamt vier Schnittstellen:

- ▶ **IMessage** enthält die wesentlichen Attribute der Nachricht und Methoden zum Versenden der Nachricht. IMessage ist die Standardschnittstelle, die bei der Instanziierung der Klasse CDO.Message zurückgeliefert wird.
- ▶ **IBodyPart** enthält Attribute und Methoden zum Zugriff auf den Nachrichteninhalt.
- ▶ **IDataSource** schlägt die Brücke zu ADO und vollzieht die Bindung an eine konkrete Nachricht im Webstore. Hiermit können Nachrichten auch im Webstore gespeichert werden.
- ▶ Die vierte Schnittstelle **IRow** ist eine reine OLE DB-Schnittstelle, die nur via C++ genutzt werden kann und die hier keine weitere Beachtung verdient.

In dem Bullet&Stick-Diagramm in der folgenden Abbildung sind alle Schnittstellen bis auf IRow duale Schnittstellen. IMessage ist grau und unterstrichen, weil dies die Standardschnittstelle ist.

Abbildung 19.15:
Mehrfachschnittstellen in
»CDO.Message«



Innerhalb der IMessage-Schnittstelle führt das Attribut Attachments zu einer BodyParts-Collection, die für jedes Attachment ein BodyPart-Objekt enthält. Diese Collection ist jedoch nicht zu verwechseln mit der BodyParts-Collection, die Sie über das Attribut BodyParts der Schnittstelle IBodyPart erreichen. Letztere Collection enthält BodyPart-Objekte für jeden Teil des Nachrichtentextes.

Beispiel

Nachrichte ausgeben

Das Skript gibt für eine Nachricht im Webstore den Nachrichteninhalte und die Anhänge aus.

```
' ### Das Skript liefert Informationen über eine bestimmte Nachricht
Sub CD03_NachrichtAuswerten()
Dim iDS ' As CDO.IDataSource
Dim iBP ' As CDO.IBodyPart
Dim iMsg ' As CDO.Message
Const adModeReadWrite = 3

' --- Angaben zum Ordner
Const DOMAIN = "sonne2000.IT-Visions.de"
Const WURZEL = "public"
Const ORDNER = "Lehre/Rundschreiben/01_2001.eml"

' --- URL zusammenbauen
url = "http://" & DOMAIN & "/" & WURZEL & "/" & ORDNER

Set iMsg = CreateObject("CDO.Message")
Set iDS = iMsg.DataSource

' --- Binden an eine Nachricht
iDS.Open url, , adModeReadWrite

' --- Basisinformationen
say "Absender:" & iMsg.Sender
say "Betreff:" & iMsg.Subject
say "Body:" & iMsg.TextBody
```

CDO 3.0 für Exchange 2000

```
' --- Teile des Inhalts
say "-- Bodyparts:"
For Each iBP In iMsg.BodyPart.BodyParts
    list_fields iBP
Next

' --- Anhänge
say "-- Attachments:"
For Each iBP In iMsg.Attachments
    say iBP.FileName
    list_fields iBP
    iBP.SaveToFile "d:\buch\anhaenge\" & iBP.FileName
Next
End Sub

' ### Ausgabe aller Felder eines (Record-)Objekts
Sub list_fields(obj)
Dim oF ' As ADO.DB.Field
say "Anzahl der Eigenschaften: " & obj.fields.Count
For Each oF In obj.fields
    If IsNull(oF.Value) Then
        say oF.Name & "= NULL"
    Else
        say oF.Name & "=" & flat(oF.Value)
    End If
Next
End Sub
```

Listing 19.25: Das Skript liefert Informationen über eine bestimmte Nachricht.

19.3.4 Arbeit mit der Klasse »Person«

Über die Klasse `CDO.Person` können Sie Informationen über einen E-Mail-Empfänger abrufen und auch Empfänger anlegen. `CDO.Person` kann sowohl ein Objekt im Active Directory als auch einen Kontakt in einem Kontakt-Ordner im Exchange Webstore repräsentieren.

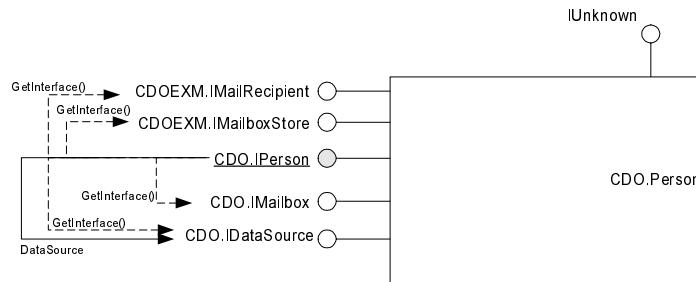
Die Klasse `CDO.Person` ist ein gutes Beispiel dafür, dass bei Microsoft zum Teil sowohl die Typbibliothek als auch die Dokumentation unzureichend ist. Die in `CDOEX.DLL` enthaltene Typbibliothek von CDO 3.0 definiert für die Klasse `Person` nur die Schnittstellen `IPerson`, `IMailbox` und `IDataSource`. Gleiches besagt die Dokumentation. Tatsächlich sind aber auch zwei Schnittstellen aus der Komponente *CDO for Exchange Management* (`CDOEXM`) implementiert (`IMailboxStore` und `IMailRecipient`).

Zu beachten ist auch, dass lediglich die Standardschnittstelle `IPerson` die Methode `GetInterface()` anbietet. Skriptsprachen können also immer nur von dieser Schnittstelle aus zu einer anderen Schnittstelle wechseln, während früh bindende Sprachen gemäß den Regeln zu `IUnknown` von jeder zu jeder anderen Schnittstelle wechseln können.

**Schlechte
Typbibliothek und
Dokumentation**

Eine Abkürzung gibt es über das Attribut `DataSource`, das von `IPerson` direkt zu `IDataSource` führt. `IPerson` ist in der folgenden Abbildung grau und unterstrichen, weil dies die Standardschnittstelle ist.

Abbildung 19.16:
Mehrfachschnittstellen der Klasse
»CDO.Person«



Beispiel

CDO.Person Das folgende Skript ermittelt die E-Mail-Adresse eines Kontakts im Active Directory. Weitere Beispiele zu `CDO.Person` finden Sie im Kapitel 19.4.

```
' --- Abruf von Daten über einen Mailempfänger
Sub CD030_PersonInfo()
Const LDAP_USER = "LDAP://sonne2000/
CN=Steuerberater,OU=Lieferanten,OU=IT-Objects GmbH,
DC=IT-Visions,DC=de"
Dim objPerson As CDO.IPerson
Dim objDS As CDO.IDataSource
Set objPerson = CreateObject("CDO.Person")

' ----- Alternativen:
' 1. Möglichkeit
objPerson.DataSource.Open LDAP_USER
' 2. Möglichket
Set objDS = objPerson.GetInterface("IDataSource")
objDS.Open LDAP_USER
' -----

say objPerson.email
End Sub
```

Listing 19.26: Abruf von Daten über einen Mailempfänger

19.3.5 Exchange-Objekte erzeugen mit CDO 3.0

SaveTo() Das Anlegen von neuen Objekten mit CDO 3.0 ist gewöhnungsbedürftig. Zunächst werden die Objekte extern instanziiert. Dann folgt allerdings nicht – wie aus vielen anderen Objektmodellen bekannt – das Anhängen an eine Collection. Vielmehr werden die Objekte über die `IDataSource`-Schnittstelle

im Webstore gespeichert. Dazu dient die Methode `SaveTo()`, die neben einem URL auch zwei weitere Parameter erwartet.

Ordner anlegen mit CDO.Folder

Die folgende Unterroutine kapselt die Anlage eines Ordners eines beliebigen Typs.

```
' ### Hilfsroutine zum Anlegen eines neuen öffentlichen Ordners
Sub CreateFolder(url, ContentClass, Description)
Const adModeReadWrite = 3
Const adCreateCollection = 8192
Const adCreateOverwrite = 67108864

say "Anlegen des Ordners: " & url
Set oF = CreateObject("CDO.Folder")
oF.ContentClass = ContentClass
oF.Description = Description
oF.DataSource.SaveTo url, , adModeReadWrite, adCreateCollection Or
adCreateOverwrite
say "Ordner angelegt!"

End Sub
```

Listing 19.27: Hilfsroutine zum Anlegen eines neuen Ordners

```
' == Beispiel für das Anlegen eines öffentlichen Ordners
Sub CDO3_neuerOrdner()

' --- Angaben zum Ordner
Const DOMAIN = "sonne2000.IT-Visions.de"
Const WURZEL = "public"
Const ORDNER = "Lehre/Rundschreiben"

' --- URL zusammenbauen
url = "http://" & DOMAIN & "/" & WURZEL & "/" & ORDNER

' --- Ordner anlegen
CreateFolder url, "urn:content-classes:contactfolder", "Ordner nur für
Professoren!"
say "Ordner angelegt!"
End Sub
```

*Listing 19.28: Beispiel für das Anlegen eines Ordners mit CreateFolder()
[CDO3_neuerOrdner.wsf]*

Nachricht erzeugen in einem Ordner mit CDO.Message

Das Skript erzeugt eine neue Nachricht in einem Ordner. Der Vorgang ist fast analog zu dem Anlegen eines Ordners. Der Unterschied zum Ordneranle-

**Neue Nachricht in
einem Ordner**

gen entsteht allerdings nicht durch die Inhaltsklasse (Attribut `ContentClass`), denn über dieses Attribut verfügt die Klasse `CD0.Message` nicht, sondern durch das Weglassen des Flagwerts `adCreateCollection` bei der Methode `SaveTo()`.

```
' === Anlegen einer neuen Nachricht
Sub CD03_neueOrdnerNachricht()
Dim oM ' As CD0.Message
Const adModeReadWrite = 3
Const adCreateOverwrite = 67108864

' --- Angaben zum Ordner
Const DOMAIN = "sonne2000.IT-Visions.de"
Const WURZEL = "public"
Const ORDNER = "Lehre/Rundschreiben/01_2001.eml"

' --- URL zusammenbauen
url = "http://" & DOMAIN & "/" & WURZEL & "/" & ORDNER
say "Anlegen einer neuen Nachricht in: " & url
Set oM = CreateObject("CD0.Message")
oM.Subject = "Rundschreiben 1/2001"
oM.Sender = "hs@it-visions.de"
oM.TextBody = "..."
oM.DataSource.SaveTo url, , adModeReadWrite, adCreateOverwrite
say "Nachricht angelegt!"
End Sub
```

*Listing 19.29: Anlegen einer neuen Nachricht in einem Ordner
[CD03_neueOrdnerNachricht.wsf]*

19.3.6 Senden einer E-Mail

CD03_SendMail() CDO 3.0 unterstützt im Gegensatz zu CDO 1.2 nicht nur das Senden von Nachrichten via MAPI, sondern auch via *Simple Mail Transport Protocol* (SMTP) und *Network News Transport Protocol* (NNTP). CDO 3.0 ist somit eine echte Obermenge von CDO 2.0, da CDO 2.0 nur SMTP- und NNTP-Nachrichten versenden kann. Welche Form des Sendens verwendet wird, muss in einem `CD0.Configuration`-Objekt festgelegt werden. Der Versand von SMTP- und NNTP-Nachrichten ist in Kapitel 20 in Zusammenhang mit CDO 2.0 dokumentiert. Der folgende Skriptcode unterscheidet sich davon nur hinsichtlich der fett markierten Zeilen.



Bitte beachten Sie, dass für die Attributnamen textliche URI-Pfade verwendet werden. Die Werte, mit denen diese Attribute beschrieben werden müssen, sind aber zum Teil numerisch.

CDO 3.0 für Exchange 2000

```
' ### Senden einer E-Mail mit Exchange 2000
Sub cdo3_sendmail(mailbox, von, an, betreff, INHALT)
Dim oConfig ' As CDO.Configuration
Dim oField ' As ADODB.Field
Dim oMsg ' As CDO.Message
Const cdoSendUsingExchange = 3
Const cdoSendUsingMethod = "http://schemas.microsoft.com /cdo/
configuration/sendusing"
Const cdoSendUserName = "http://schemas.microsoft.com/ cdo/
configuration/sendusername"

' --- Instanziieren der notwendigen Klassen
Set oMsg = CreateObject("CDO.Message")
Set oConfig = CreateObject("CDO.Configuration")

' --- Setzen der Konfigurationsinformationen
say "Senden einer E-Mail von Postfach: " & mailbox
With oConfig.fields
    .Item(cdoSendUsingMethod) = cdoSendUsingExchange
    .Item(cdoMailboxURL) = mailbox
    .Item(cdoSendUserName) = von
    .Update
End With

' --- Setzen der Nachrichteneigenschaften
With oMsg
    ' --- Zuweisen der Konfiguration
    .Configuration = oConfig
    .From = von
    .To = an
    .Subject = betreff
    .TextBody = INHALT
    ' --- Anhängen eines Attachments
    Set iBodyPart = .AddAttachment("d:\buch\docs\test.doc")
    iBodyPart.ContentMediaType = "text/html"
    ' --- Versand
    .Send ' Senden
End With

say "E-Mail wurde gesendet an " & an
End Sub
```

Listing 19.30: Senden einer E-Mail mit CDO 3.0 via Exchange 2000

```
' === Senden einer E-Mail mit Exchange2000
Sub CD03_test_sendmail()
' --- Angaben zum Ordner
Const DOMAIN = "sonne2000.IT-Visions.de"
```

Komponenten für den Exchange Server 2000

```
Const WURZEL = "exchange"
Const POSTFACHNAME = "hs"
' --- URL zusammenbauen
url = "http://" & DOMAIN & "/" & WURZEL & "/" & POSTFACHNAME
' --- Senden einer Nachricht
cdo3_sendmail url, "Holger.Schwichtenberg@IT-Visions.de",
"B.Hildebrandt@gmx.net", "Important Message", "You make me happy!"
End Sub
```

Listing 19.31: E-Mail senden mit CDO3_SendMail()
[cdo3_test_sendmail.vbs]

19.4 CDO for Exchange Management

Vereinfachung der Verwaltung mit CDOEXM

Mit Exchange 2000 liefert Microsoft eine weitere Komponente zur Verwaltung des Exchange Servers aus, *CDO for Exchange Management (CDOEXM)*. CDOEXM bündelt Funktionen, die bisher in verschiedenen Komponenten verstreut waren. Dabei vereinfacht es einige Aufgaben sehr stark, die über ADSI zwar auch, aber aufwendiger zu lösen wären. Ganz deutlich merkt man das am Beispiel des Anlegens eines Postfachs für einen bestehenden Benutzer: Während die Lösung für Exchange 5.5 eines der längsten Skripte dieses Buches ist, ist das Skript für Exchange 2000 mit CDOEXM ein Zweizeiler. Der Exchange Server 2000 kann aber natürlich auch weiterhin direkt per ADSI verwaltet werden.

Tabelle 19.9:
Schnellinfo
CDOEXM

Name und Abkürzung	CDO for Exchange Management (CDOEXM)
Name der Komponentendatei	cdoexm.dll
Interner Name der Typbibliothek	CDOEXM
Helpstring der Typbibliothek	Microsoft CDO for Exchange Management Library
Hersteller	Microsoft
Lizenzierung	Bestandteil von Exchange 2000
Besprochene Version	6.0.4417.0
NT4 und Windows 95/98/ME	nicht unterstützt
Windows2000	Installation des Exchange 2000 Servers oder der Client-Tools
Dokumentation	MSDN Library [MSL00]

CDOEXM ermöglicht die Automation folgender administrativer Aufgaben:

- ▶ Anlegen und Verwalten von Postfächern und anderen Empfängern
- ▶ Anlegen und Verwalten von Speichergruppen
- ▶ Anlegen und Verwalten von Exchange-Datenbanken (öffentlichen und privaten Speichern)
- ▶ Anlegen und Verwalten der Hierarchien von öffentlichen Ordnern.

In diesem Buch wird lediglich der erste Punkt beschrieben.

19.4.1 Empfänger-Typen

Auf dem Exchange Server 5.5 gab es drei Typen von Empfängern: Postfächer, Benutzerdefinierte Empfänger und Verteilerlisten. Im Exchange Server 2000 sind zwei weitere Typen hinzugekommen: E-Mail-aktivierte Benutzer und Ordner.

Der Typ E-Mail-aktiver Benutzer ist eine Mischung aus dem, was bisher ein Postfach und ein benutzerdefinierter Empfänger war. Aus der Sicht von Exchange ist ein E-Mail-aktiver Benutzer nur ein benutzerdefinierter Empfänger mit E-Mail-Adresse, aber ohne Postfach. Aus der Sicht des Active Directory ist ein E-Mail-aktiver Benutzer ein vollwertiges Benutzerkonto, das sich an der Domäne anmelden darf. Auch ein Ordner kann direkt Nachrichten über eine E-Mail-Adresse empfangen.

E-Mail-aktivierte Benutzer

Empfänger-typ	E-Mail-Adresse	Speicherung der E-Mails	Verbindung zur NT4-Benutzer-datenbank	Klassenname
Postfach	Ja	Ja	Verknüpft mit einem Benutzerkonto	organizationalPerson
Benutzerdefinierter Empfänger (Weiterleitung)	Ja	Nein, Weiterleitung	keine	remote-address
Verteilerliste	Ja	Nein, Verteilung auf Mitglieder	keine	groupOfNames

Tabelle 19.10:
Empfängertypen in
Exchange 5.5

Tabelle 19.11:
Empfängertypen in
Exchange 2000

Empfängertyp	E-Mail-Adresse	Speicherung der E-Mails	Verbindung zum Active Directory	Active Directory-Klassenname
Postfach	Ja	Ja	Erweiterung eines Benutzers	user
E-Mail-Aktivierter Benutzer	Ja	Nein, Weiterleitung	Erweiterung eines Benutzers	user
Benutzerdefinierter Empfänger	Ja	Nein, Weiterleitung	Erweiterung eines Kontakts	contact
Verteilerliste	Ja	Nein, Verteilung auf Mitglieder	Erweiterung einer Gruppe	group
Öffentlicher Ordner	Ja	Ja, im Ordner	Eigene Klasse	publicFolder

19.4.2 Objektmodell

CDOEXM besteht einerseits aus einer Objekthierarchie für den Bereich Datenbankverwaltung und andererseits aus einigen Schnittstellendefinitionen, die CDO- und ADSI-Klassen erweitern.

In der nachstehend dargestellten Objekthierarchie besitzen alle Klassen eine Fields-Collection mit Field-Objekten.

Abbildung 19.17:
CDOEXM-Objektmodell

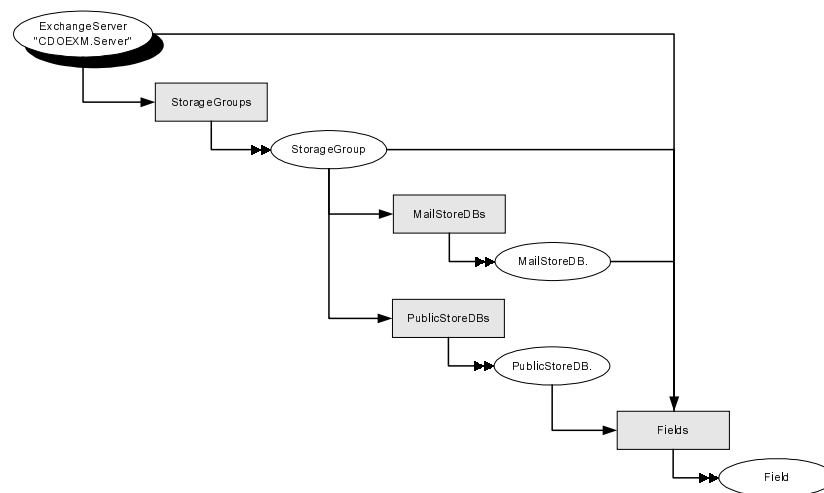


Abbildung 19.18:
Typbibliothek

Mehrfachschnittstellen

CDOEXM verwendet das gleiche Mehrfachschnittstellenkonzept wie CDO 3.0. Die folgende Tabelle zeigt die CDOEXM-Schnittstellen und die Klassen, in denen sie implementiert sind. Die enge Verzahnung von CDOEXM mit CDO 3.0 einerseits und ADSI andererseits wird dadurch deutlich, dass auch ADSI-Klassen von CDOEXM um zusätzliche Schnittstellen erweitert werden.

**Enge Verzahnung
mit CDO 3.0 und
ADSI**

Schnittstelle	Klassen	Erläuterung
IMailRecipient	MailRecipient FolderAdmin CD0.Folder CD0.Person Contact (Active Directory) Group (Active Directory) User (Active Directory)	Verwaltung von Kontakten und Benutzern, denen E-Mail-Adressen zugeordnet sind
IMailbox	CD0.Person	Verwaltung von Postfächern

Tabelle 19.12:
CDOEXM-Schnitt-
stellen und die Klas-
sen, in denen sie
implementiert sind

Komponenten für den Exchange Server 2000

*Tabelle 19.12:
CDOEXM-Schnitt-
stellen und die
Klassen, in denen sie
implementiert sind
(Forts.)*

Schnittstelle	Klassen	Erläuterung
IMailboxStore	Mailbox CDO.Person User (Active Directory)	Verwaltung von Postfächern
IDistributionList	Group (Active Directory) Mailgroup	Verwaltung von Verteilern
IDataSource2	ExchangeServer FolderTree PublicStoreDB MailboxStoreDB StorageGroup	Ermöglicht die Bindung an Exchange-Objekte und die Speicherung (Integration mit ADO wie bei CDO 3.0)
IExchangeServer	ExchangeServer	Konfiguration des Exchange Servers und Zugriff auf die Speichergruppen über die Collection Storage Groups
IStorageGroup	StorageGroup	Verwaltung einer Speichergruppe
IFolderTree	FolderTree	Verwaltung von Ordnerhierarchien
IPublicStoreDB	PublicStoreDB	Verwaltung eines Speichers für öffentliche Ordner
IMailboxStoreDB	MailboxStoreDB	Verwaltung eines Postfachspeichers

*Tabelle 19.13:
Klasse mit den
implementierten
Schnittstellen*

Klasse	Schnittstellen	Erläuterung
User (Active Directory)	IADsUser IMailRecipient IMailboxStore	Benutzer mit Postfach oder E-Mail-Aktivierter Benutzer
Contact (Active Directory)	IADsUser IMailRecipient	Kontakt mit E-Mail-Adresse
Group (Active Directory)	IADsGroup IMailRecipient IDistributionList	Gruppe mit E-Mail-Adresse
CDO.Person	IPerson IDataSource IMailRecipient IMailboxStore	Benutzer mit Postfach oder E-Mail-Aktivierter Benutzer
CDO.Folder	IFolder IDataSource IMailRecipient	Ordner

19.4.3 Anlegen eines Postfachs mit ADSI und CDOEXM

Das Anlegen eines Postfachs für einen Benutzer ist mit CDO for Exchange Management eine unglaublich kurze Angelegenheit, wenn der Benutzer bereits im Active Directory besteht. Sie können einen Benutzer mit dem in Kapitel 13.8 (ADSI für Active Directory) beschriebenen Skript anlegen.

Das Skript bindet an ein ADS-User-Objekt über einen normalen LDAP-Pfad. Ergebnis dieser Operation sollte eine IADs-Schnittstelle sein. So verwundert es, dass darauf die Methode `CreateMailBox()` ausgeführt werden kann. Des Rätsels Lösung liegt darin, dass sich CDOEXM als eine ADSI-Extension für die ADS-Klasse `User` registriert. Dadurch wird die Implementierung der Klasse `User` im ADSI-LDAP-Provider um die Schnittstelle `IMailboxStore` ergänzt. In früh bindenden Umgebungen kann die Objektvariable daher direkt auf `CDOEXM.IMailboxStore` deklariert werden.

In diesem Fall ist in spät bindenden Umgebungen kein Schnittstellenwechsel mit `GetInterface()` durchzuführen, da die Implementierung nicht dem Mehrfachschnittstellen-Paradigma von CDO 3.0/CDOEXM, sondern dem Paradigma von ADSI unterliegt. In ADSI gibt es zwar Mehrfachschnittstellen, aber `IDispatch` ist immer die Obermenge über alle benutzerdefinierten Schnittstellen.

Die folgende Tabelle zeigt die Methoden von `CDOEXM.IMailboxStore`.

Methode	Erläuterung
<code>CreateMailbox (strHomeMDBURL)</code>	Erzeugung eines Postfachs in einem durch einen LDAP-URL zu bestimmenden Exchange-2000-Postfachspeicher
<code>DeleteMailbox()</code>	Löschen einer Mailbox
<code>MoveMailbox (strHomeMDBURL)</code>	Bewegen einer Mailbox in einen anderen Postfachspeicher

`CreateMailbox()` erwartet nur einen Parameter: den LDAP-Pfad zu dem Postfachspeicher (der Exchange-Datenbank), in dem das Postfach abgelegt werden soll. Das Postfach gehört immer zu genau einem Speicher. Der Pfad ist sehr lang; bitte schauen Sie ihn im Configuration Container Ihres Active Directory nach. Auch hier sind englische und deutsche Namen bei den Schlüsselattributen der Verzeichnisobjekte vermischt.

CreateMailBox()

ADSI-Extension



*Tabelle 19.14:
Methoden der
Schnittstelle
IMailboxStore*

**Pfad zum Post-
fachspeicher**

Komponenten für den Exchange Server 2000

Abbildung 19.19:
In den Tiefen des
Configuration
Naming Context
versteckt: Der Post-
fachspeicher
(Ansicht mit dem
Active Directory
Browser
ADSVW.EXE)



Attribute Weitere Attribute müssen in dieser Aktion nicht gesetzt werden: Dem neuen Postfach werden automatisch E-Mail-Adressen gemäß den im Exchange Server eingestellten Adressvorlagen (Aufbau einer E-Mail-Adresse eines bestimmten Typs) und Empfängerrichtlinien (Welchen Typ von E-Mail-Adresse erhält ein Postfach gemäß bestimmten Kriterien?) zugewiesen. Natürlich können Attribute sofort oder später gesetzt werden. Das zeigt eines der später folgenden Skripte. Umfangreich erscheint das nachstehende Skript nur wegen des langen LDAP-Pfads zum Postfachspeicher.

```
' === Anlegen eines Postfachs mit CDOEXM für einen bestehenden AD-User
Sub CreateMailBox()
Dim objMailbox ' As CDOEXM.IMailboxStore

Const LDAP_USER = "LDAP://sonne2000/CN=HS,
OU=IT-Objects GmbH,DC=IT-Visions,DC=de"

Const LDAP_Postfachspeicher = "
LDAP://
```


CDO for Exchange Management

```
CN=Postfachspeicher (SONNE2000),
CN=Erste Speichergruppe,
CN=InformationStore,
CN=SONNE2000,
CN=Servers,
CN=Erste administrative Gruppe,
CN=Administrative Groups,
CN=IT-Objects,
CN=Microsoft Exchange,
CN=Services,
CN=Configuration,
DC=IT-Visions,
DC=de"

say "Postfach erzeugen für: " & LDAP_USER

' --- Zugriff auf Verzeichnisobjekt
Set objMailbox = GetObject(LDAP_USER)

' --- Postfach anlegen
objMailbox.CreateMailBox LDAP_Postfachspeicher
objMailbox.SetInfo

' --- Informationen ausgeben
say "Postfach wurde angelegt für: " & objMailbox.Name
say "Der Benutzer hat folgende SMTP-E-Mail-Adresse erhalten: " & _
objMailbox.SMTPEmail
say "Der Benutzer hat folgende X400-E-Mail-Adresse erhalten: " & _
objMailbox.textEncodedORAddress
End sub
```

Listing 19.32: Anlegen eines Postfachs mit CDOEXM für einen bestehenden Active Directory-Benutzer [CDOEXM_CreateMailBox.wsf]

Alternatives Vorgehen mit CDO 3.0 und CDOEXM

Sie können einen Active Directory-Benutzer auch mit CDO 3.0 anlegen. Diese Lösung sei allein schon deswegen hier vorgestellt, weil die Integration mit CDOEXM eine interessante Konstellation ist.

CDO 3.0 bietet mit der Klasse `CDO.Person` einen eigenen Weg, einen Benutzer anzulegen. Ein so angelegter Benutzer benötigt aber auch die »Behandlung« mit `CreateMailbox()`. Die Schnittstelle `IMailboxStore` ist von der Klasse `CDO.Person` aus erreichbar.

CDO.Person

Sie müssen übrigens in diesem Fall die Schnittstelle – auch in VBScript – explizit wechseln. Das war im ersten Ansatz nicht nötig, da ADSI eine `IDispatch`-Schnittstelle anbietet, die eine Obermenge über alle benutzerdefinierten Schnittstellen ist. Das ist bei CDO 3.0 und CDOEXM – wie geschildert – nicht der Fall.

Schnittstellenwechsel

Komponenten für den Exchange Server 2000

In dem folgenden Skript wird zu Demonstrationszwecken auch noch auf Attribute aus den Schnittstellen CDOEXM.IMailRecipient und CDO.IMailbox lesend bzw. schreibend zugegriffen. Beide Schnittstellen müssen explizit angefordert werden.

```
' === Anlegen eines AD-Users mit Postfach via CDO 3.0 und CDOEXM
Sub CDOEXM_CDOWriteUser()
CDOWriteUser "Sonne2000", "OU=Test,DC=IT-Visions,DC=de", _
"HS", "Holger", "Schwichtenberg"
End Sub
```

```
' ### Anlegen eines AD-Users mit Postfach via CDO 3.0 und CDOEXM
Sub CDOWriteUser(ServerName, Container, emailname, Firstname,
lastname)
Dim objPerson ' As New CDO.Person
Dim objMailboxStore ' As CDOEXM.IMailboxStore
Dim objMailbox ' As CDO.IMailbox
Dim objMailRecipient ' As CDOEXM.IMailRecipient
```

```
Dim u As iadsuser
Dim email As String
Dim ldap_pfad As String
```

```
Const LDAP_Postfachspeicher = "
LDAP://
CN=Postfachspeicher (SONNE2000),
CN=Erste Speichergruppe,
CN=InformationStore,
CN=SONNE2000,
CN=Servers,
CN=Erste administrative Gruppe,
CN=Administrative Groups,
CN=IT-Objects,
CN=Microsoft Exchange,
CN=Services,
CN=Configuration,
DC=IT-Visions,
DC=de"
' --- AD-Benutzer anlegen
Set objPerson = CreateObject("CDO.person")
objPerson.Firstname = Firstname
objPerson.lastname = lastname
objPerson.fields("userPrincipalName") = lastname
objPerson.fields("userAccountControl") = 512
objPerson.fields("userPassword") = "password"
objPerson.fields.Update
ldap_pfad = "LDAP://" & ServerName & "/CN=" & _
```

CDO for Exchange Management

```
emailname & "," & Container
say "Anlegen eines Benutzers als: " & ldap_pfad
objPerson.DataSource.SaveTo ldap_pfad
say "Benutzer angelegt!"

' --- Schnittstelle wechseln
Set objMailboxStore = objPerson.GetInterface("IMailboxStore")

' --- Postfach anlegen
objMailboxStore.CreateMailBox LDAP_Postfachspeicher
objPerson.DataSource.Save
say "Postfach wurde angelegt!"

' --- Empfängerattribute setzen
Set objMailRecipient = objPerson.GetInterface("IMailRecipient")
email = Firstname & "." & lastname & "@IT-Objects.de"
objMailRecipient.HideFromAddressBook = True
objPerson.DataSource.Save
say "Postfach wurde versteckt!"
' --- Postfach anlegen
Set objMailbox = objPerson.GetInterface("IMailbox")
say "Pfad zu Inbox:" & objMailbox.Inbox
End Sub
```

*Listing 19.33: Anlegen eines AD-Users mit Postfach via CDO 3.0 und CDOEXM
[CDOEXM_CDCreateUser.wsf]*

19.4.4 Festlegung der Postfacheigenschaften

Die Eigenschaften des Postfachs sind auf die Schnittstellen `IMailRecipient` und `IMailboxStore` aufgeteilt. In der erstgenannten Schnittstelle befinden sich Angaben wie der Alias und die zugeordneten E-Mail-Adressen. In `IMailboxStore` befinden sich Angaben zu den Speichergrenzen des Postfachs.

**IMailRecipient
versus IMailbox-
Store**

Im folgenden Beispiel ist ein Schnittstellenwechsel wieder NICHT erforderlich, da Ausgangspunkt ein ADSI-Objekt mit einheitlicher `IDispatch`-Schnittstelle ist.

Der folgende Code könnte auch in die zuvor dargestellten Routinen zum Anlegen eines Postfachs jeweils am Ende eingefügt werden.

Limits setzen

```
' === Änderung der Attribute eines Postfachs
Sub CDOEXM_PostfachattributeSetzen()
Dim objMailbox ' KEIN TYP!
Const LDAP_USER = "LDAP://sonne2000/CN=HS,OU=IT-Objects GmbH,DC=IT-Visions,DC=de"
say "Postfach ändern: " & LDAP_USER
```

Komponenten für den Exchange Server 2000

```
' --- Zugriff auf Verzeichnisobjekt
Set objMailbox = GetObject(LDAP_USER)

' --- Adresse setzen
objMailbox.SMTPEmail = "Geschaeftsfuehrer@it-objects.de"
objMailbox.Alias = "GF"

' --- Speichergrenzwerte setzen in KB
objMailbox.EnableStoreDefaults = True ' Standard aus
objMailbox.StoreQuota = 10000 ' Warnmeldung
objMailbox.OverQuotaLimit = 40000 ' Senden verbieten
objMailbox.HardLimit = 60000 ' Senden/Empfangen verbieten

' --- Änderungen speichern
objMailbox.SetInfo
say "Attribute wurden gesetzt!"
End Sub
```

Listing 19.34: Änderung der Attribute eines Postfachs
[CDOEXM_PostfachattributeSetzen.wsf]

Abbildung 19.20:
Vom Skript
eingestellte
Speichergrenzen



Das gleiche Beispiel in VB6

Schnittstellen- jonglieren in VB6

In früh bindenden Umgebungen sieht die gleiche Routine gewaltig anders aus. Damit bekommen Sie einen Eindruck davon, dass das CDO 3.0-/CDOEXM-Mehrfachschnittstellenkonzept die Kompatibilität von VB6- und VB-Script-Code negativ beeinflusst. Hier werden drei Schnittstellenzeiger benötigt:

CDO for Exchange Management

- ▶ ein Zeiger vom Typ `IADsUser` für die Methode `SetInfo()`
- ▶ ein Zeiger vom Typ `objMailRecipient` für die Änderungen der Attribute `SMTPEmail` und `Alias`
- ▶ ein Zeiger vom Typ `IMailboxStore` für die Änderungen der Postfachgrenzen.

Das von `GetObject()` zurückgelieferte User-Objekt unterstützt alle drei Schnittstellen.

```
' === Änderung der Attribute eines Postfachs (NUR VB6!!!)
Sub CDOEXM_PostfachattributeSetzen_VB6()
Dim objUser As ActiveDs.IADsUser
Dim objMailRecipient As CDOEXM.IMailRecipient
Dim objMailbox As CDOEXM.IMailboxStore

Const LDAP_USER = "LDAP://sonne2000/CN=HS,OU=IT-Objects GmbH,
DC=IT-Visions,DC=de"
say "Postfach ändern: " & LDAP_USER
' --- Zugriff auf Verzeichnisobjekt
Set objUser = GetObject(LDAP_USER)
Set objMailbox = objUser
Set objMailRecipient = objUser

' --- Adresse setzen
objMailRecipient.SMTPEmail = "Geschaeftsfuehrer@it-objekts.de"
objMailRecipient.Alias = "GF"

' --- Speichergrenzwerte setzen in KB
objMailbox.EnableStoreDefaults = True ' Standard aus
objMailbox.StoreQuota = 10000 ' Warnmeldung
objMailbox.OverQuotaLimit = 40000 ' Senden verbieten
objMailbox.HardLimit = 60000 ' Senden/Empfangen verbieten

' --- Änderungen speichern
objUser.SetInfo
say "Attribute wurden gesetzt!"
End Sub
```

Listing 19.35: Die gleiche Lösung wie das vorherige VBScript, hier in VB6 mit frühem Binden

Wenn Sie in früh bindenden Umgebungen (z.B. VB6) einen Zeiger auf eine der beiden Schnittstellen `objMailRecipient` oder `IMailboxStore` haben, dann müssen Sie beachten, dass keine der Schnittstellen den Schnittstellenwechsel mit `GetInterface()` unterstützt. Sie müssen also den Schnittstellenwechsel mit `GetInterface()` immer von der `CDO.IPerson` aus vollziehen oder aber – wie im obigen Beispiel – normales `IUnknown`-basiertes Interface-Casting nutzen.



19.4.5 Einrichtung von E-Mail für externe Empfänger

Exchange 2000 kennt zwei Typen von externen Empfängern:

- ▶ externe E-Mail-Adressen für Benutzerkonten
- ▶ externe E-Mail-Adressen für Kontakteinträge.

Unterschied zwischen Benutzer und Kontakt

Aus der Sicht des E-Mail-Routing gibt es keinen Unterschied: In beiden Fällen werden eingehende E-Mails für diese Empfänger nicht gespeichert, sondern gemäß dem Domainbestandteil der E-Mail-Adresse an ein anderes E-Mail-System weitergeleitet. Aus der Sicht des Active Directory sind dies vollwertige Benutzer, die sich an der Domäne anmelden können. Kontakte können sich nicht anmelden.

Externe E-Mail-Adressen für Benutzerkonten bieten sich an für Lieferanten oder Kunden, die dauerhaft in Ihrem Unternehmen vor Ort tätig sind und dort Computer nutzen, aber ihre E-Mails über ihre eigenen E-Mail-Systeme lesen wollen. Externe E-Mail-Adressen für Kontakteinträge bieten sich an, um beliebige externe Personen im globalen Adressbuch aufzuführen.

Arbeit mit der Schnittstelle »IMailRecipient«

IMailRecipient

Programmiertechnisch sind beide Fälle gleich: Die Hauptaufgabe erfüllt die Schnittstelle `CDOEXM.IMailRecipient`. `CDOEXM.IMailRecipient` bietet zwei zentrale Methoden: `MailEnable()` und `MailDisable()`. `ActiveDS.IADsUser` wird nur für die Ausführung von der Methode `SetInfo()` benötigt. Die Schnittstelle `CDOEXM.IMailboxStore` wird hierbei nicht verwendet. Beim Scripting ist ein expliziter Wechsel zwischen `CDOEXM.IMailRecipient` und `ActiveDS.IADsUser` nicht nötig.

Ob ein Benutzerkonto oder ein Kontakt eine E-Mail-Adresse erhalten soll, entscheidet sich lediglich durch die Active Directory-Klasse des angesprochenen Objekts.

Im Code kein Unterschied zwischen Benutzer und Kontakt

Das folgende Beispiel zeigt die Aktivierung der E-Mail für das Objekt `CN=JG,OU=Externe Mitarbeiter,OU=IT-Objects GmbH`. Dabei ist unerheblich, ob dies ein Benutzer oder ein Kontakt ist. Anhand des Inhalts des Attributs `SMTPEmail` wird zunächst geprüft, ob der Benutzer schon eine E-Mail-Adresse hat. In diesem Fall wird die bestehende E-Mail-Adresse deaktiviert.

```
' === Aktivieren einer externen E-Mail-Adresse
Sub CDOEXM_EnableEMAIL()
Dim objMailRecipient ' As CDOEXM.IMailRecipient
Const LDAP_USER = "LDAP://sonne2000/CN=JG,OU=Externe
Mitarbeiter,OU=IT-Objects GmbH,DC=IT-Visions,DC=de"
Const LDAP_Postfachspeicher = "LDAP://CN=Postfachspeicher
(SONNE2000),CN=Erste
Speichergruppe,CN=InformationStore,CN=SONNE2000,CN=Servers,CN=Erste
administrative Gruppe,CN=Administrative Groups,CN=IT-
Objects,CN=Microsoft Exchange,CN=Services,CN=Configuration,
DC=IT-Visions,DC=de"
```

CDO for Exchange Management

```
say "E-Mail aktivieren für: " & LDAP_USER

' --- Zugriff auf Verzeichnisobjekt
Set objMailRecipient = GetObject(LDAP_USER)

' --- Postfach anlegen
If objMailRecipient.SMTPEmail <> "" Then
    say "Benutzer verfügt bereits über eine E-Mail-Adresse: " &
objMailRecipient.SMTPEmail
    objMailRecipient.MailDisable
    say "Bestehende E-Mail-Adresse wurde deaktiviert!"
End If
objMailRecipient.MailEnable "jg@lieferant.de"
objMailRecipient.SMTPEmail = "jg@lieferant.de"
objMailRecipient.Alias = "JGAU"
objMailRecipient.SetInfo

' --- Informationen ausgeben
say "Eine externe E-Mail-Adresse wurde aktiviert für: " & _
    objMailRecipient.Name
say "SMTP-Adresse: " & objMailRecipient.SMTPEmail
End Sub
```

Listing 19.36: Aktivieren einer externen E-Mail-Adresse

20 SMTP-Mailkomponenten

Eine Komponente für den Versand von E-Mails per *Simple Mail Transport Protocol* (SMTP) wird in vielen Anwendungen benötigt. Microsoft bot eine solche Komponente zunächst nur im Rahmen des Site Servers an, so dass viele Entwickler auf die kostengünstigeren Alternativen von anderen Anbietern auswichen. Mit dem NT4-Option Pack stellte Microsoft erstmals eine eigene Mailkomponente als kostenloses Add-on zur Verfügung. Diese Komponente heißt *Collaboration Data Objects for NT Server*, kurz CDONTS. In Windows2000 wurde sie komplett überarbeitet und heißt nun *Collaboration Data Objects for Windows2000* bzw. CDOYS.

CDONTS und CDOYS

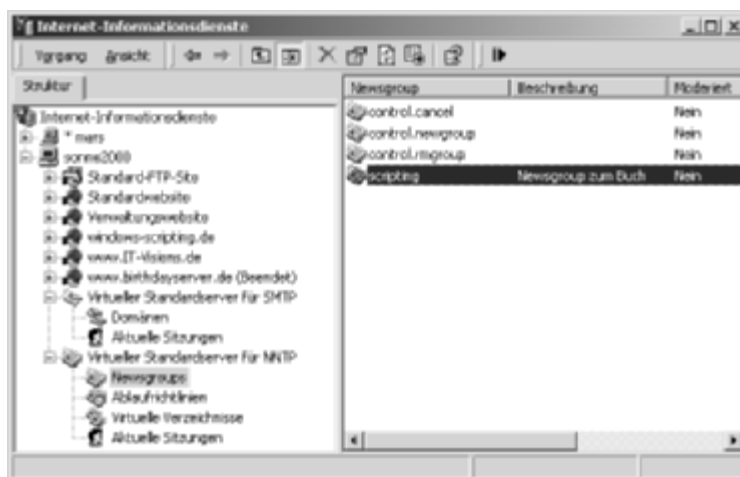


Abbildung 20.1:
Der IIS ist auch ein einfacher SMTP- und NNTP-Server. Der SMTP-Server verwaltet im Gegensatz zum Exchange Server nur einen Posteingang für alle Empfänger. Die Aufteilung der Nachrichten auf verschiedene Empfänger muss per eigener Software erfolgen.

Diese Microsoft-Komponente unterstützt primär das Versenden von Nachrichten in dem Internet-Standard SMTP. Außerdem können im Dateisystem in Form von .EML-Dateien gespeicherte E-Mails ausgelesen werden. Der Hintergrund für diese Funktionalität sind die SMTP-Dienste in NT4 und Windows2000. Diese Dienste nutzen ein so genanntes *DropDirectory* zur Anlage von Nachrichten im .EML-Format.

DropDirectories

Beide Komponenten unterstützen im Gegensatz zu den großen Brüdern CDO 1.2 und CDO 3.0 nicht den Zugriff auf MAPI-Mailstores, also weder auf Postfächer noch auf öffentliche Ordner!

Kein MAPI

In diesem Kapitel wird mit JMAIL auch eine Freeware-Alternative zu CDONTS vorgestellt.

JMAIL

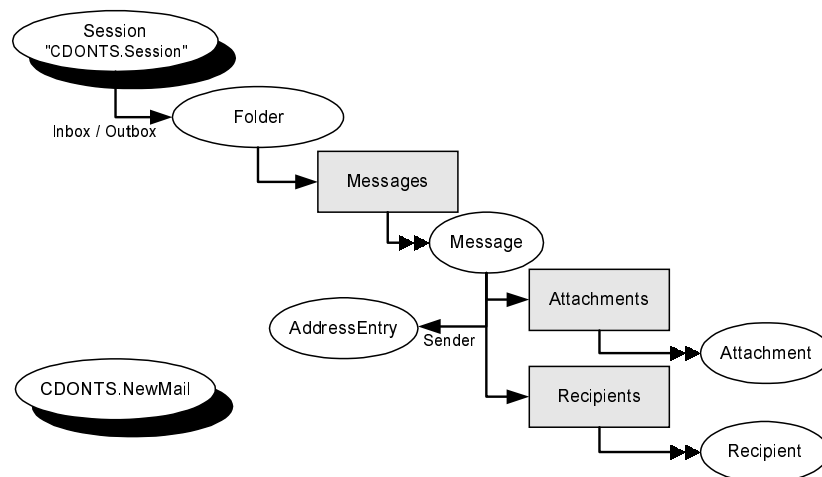
20.1 CDONTS für NT4

CDONTS CDONTS ist eine Light-Version von CDO 1.21 und wird mit dem NT4-Option-Pack mitgeliefert. Unterstützt werden nur SMTP-E-Mails und das Auslesen von in .EML-Dateien gespeicherten Nachrichten. In Windows2000 wurde diese Komponente durch das neuere CDO 2.0 (CDOSYS) ersetzt. CDONTS benötigt als Voraussetzung eine Installation des Internet Explorers ab Version 4.0, da es keinen eigenen SMTP-Dienst mitbringt. Wie der Name schon suggeriert, empfiehlt Microsoft nur den Einsatz auf einem NT-Server.

Tabelle 20.1:
Schnellinfo über die
Mailkomponenten
des Betriebssystems

Name und Abkürzung	CDO for NT Server (CDONTS)
Name der Komponentendatei	cdonts.dll
Interner Name der Typbibliothek	CDONTS
Helpstring der Typbibliothek	Microsoft CDO for NTS 1.2 Library
Hersteller	Microsoft
Lizenzierung	Kostenloses Add-on
Besprochene Version	6.0.3939.0 (DLL-Version). Intern hält sich die Komponente für Version 1.2.
NT4	NT4 Option Pack; alternativ ist die Einzelregistrierung möglich durch: regsvr32 cdonts.dll
Windows 95/98/ME	nicht unterstützt
Windows2000, Whistler (Windows XP / Windows.NET)	enthalten
Dokumentation	MSN Library [MSL00] »Messaging and Collaboration«

Abbildung 20.2:
CDONTS-Objekt-
modell



Objektmodell

Das Objektmodell von CDONTS erinnert stark an CDO 1.21. Die einzelnen Klassen sind jedoch weit weniger mächtig. Hinzugekommen ist die Klasse `NewMail`, die einen schnellen Versand von Nachrichten ermöglicht.



Abbildung 20.3:
CDONTS-
Typbibliothek

Senden einer E-Mail

Das Beispiel zeigt den schnellsten Weg, mit CDONTS eine SMTP-E-Mail zu versenden. Die Klasse hat noch weitere Eigenschaften (z.B. `CC`, `BCC`, `Importance`), die vor dem Senden gesetzt werden können.

```
Dim mailer ' As CDONTS.NewMail
Set mailer = CreateObject("CDONTS.NewMail")
mailer.AttachFile "d:\grafiken\hase.gif"
mailer.Send "hs@it-visions.de", "joerg@bolender.net", "Frohe Ostern!",
"Ich wünsche Dir frohe Ostern!"
say "Gesendet mit CDONTS " & mailer.VERSION
```

Listing 20.1: Senden einer E-Mail mit CDONTS [cdonts_sendmail.vsf]

20.2 CDO for Windows2000

CDO Collaboration Data Objects for Windows2000 ist eine Untermenge der Funktionalität von CDO 3.0 und Nachfolger von CDONTS. CDO for Windows2000 (Kurzname CDOSYS) trägt die Versionsnummer CDO 2.0.

NNTP CDO for Windows2000 unterstützt neben dem Versand von SMTP-Nachrichten und der Aufnahme von .EML-Dateien auch den Versand von Newsgroup-Beiträgen wie NNTP (Network News Transport Protocol). Der NNTP-Server des IIS speichert die Newsgroup-Beiträge in Form von .NWS-Dateien, die nicht über die DropDirectory-Funktionen gelesen werden können.

Während unter NT4 die Ein- und Ausgangsordner durch die Konfiguration des SMTP-Dienstes vorgegeben sind, kann unter Windows2000 jedes beliebige Verzeichnis nach .EML-Dateien durchsucht werden. CDOSYS kann also auch für andere .EML-Dateien genutzt werden, die z.B. beim Export einer einzelnen Nachricht aus Outlook oder Outlook Express entstehen.

Tabelle 20.2:
Schnellinfo CDO
for Windows2000
(CDO 2.0)

Name und Abkürzung	CDO for Windows2000 (CDO 2.0)
Name der Komponentendatei	cdosys.dll
Interner Name der Typbibliothek	CDO
Helpstring der Typbibliothek	Microsoft CDO for Windows2000 Library
Hersteller	Microsoft
Lizenzierung	Bestandteil von Windows2000
Besprochene Version	6.0.3939
NT4, Windows 95/98/ME	nicht verfügbar
Windows2000	enthalten
Whistler (Windows XP / Windows.NET)	enthalten
Dokumentation	MSDN Library [MSL00] »Messaging and Collaboration«

Objektmodell

Objektmodell CDO für Windows2000 unterstützt nur folgende Klassen aus CDO 3.0: BodyPart, BodyParts, Configuration, DropDirectory, Message und Messages. Dabei sind zwar die Schnittstellennamen, nicht aber die Schnittstellendefinitionen identisch. Beispielsweise besitzt `IDataSource` kein `ActiveConnection`-Attribut.

Senden einer E-Mail mit CDOSYS

Zum Senden einer E-Mail mit CDO 2.0 sind folgende Schritte notwendig:

E-Mail senden

- ▶ Instanziiieren der Klassen `CDO.Message` und `CDO.Configuration`
- ▶ Setzen der Konfigurationsparameter (Mailserver, Benutzername, Passwort etc.) über die `Fields`-Collection des `Configuration`-Objekts
- ▶ Zuweisen des `Configuration`-Objekts an das Attribut `Configuration` der `Message`-Klasse
- ▶ Setzen der nachrichtenbezogenen Attribute (z.B. Absender, Empfänger, Betreff, Inhalt)
- ▶ Senden der Nachricht mit der Methode `Send()` auf dem `Message`-Objekt

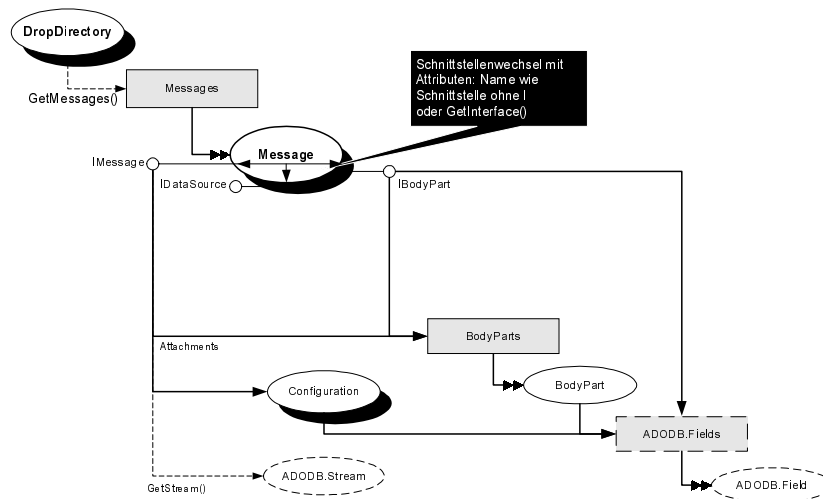


Abbildung 20.4:
Objektmodell von
CDO 2.0

Beiträge in Newsgroups

Sie können mit CDO 2.0 auch Beiträge auf NNTP-Servern absetzen. Dazu beachten Sie bitte folgende Änderungen gegenüber dem SMTP-Versand in der Routine `CDOSYS_send()`. Weisen Sie dazu dem Attribut `Newsgroups` eine Liste der Newsgroup-Namen zu (`oMsg.Newsgroups = AN`) und rufen Sie `Post()` statt `Send()` auf.

Beispiel

Die Routine `CDOSYS_send()` unterscheidet anhand der übergebenen Empfängeradresse zwischen einer E-Mail und einem News-Beitrag. Die Nachricht wird dann per NNTP versendet, wenn der übergebene Empfängerstring kein @ enthält.

CDOSYS_send()

```
Sub CDOSYS_send(VON, AN, BETREFF, INHALT)
Const MAILSERVER = "Sonne2000"
Const NEWSSERVER = "Sonne2000"
```

SMTP-Mailkomponenten

```
Dim oConfig ' As CDO.Configuration
Dim oField ' As ADO.DB.Field
Dim oMsg ' As CDO.Message
'Dim iBodyPart ' As CDO.iBodyPart
Dim News ' As Boolean
' -- E-Mail-Empfänger oder Newsgroup
News = (InStr(AN, "@") = 0)
' -- Instanziieren der notwendigen Klassen
Set oMsg = CreateObject("CDO.Message")
Set oConfig = CreateObject("CDO.Configuration")
' -- Setzen der Konfigurationsinformationen
With oConfig.Fields
    .Item(cdoSMTPServer) = MAILSERVER
    .Item(cdoNNTPServer) = NEWSSERVER
    '.Item(cdoSendUserName) = "username"
    '.Item(cdoSendPassword) = "password"
    .Update
End With
' -- Setzen der Nachrichteneigenschaften
With oMsg
    ' -- Zuweisen der Konfiguration
    .Configuration = oConfig
    .From = VON
    If News Then
        .Newsgroups = AN
    Else
        .To = AN
    End If
    .Subject = BETREFF
    .TextBody = INHALT
    ' -- Versand
    If News Then
        .Post ' Senden per NNTP
    Else
        .Send ' Senden per SMTP
    End If
End With
' -- Kontrollausgabe
If News Then
    say "Newsgroup-Beitrag gesendet an " & AN
Else
    say "E-Mail gesendet an " & AN
End If
End Sub
```

*Listing 20.2: Senden einer E-Mail oder Veröffentlichen in einer Newsgroup mit CDO 2.0
[CDOSYS_send.wsf]*

CDO for Windows2000

```
' -- E-Mail
CDOSYS_send "Holger@Schwichtenberg.de", "Silke.Bartmann@egalwo.de",
"Nette Nachricht", _
"Hallo Silke, ..."

' -- News
CDOSYS_send "hs@windows-scripting.de", _
"Scripting-Diskussionsgruppe", "WSH 5.6", "..."
```

Listing 20.3: Demo-Verwendung von CDOSYS_send() [test_cdosys_send.wsf]

Attachments

CDO 2.0 kann auch Attachments versenden. Die Methode AddAttachment(), die auf der IMessage-Schnittstelle eines Message-Objekts aufzurufen ist, hängt die angegebene Datei an die Nachricht an und liefert eine IBodyPart-Schnittstelle zu diesem Attachment.

Anhänge

```
Dim iBodyPart ' As CDO.iBodyPart
Set iBodyPart = oMsg.AddAttachment("d:\buch\docs\test.doc")
iBodyPart.ContentMediaType = "text/html"
```

Listing 20.4: Anhängen eines Attachments an eine Nachricht

Lesen von .eml-Dateien

Den Zugriff auf in einem Dateisystemverzeichnis gespeicherte .EML-Dateien ermöglicht die Klasse DropDirectory (mit genau einer Schnittstelle: IDropDirectory). IDropDirectory implementiert nur eine Methode: GetMessage(dir) liefert eine CDO.Messages-Collection aller in dem angegebenen Verzeichnis enthaltenen .EML-Dateien. Die Messages-Collection kann mit For...Each durchlaufen werden.

DropDirectories

```
Dim oDropDir ' As CDO.DropDirectory
Dim oMsgs ' As CDO.IMessages
Dim oMsg ' As CDO.Message
Dim a ' Zähler
' -- Name des Verzeichnisses
Const DIR = "\\Sonne2000\w2kssystem\inetpub\mailroot\drop"
' -- Instanz zum Zugriff auf DropDir
Set oDropDir = CreateObject("CDO.DropDirectory")
' -- Zugriff auf Inhalt
Set oMsgs = oDropDir.GetMessages(DIR)
a = 1
If oMsgs.Count = 0 Then say "KEINE NACHRICHTEN!"
' -- Iteration über alle Nachrichten
For Each oMsg In oMsgs
    say "Nachricht " & a & " von " & oMsgs.Count
    say "von " & oMsg.From & " an " & oMsg.To
```

SMTP-Mailkomponenten

```
say "Betreff: " & oMsg.Subject  
say oMsg.TextBody  
Next
```

Listing 20.5: Auslesen der .eml-Dateien in einem Verzeichnis [CDOSYS_get2.wsf]

Verbindung zu ADO

Die IMessage-Schnittstelle kann über die Methode `GetStream()` die Nachricht auch in Form eines `ADODB.Stream`-Objekts liefern. Das Stream-Objekt enthält die komplette Nachricht inklusive aller Header.

```
Dim oStream 'As ADODB.Stream  
Set oStream = oMsg.GetStream
```

Danach stehen alle Möglichkeiten der Stream-Klasse zur Verfügung, z.B.:

```
say oStream.ReadText(-1)  
oStream.SaveToFile "d:\buch\mails" & a & ".txt"
```

20.3 JMAIL

JMAIL JMAIL ist eine beliebte Freeware-Komponente für den Versand von SMTP-E-Mails. Im Gegensatz zu CDONTS kann sie problemlos auf einem Client eingesetzt werden. JMAIL verbindet sich mit einem beliebigen SMTP-Server. Dies kann CDOSYS auch, so dass JMAIL seine Daseinsberechtigung unter Windows2000 verloren hat.

Java?

Das J in JMAIL steht keineswegs – wie viele denken – für Java, sondern für den Vornamen des Schöpfers dieser Komponente, Jonas Frost. JMAIL ist also weder in Java programmiert noch benötigt es Java.

Tabelle 20.3:
Schnellinfo JMAIL

Name und Abkürzung	JMAIL
Name der Komponentendatei	gmail.dll
Interner Name der Typbibliothek	gmail
Helpstring der Typbibliothek	gmail Library
Hersteller	Dimac, http://www.dimac.net/
Lizenzierung	Freeware
Besprochene Version	3.7.0.510
alle Windows-Versionen	Installationsroutine JMAIL.exe
Dokumentation	http://tech.dimac.net/

Nur eine Klasse

SMTPMail JMAIL implementiert nur die Klasse `SMTPMail`. Die in der Typbibliothek ebenfalls enthaltene Klasse `POP3Mail` ist laut Auskunft des Herstellers für kommende Versionen zur Implementierung vorgesehen. Die wichtigsten

JMAIL

Mitglieder der Klasse `SMTPMail` erklären sich durch das folgende Beispiel von selbst. Der Screenshot zeigt weitere Mitglieder.

Beispiel

Das folgende Skript arbeitet analog zum Beispiel zu CDOSYS (vgl. Kapitel 20).

```
JMAIL_Send "Holger@Schwichtenberg", "Sandra.Buse@ueberall.de", _  
"Re:Weltbester und liebster Freund!", _  
"Liebe Sandra, danke für diese tolle Nachricht! Dein Holger"
```

Listing 20.6: Senden einer E-Mail mit JMAIL [test_jmail_send.wsf]

```
Sub JMAIL_Send(von, an, betreff, INHALT)  
Dim JMAIL  
Const MAILSERVER = "Sonne2000"  
Set JMAIL = CreateObject("JMail.SMTPMail")  
' -- Attribute setzen  
JMAIL.ServerAddress = MAILSERVER '(IP oder DNS-Name)  
JMAIL.Sender = von  
JMAIL.Subject = betreff  
JMAIL.AddRecipient an  
JMAIL.body = INHALT  
JMAIL.Priority = 3 ' 1 bis 5, 1 = Höchste  
'JMail.AddHeader "Zusatzinfo", "TEST"  
'JMail.AddAttachment "c:\temp\x.txt"  
' -- Absenden  
JMAIL.Execute  
say "Nachricht gesendet!"  
End Sub
```

Listing 20.7: Senden einer E-Mail mit JMAIL

SMTP-Mailkomponenten

Abbildung 20.5:
JMAIL-
Typbibliothek



21 TCP/IP-Kommunikationskomponenten

In vielen Automatisierungsprojekten werden Kommunikationskomponenten benötigt, die in einem TCP/IP-Netz (sowohl Internet als auch LAN und WAN) Datenkommunikation ermöglichen.

Netzwerk-
kommunikation

Microsoft liefert zwei ActiveX-Steuerelemente, die auch als nichtvisuelle Automatisierungskomponenten verwendet werden können:

- ▶ die WinSock-Komponente
- ▶ die Internet Transfer-Komponente

Gerade die Arbeit mit der Internet Transfer-Komponente ist oft umständlich. Andere Anbieter liefern da wesentlich einfacher zu programmierende Komponenten. Hier werden vorgestellt:

- ▶ ASPInet (eine sehr kleine, einfache FTP-Komponente)
- ▶ Marby FTPX (eine sehr mächtige FTP-Komponente)

21.1 Microsoft WinSock-Komponente

Die WinSock-Komponente ermöglicht den Zugriff auf die Transportschicht des TCP/IP-Protokollstacks (Schicht 4 im ISO/OSI-Referenzmodell) in Form der Protokolle TCP (Transmission Control Protocol) und UDP (User Datagram Protocol). Die Komponente besteht nur aus einer Klasse mit dem Namen WinSock. Die Klasse besitzt zwei Schnittstellen: `IMWinSockControl` und `DMSWinSockControlEvents`. Die erste Schnittstelle definiert Attribute und Methoden, die zweite die Ereignisse der Klasse.

TCP und UDP

Sie können mit Hilfe der WinSock-Komponente sowohl TCP/UDP-Server als auch TCP/UDP-Clients erstellen.

Name und Abkürzung	WinSock-Komponente
Name der Komponentendatei	MSWINSCK.ocx (Typbibliothek: MSWINSCK.ocx)
Interner Name der Typbibliothek	MSWinSockLib
Helpstring der Typbibliothek	Microsoft WinSock-Steuerelement 6.0
Abweichende ProgID	MSWinSock.WinSock
Hersteller	Microsoft
Lizenzierung	Teil von Visual Basic 6.0
Besprochene Version	6.0

Tabelle 21.1:
Schnellinfo
WinSock-Komponente

TCP/IP-Kommunikationskomponenten

Tabelle 21.1:
Schnellinfo Win-
Sock-Komponente
(Forts.)

alle Windows-Versionen	Installation von Visual Basic 6.0; alternativ ist die Einzelregistrierung möglich durch: regsvr32 MSWINSCK.ocx
Dokumentation	MSDN Library [MSL00]

Abbildung 21.1:
MSWinsock-Typ-
bibliothek



Tabelle 21.2:
Attribute der
Schnittstelle IMS-
WinSockControl

Attribut	Erläuterung
Protocol	Auswahl zwischen TCP und UDP über eine Konstante: CONST sockTCPProtocol = 0 CONST sockUDPProtocol = 1
RemoteHost	Name des entfernten Rechners
RemoteHostIP	IP-Adresse des entfernten Rechners
RemotePort	Portnummer auf dem entfernten Rechner
LocalHostName	Name des lokalen Rechners
LocalIP	Lokale IP-Adresse
LocalPort	Portnummer auf dem lokalen Rechner
State	Zustand der Verbindung (Werte 0 bis 9)
BytesReceived	Empfangene Bytes

Microsoft WinSock-Komponente

```

CONST sockClosed = 0
CONST sockOpen = 1
CONST sockListening = 2
CONST sockConnectionPending = 3
CONST sockResolvingHost = 4
CONST sockHostResolved = 5
CONST sockConnecting = 6
CONST sockConnected = 7
CONST sockClosing = 8
CONST sockError = 9

```

Listing 21.1: Konstanten für den Zustand eines WinSock-Objekts

Methode	Erläuterung
Connect(RemoteHost, Remote-Port)	Aufbau der Verbindung zu einem entfernten Rechner
Listen()	Warten auf eine eingehende Verbindung
Accept(requestID)	Akzeptieren einer eingehenden Verbindung
SendData(data)	Übertragung von Daten
GetData(data)	Empfang von Daten
Close()	Schließen einer Verbindung
Bind(LocalPort, LocalIP)	Binden eines TCP/UDP-Servers an eine bestimmte IP-Adresse und Portnummer

Tabelle 21.3:
Methoden der
Schnittstelle IMS-
WinSockControl

Ereignis	Erläuterung
Error(Number, Description, Scode, Source, HelpFile, HelpContext, CancelDisplay)	Es ist ein Fehler aufgetreten.
DataArrival(bytesTotal)	Daten sind eingetroffen.
Connect()	Verbindungsaufbau ist erfolgt.
ConnectionRequest (requestID)	Ein entfernter Client will eine Verbindung aufnehmen.
Close()	Verbindung wurde beendet.
SendProgress(bytesSent, bytesRemaining)	Fortschritt der Übertragung
SendComplete()	Datenübertragung wurde beendet.

Tabelle 21.4:
Ereignisse der
Schnittstelle DMS-
WinSockControl-
Events

Einfaches Beispiel

Sie können die WinSock-Komponente dazu nutzen, den Namen des lokalen Computers und seine IP-Adresse zu ermitteln. Sofern ein Computer über mehrere IP-Adressen verfügt, liefert das Attribut LocalIP dennoch nur die primäre IP-Adresse.

Ermittlung der TCP/IP-Konfigura- tion

TCP/IP-Kommunikationskomponenten

```
Dim oWS ' As MSWinsockLib.Winsock
Set oWS = CreateObject("MSWinsock.Winsock")
say "TCP/IP-Konfiguration:"
say "IP-Adresse: " & oWS.LocalIP
say "Computername: " & oWS.LocalHostName
```

Listing 21.2: Ermittlung von Computername und IP-Adresse [TCPIP_winsock.wsf]

Portscanner

Portscanner Das folgende Beispiel ist ein Portscanner, der überprüft, welche Ports auf einem Rechner abgehört werden (»offen sind«). Da die WinSock-Komponente sehr stark ereignisorientiert arbeitet, ist der Portscanner hier als WSH-Skript implementiert.

```
' === TCP-Portscanner (c) Holger Schwichtenberg 2000
Const UNGRENZE = 1
Const OBERGRENZE = 255
const REMOTEHOST = "sonne2000"
Dim oWS ' As MSWinSockLib.WinSock
Dim result ' Zeigt an, ob es ein Ergebnis von WinSock gab
Dim port ' zu scannender Port
' -- Instanzieren und für Events registrieren
Set oWS = WScript.CreateObject("MSWinSock.WinSock","oWS_")
oWS.aboutbox
say "Client: " & oWS.LocalHostName & "(" & oWS.LocalIP & ")"
For port = UNGRENZE To OBERGRENZE
    If oWS.State <> 0 Then oWS.Close ' Schließen, falls offen
    say "Scanning Port " & Port & "..."
    Result = False ' Flag zurücksetzen
    oWS.Connect REMOTEHOST, port
    ' -- Warten, bis Flag gesetzt
    Do While Not result
        WScript.sleep(1)
    Loop
Next
' -- Hilfsroutine
Sub say(s)
WScript.Echo s
End Sub
' --- Ereignis: Verbindungsaufbau erfolgreich
Sub oWS_Connect()
say "Mit Port " & oWS.remoteport & " verbunden!"
oWS.Close
result = True ' Flag setzen
End Sub
' --- Ereignis: Verbindungsaufbau nicht erfolgreich
Sub oWS_Error( Number , Description , Scode, Source , HelpFile , _
HelpContext , CancelDisplay )
```

Internet Transfer-Komponente

```
say "Fehler bei Port " & a & ": " & Description
oWS.Close
Result = True ' Flag setzen
End Sub
```

Listing 21.3: Ein Portscanner für den Windows Scripting Host [CD: /code/komponenten/
WinSock/_portscanner.vbs]

21.2 Internet Transfer-Komponente

Die Internet Transfer-Komponente bietet die Funktion auf der Anwendungsschicht des TCP/IP-Protokollstacks (Schichten 5-7 des ISO/OSI-Modells) an. Unterstützt werden aber im Wesentlichen nur die Protokolle HTTP (Hypertext Transfer Protocol), FTP (File Transfer Protocol) sowie HTTPS (HTTP over Secure Socket Layer).

HTTP, FTP, HTTPS

Name und Abkürzung	Internet Transfer-Komponente
Name der Komponentendatei	MSINET.ocx (Typbibliothek: MSINET.oca)
Interner Name der Typbibliothek	InetCtlsObjects
Helpstring der Typbibliothek	Microsoft Internet-Steuerelement 6.0
Abweichende ProgID	InetCtls.Inet
Hersteller	Microsoft
Lizenzierung	Teil von Visual Basic 6.0
Besprochene Version	6.0
alle Windows-Versionen	Installation von Visual Basic 6.0; alternativ ist die Einzelregistrierung möglich durch: regsvr32 MSINET.ocx
Dokumentation	MSDN Library [MSL00]

Tabelle 21.5:
Schnellinfo Internet
Transfer-
Komponente

Klassen

Auch diese Komponente enthält nur eine Klasse mit dem Namen `INet`, mit zwei Schnittstellen `IInet` (für Attribute und Methoden) und `DInetEvents` (für genau ein Ereignis).

Abbildung 21.2:
Klasse »INET« im
VB6-Objektkatalog



Tabelle 21.6:
Attribute der
Schnittstelle IInet

Attribut	Erläuterung
Protocol	Verwendetes Protokoll: CONST icUnknown = 0, CONST icDefault = 1, CONST icFTP = 2, CONST icGopher = 3, CONST icHTTP = 4, CONST icHTTPS = 5
RemoteHost	Entfernter Rechner
RemotePort	Port auf dem entfernten Rechner
ResponseCode	Fehlernummer
ResponseInfo	Fehlerbeschreibung
StillExecuting	Zeigt an, ob das Objekt mit einer Anfrage beschäftigt ist (True/False)
URL	URL der gewünschten Ressource
Proxy	Proxy Server
Document	Enthält das Dokument bzw. den Befehl, der mit der Execute()-Methode ausgeführt werden soll
AccessType	Gibt die Art der Internetverbindung an: CONST icUseDefault = 0, CONST icDirect = 1, CONST icNamedProxy = 2
UserName	Benutzername
Password	Kennwort
RequestTimeout	Anzahl der Sekunden, die eine Anfrage dauern darf

Internet Transfer-Komponente

Methode	Erläuterung
OpenURL(URL, [DataType])	Lädt eine Ressource anhand eines URLs. Der optionale Datentyp ist CONST icString = 0 oder CONST icByteArray = 1.
Execute(URL, Operation, Data, InputHdrs)	Führt eine Operation aus
Cancel()	Hält die aktuelle Anfrage an
GetChunk(Size, [DataType])	Liest einen Block der angegebenen Größe aus dem Empfangspuffer
GetHeader(strHdrName)	Liefert die Informationen aus einem Header oder alle Headerdaten

Tabelle 21.7:
Methoden der
Schnittstelle *Inet*

Universeller Event für Zustandsänderungen

Im Gegensatz zur WinSock-Komponente definiert die Internet Transfer-Komponente nicht verschiedene Ereignisse, sondern nur ein Ereignis, das bei allen Zustandsänderungen aufgerufen wird. Das Ereignis `StateChanged` (`state`) teilt über einen Parameter mit, was passiert ist.

StateChanged()

```
' *** StateConstants
CONST icNone = 0
CONST icResolvingHost = 1
CONST icHostResolved = 2
CONST icConnecting = 3
CONST icConnected = 4
CONST icRequesting = 5
CONST icRequestSent = 6
CONST icReceivingResponse = 7
CONST icResponseReceived = 8
CONST icDisconnecting = 9
CONST icDisconnected = 10
CONST icError = 11
CONST icResponseCompleted = 12
```

Listing 21.4: Konstanten für mögliche Zustandsänderungen in der Internet Transfer-Komponente

Synchrone Befehle

Mit der Internet Transfer-Komponente können Abrufe synchron mit `OpenURL()` oder asynchron mit `Execute()` erfolgen. `OpenURL()` ist dabei ein Sonderfall der `Execute()`-Methode, bei der nur ein URL angegeben werden muss. `OpenURL()` eignet sich zum Abruf von Ressourcen via HTTP.

OpenURL()

```
Dim oIC 'As InetCtlsObjects.inet
Dim str 'As String
Set oIC = CreateObject("InetCtls.Inet")
```

```
str = oIC.OpenURL("http://www.windows-scripting.de/start.asp")
say str
```

Listing 21.5: Synchroner Abruf einer Seite per HTTP [inet_sync.wsf]

GetHeader() OpenURL() liefert nur den Inhalt der abgerufenen Ressource. Die HTTP-Header können mit GetHeader() eingelesen werden.

```
str = oIC.OpenURL("http://www.windows-scripting.de/start.asp")
' -- Einzelne Header-Informationen
say oIC.GetHeader("Server")
say oIC.GetHeader("Content-type")
say oIC.GetHeader("Content-length")
say oIC.GetHeader("Cache-control")
' -- Gesamter Header
say oIC.GetHeader()
```

Listing 21.6: Einlesen von Header-Informationen im Rahmen einer HTTP-Anfrage [inet_sync.wsf]

Asynchrone Befehle

Execute() Bei der asynchronen Variante wird Execute() verwendet. Execute() gibt die Kontrolle sofort nach dem Aufruf an den aufrufenden COM-Client zurück. Jeder Versuch, die Daten sofort danach mit GetChunk() abzurufen, führt zu einem Fehler. Der Einbau einer Warteschleife ist keine gute Möglichkeit, denn die Zeit bis zum Eintreffen der Antwort ist nicht deterministisch. Es gibt zwei Möglichkeiten, den Zeitpunkt des Eintreffens der Daten zu ermitteln:

- darauf warten, dass StateChanged(state) mit dem Zustand icResponse Completed (=12) ausgelöst wird
- darauf warten, dass das Attribut StillExecuting den Wert False erhält



In beiden Fällen muss mit dem Statement die Kontrolle zwischenzeitlich an die Umgebung zurückgegeben werden, da sonst die Zustandsänderung nicht eintreten kann. Hier wird die Routine wait() aus der WS_vbWSH.lib verwendet, die WScript.Sleep() (für WSH) bzw. DoEvents (für VB6/VBA) kapselt.

```
Dim str 'As String
Dim oIC 'As InetCtlsObjects.inet
Set oIC = CreateObject("InetCtls.Inet")
say "Verbindungsaufbau..."
' -- Anfrage absenden
oIC.Execute "http://1.1.1.21", ""
' -- Warten bis Antwort eingegangen
Do While oIC.StillExecuting
    Wait ' Erfordert die Einbindung der WS_vbWSH.lib!
Loop
```

Internet Transfer-Komponente

```
' -- Daten blockweise einlesen
Do
    Block = oIC.GetChunk(1024, icString)
    str = str & Block
Loop While Not Len(Block) = 0
' -- Ausgabe
say str
```

Listing 21.7: Asynchrone Befehlsausführung mit Execute() [inet_async.wsf]

Nachdem das Skript solange gewartet hat, bis alle Daten eingetroffen sind, können diese blockweise auf dem Empfangspuffer ausgelesen werden. GetChunk() liefert einen Leerstring zurück, wenn alle Daten empfangen wurden.

GetChunk()

Absenden von FTP-Befehlen

FTP-Befehle können nur mit Execute() ausgeführt werden, da neben dem URL auch ein FTP-Befehl übergeben werden muss. Dieser nimmt bei Execute() den zweiten Platz in der Parameterliste ein. Übermittelt werden können alle gültigen FTP-Befehle (CD, CDUP, GET, PUT, MKDIR, DELETE, etc.) inklusive der zugehörigen Parameter. Im folgenden Skript wird die Hilfsroutine inet_doit() implementiert, die die Befehlsausführung und die anschließende Warteschleife kapselt. Ergebnisse von FTP-Befehlen wie DIR können wieder mit GetChunk() eingelesen werden.

FTP-Befehle

```
Const FTPSERVER = "ftp://1.1.1.20"
Dim str 'As String
Dim oIC 'As InetCtlsObjects.inet
Set oIC = CreateObject("InetCtls.Inet")
' http://www.it-visions.de
' -- FTP Download
inet_doit oIC, FTPSERVER, "GET default.asp d:\buch\download\d.asp"
Do While oIC.StillExecuting
    Wait ' Erfordert die Einbindung der WS_vbWSH.lib!
Loop
say "Datei wurde heruntergeladen..."
' -- FTP Upload
inet_doit oIC, FTPSERVER, _
"PUT d:\buch\download\d.asp kopie_default.asp"
say "Datei wurde hochgeladen..."
inet_doit oIC, FTPSERVER, "CD skripte"
inet_doit oIC, FTPSERVER, "DIR"
' -- Daten blockweise einlesen
Do
    Block = oIC.GetChunk(1024, icString)
    str = str & Block
Loop While Not Len(Block) = 0
say str
```

TCP/IP-Kommunikationskomponenten

```
' ### Ausführung eines Befehls und warten
Sub inet_doit(oiC, ByVal url, command)
oiC.Execute CStr(url), CStr(command)
Do While oiC.StillExecuting
    Wait ' Erfordert die Einbindung der WS_vbWSH.lib!
Loop
End Sub
```

Listing 21.8: Herunterladen einer Datei, Heraufladen unter neuem Namen, Verzeichniswechsel und das Anzeigen eines Verzeichnisinhalts mit FTP[inet_ftp.wsfl]

21.3 ASPInet

Eine sehr schlanke und einfache FTP-Komponente ist *ASPInet* von Stephen Genusa. ASPInet basiert auf der Microsoft Internet Transfer-Komponente und ermöglicht nur den Up- bzw. Download einer Datei, deren Name bereits bekannt ist.

Tabelle 21.8:
Schnellinfo TCP/IP-
Kommunikations-
komponenten

Name und Abkürzung	ASPInet
Name der Komponentendatei	ASPInet.DLL
Interner Name der Typbibliothek	Asplnet
Helpstring der Typbibliothek	Asplnet Library
Hersteller	Stephen Genusa (www.genusa.com)
Lizenzierung	Shareware
Besprochene Version	1.0
alle Windows-Versionen	Einzelregistrierung der DLL: regsvr32 ASPInet.DLL [CD:/install/komponenten/aspinet]
Dokumentation	keine

Klassen

Die Komponente enthält nur eine Klasse FTP mit einem Attribut und zwei Methoden:

- FTPGetFile()** ► FTPGetFile(HostName, UserName, Password, RemoteFileName, LocalFileName, NotOverwrite, TransferType)
- Diese Methode ermöglicht das Herunterladen einer Datei. NotOverwrite = False bedeutet, dass eine bestehende lokale Datei überschrieben würde. NotOverwrite = True bedeutet, dass es zu einem Fehler kommt, wenn es die lokale Datei bereits gibt. Der TransferType ist eine der folgenden Konstanten:
- ```
Const FTP_TRANSFER_TYPE_ASCII = 1
Const FTP_TRANSFER_TYPE_BINARY = 2
```

## ASPInet

- FTPPutFile(HostName, UserName, Password, RemoteFileName, LocalFileName, TransferType)

Mit dieser Methode können Sie eine lokale Datei zum FTP-Server senden.

Beide Methoden geben im Erfolgsfall True zurück, sonst False. Das Attribut GetLastError() liefert dann die Fehlernummer des letzten Fehlers. Diese ist 0, wenn FTPGetFile() daran gescheitert ist, dass die lokale Datei bereits existierte.

**FTPPutFile()**

**GetLastError()**

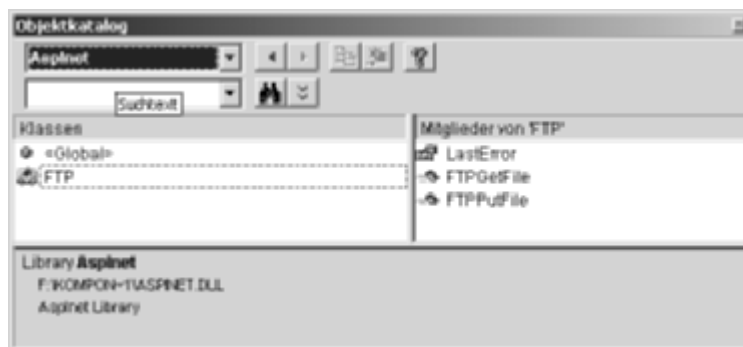


Abbildung 21.3:  
Die Komponente  
ASPInet hat insge-  
samt nur drei  
Mitglieder.

### Beispiel zu ASPInet

Die Datei DEFAULT.ASP wird zunächst heruntergeladen und dann unter dem Namen DEFAULT2.ASP wieder zum Server hochgeladen.

```
Const FTP_TRANSFER_TYPE_ASCII = 1
Const FTP_TRANSFER_TYPE_BINARY = 2
' -- Instanziierung
Set FtpConn = CreateObject("AsPInet.FTP")
' -- Download
ergebnis = FtpConn.FTPGetFile("sonne2000", "hs", "egal", _
 "default.asp", "d:\buch\download\default.asp", False, _
 FTP_TRANSFER_TYPE_BINARY)
If ergebnis <> True Then
 say "Fehler: " & FtpConn.LastError
Else
 say "Download beendet!"
End If
' -- Upload
ergebnis = FtpConn.FTPPutFile("sonne2000", "hs", "egal", _
 "default2.asp", "d:\buch\download\default.asp", _
 FTP_TRANSFER_TYPE_BINARY)
If ergebnis <> True Then
 say "Fehler: " & FtpConn.LastError
Else
 say "Upload beendet!"
End If
```

Listing 21.9: Down- und Upload einer Datei mit ASPInet [ASPINET\_test.wsf]

## 21.4 Mabry FTPX

**Mabry FTPX** Mabry FTPX ist eine sehr mächtige Komponente für den FTP-Datenaustausch. FTPX kapselt die FTP-Befehle in Methoden der Klasse `FtpXObj`. Die Klasse `FtpXObj` besteht aus zwei Schnittstellen `IFtpXObj` (mit Attributen und Methoden) und `IFtpXObjEvents` (mit Ereignissen). Die komplette Dokumentation dieser Komponente finden Sie auf der Buch-CD [CD:/install/komponenten/mabry\_ftpx/mftpx.chm].



Besonderheit von FTPX ist, dass das Inhaltsverzeichnis eines FTP-Ordners in ein ADO-Recordset eingelesen werden kann, so dass ein schneller und komfortabler Zugriff darauf besteht. Dazu wird ein *FTPRecordset OLE DB Provider* mitgeliefert.

Tabelle 21.9:  
Schnellinfo TCP/IP-  
Kommunikations-  
komponenten

|                                        |                                                               |
|----------------------------------------|---------------------------------------------------------------|
| <b>Name und Abkürzung</b>              | FTPX                                                          |
| <b>Name der Komponentendatei</b>       | ftpx.dll                                                      |
| <b>Interner Name der Typbibliothek</b> | FtpXObjLib                                                    |
| <b>Helpstring der Typbibliothek</b>    | Mabry FtpX COM Object                                         |
| <b>Abweichende ProgID</b>              | Mabry.FtpXObj                                                 |
| <b>Hersteller</b>                      | Mabry, <a href="http://www.mabry.com">www.mabry.com</a>       |
| <b>Lizenzierung</b>                    | Kommerzielles Produkt                                         |
| <b>Besprochene Version</b>             | 2.0.3.10                                                      |
| <b>alle Windows-Versionen</b>          | Setup-Routine<br>[CD:/install/komponenten/mabry_ftpx]         |
| <b>Dokumentation</b>                   | Help-Datei [CD:/install/komponenten/<br>mabry_ftpx/mftpx.chm] |

### Beispiel

**Beispiel** Das folgende Skript erfüllt die gleiche Aufgabe wie das Skript im nächsten Listing, dieses Mal jedoch komfortabler mit FTPX realisiert.

```
Dim ftpx ' As FtpXObj
Dim rs ' As ADO.DB.Recordset
Const MSFTP50Pattern = "dddddddddddddd.....ttt.ssssssss.N"
Const FTPSERVER = "1.1.1.20"
' -- Instanziierung
Set ftpx = CreateObject("Mabry.FtpXObj")
' -- Modus
ftpx.Blocking = True
ftpx.BlockingMode = FtpPseudoBlocking
ftpx.host = FTPSERVER
' -- Verbindungsaufbau
ftpx.Connect "HS", ""
ftpx.DirItemPattern = MSFTP50Pattern
```

## Mabry FTPX

```
' -- FTP Download
ftpx.GetFile "default.asp", "d:\buch\download\d2.asp"
say "Datei wurde heruntergeladen..."
' -- FTP Upload
ftpx.PutFile "d:\buch\download\d.asp", "Kopie2_default.asp"
say "Datei wurde heraufgeladen..."
' -- Verzeichnis listen
ftpx.ChangeDir "skripte"
ftpx.GetDirList
Set rs = ftpx.Recordset
list_table rs
```

Listing 21.10: Demo zur Mabry FTP-Komponente [FTPX\_test.wsf]



Abbildung 21.4:  
Einblick in die Typ-  
bibliothek der FTPX-  
Komponente

## Formatbeschreibung von Verzeichnisinhalten

Da die Form der Ausgabe des Verzeichnisinhalts bei FTP-Servern nicht standardisiert ist, verfügt FTPX über eine Methode, die Form des Inhalts zu beschreiben, so dass das Einlesen der Daten in das Recordset korrekt erfolgen kann. »dddddddddddddd.....ttt.ssssssss.N« ist das Pattern für den Microsoft FTP Service 5.0 (Teil des IIS 5.0). Eine detaillierte Beschreibung der Muster finden Sie auf der Homepage des Herstellers [MAR00]. Das Recordset selbst hat dann stets den gleichen Aufbau: TYPE, PERMISSIONS, LINKS, OWNER, GROUP, SIZE, DATE, NAME (Namen der Spalten von links nach rechts).

## DirItemPattern





# 22 Reguläre Ausdrücke mit RegExp

Ein Grund für die starke Verbreitung der Skriptsprache Perl in der Anwendungsentwicklung im Internet sind die mächtigen Funktionen für *Reguläre Ausdrücke* (RAs). Reguläre Ausdrücke sind eine formale Sprache für die Beschreibung von Mustern. Alphanummerische Zeichenketten können mit regulären Ausdrücken verglichen werden, um festzustellen, ob die Zeichenkette einem bestimmten Muster entspricht. Zur Theorie der regulären Ausdrücke siehe [AHO96], S. 703.

Eine rudimentäre Form dieses Mustervergleichs kennt auch das Windows-Dateisystem mit den Platzhaltern »\*« und »?«. Ein Stern steht für beliebig viele Zeichen, ein Fragezeichen für ein einzelnes Zeichen. Beispiel: `dir a*.htm?`. Dieses DOS-Kommando listet alle Dateien auf, deren Name mit »a« beginnt und deren Extension aus vier Zeichen besteht, wobei das letzte ein beliebiges Zeichen sein darf. Leider sind diese Platzhalter sehr beschränkt. Beispielsweise steht das Fragezeichen immer für genau ein Zeichen; es ist also nicht möglich, dass an dieser Stelle gar kein Zeichen steht. Das obige Kommando würde also keine Dateien mit der Extension .HTM finden.

Den Bedarf nach regulären Ausdrücken hat auch Microsoft erkannt und liefert daher seit VBScript 5.1 drei Klassen (in VBScript 5.5 sind es vier) aus, mit denen reguläre Ausdrücke definiert und mit Zeichenketten verglichen werden können. Diese Klassen bilden keine eigenständige Komponente, sondern sind Bestandteil der VBSCRIPT.DLL. Man könnte zwar nun vermuten, dass nur VBS diese Klassen nutzen kann; das ist jedoch zum Glück nicht so: Alle Umgebungen können darauf zugreifen. Freilich ist die Voraussetzung immer, dass VBScript installiert ist. Bleibt die Frage, warum Microsoft für eine so mächtige Funktion keine eigene DLL übrig hatte.

**Reguläre Ausdrücke**

**Mustervergleich in MS-DOS**

**Teil der VBScript.dll**

|                                            |                                                    |
|--------------------------------------------|----------------------------------------------------|
| <b>Name und Abkürzung</b>                  | RegExp                                             |
| <b>Name der Komponentendatei</b>           | VBSCRIPT.DLL                                       |
| <b>Interner Name der Typbibliothek</b>     | VBScript_RegExp                                    |
| <b>Helpstring der Typbibliothek</b>        | Microsoft VBScript Regular Expressions             |
| <b>Abweichende ProgID</b>                  | VBScript.RegExp                                    |
| <b>Hersteller</b>                          | Microsoft                                          |
| <b>Lizenzierung</b>                        | Kostenloses Add-on                                 |
| <b>Besprochene Version</b>                 | 5.6.0.6625                                         |
| <b>NT4, Windows2000, Windows 95/98/ME</b>  | Installation des WSH 5.6 [CD:/install/hsots/WSH]]. |
| <b>Whistler (Windows XP / Windows.NET)</b> | enthalten                                          |
| <b>Dokumentation</b>                       | MSDN Library [MSL00]                               |

Tabelle 22.1:  
Schnellinfo RegExp

## 22.1 RegExp-Objektmodell

**Objektmodell** Die *RegExp*-Komponente besteht aus drei Klassen. Stammobjekt und als einzige instanzierbar ist die Klasse *RegExp*. Nach der Instanziierung muss einem *RegExp*-Objekt über das Attribut *Pattern* ein Muster übergeben werden. Dies ist die Grundvoraussetzung. Weitere Optionen sind über die Attribute *IgnoreCase* und *Global* verfügbar.

Abbildung 22.1:  
RegExp-  
Objektmodell

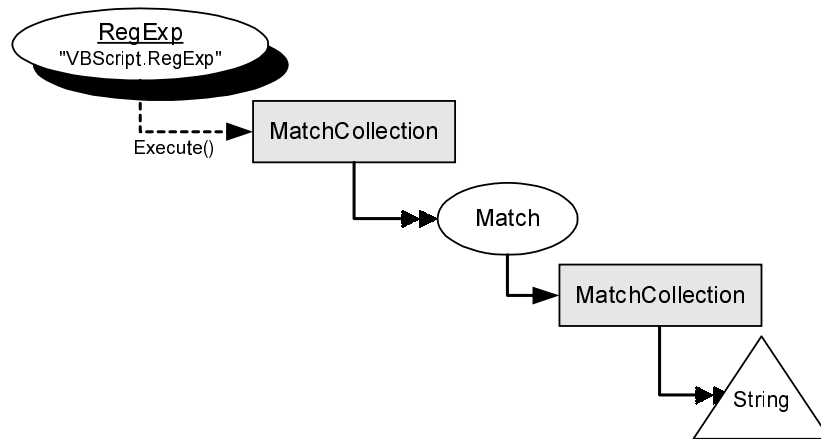
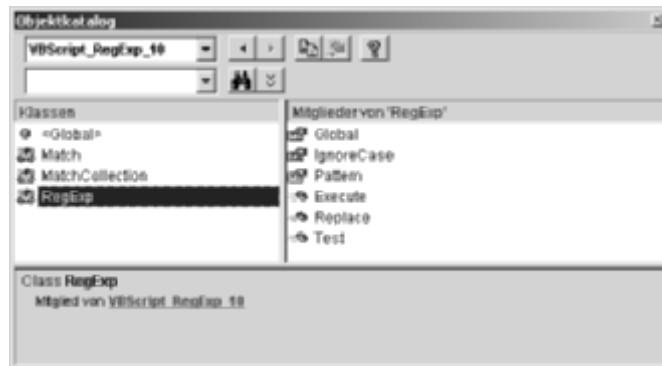


Abbildung 22.2:  
Die Klasse  
»RegExp« im  
VB6-Objektkatalog



### 22.1.1 Klasse »RegExp«

**Instanziierung** Auf Grund der Tatsache, dass *RegExp* Teil der VBSCRIPT.DLL ist, ist bei der Verwendung innerhalb von VBScript eine Angabe des Komponentennamens nicht notwendig und eine Instanziierung mit dem *New*-Statement möglich, was sonst für COM-Klassen nicht erlaubt ist.

```
Set objRA = new RegExp
```

### RegExp-Objektmodell

Aus allen anderen Sprachen ist jedoch die übliche Form zu verwenden:

```
Set objRA = CreateObject("VBScript.RegExp")
```

Bitte beachten Sie, dass auch bei dieser Komponente die ProgID einen von der Typbibliothek abweichenden Komponentennamen enthält. Das große VB will es also so lesen:

```
Set objRA = new VBScript_RegExp.RegExp
```

### Mitglieder

Nach der Instanziierung der Klasse `RegExp` stehen drei Methoden und vier Attribute zur Verfügung (siehe folgende Tabellen).

### Mitglieder

| Methode                                                                                 | Erläuterung                                                                                                                                                                                                                                                                                     |
|-----------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>Test(sourceString As STRING)<br/>As BOOLEAN</code>                                | Ein einfacher Vergleich eines Strings mit dem Muster. Das Ergebnis ist <i>True</i> oder <i>False</i> .                                                                                                                                                                                          |
| <code>Execute(sourceString As<br/>STRING) As MatchCollection</code>                     | Die Ausführung eines detaillierten Vergleichs, bei dem bei Übereinstimmung die Position, die Länge und der genaue Inhalt der Fundstelle festgehalten werden. Wahlweise wird nur die erste aller Fundstellen aufgezeichnet. Die Aufzeichnung erfolgt konsequenterweise in Form einer Collection. |
| <code>Replace(sourceString As<br/>STRING, replaceString As<br/>STRING) As STRING</code> | Die Ersetzung einer Fundstelle (wahlweise aller Fundstellen) durch eine andere Zeichenkette. Zurückgegeben wird die modifizierte Zeichenkette.                                                                                                                                                  |

Tabelle 22.2:  
Methoden der Klasse  
*RegExp*

| Attributname            | Erläuterung                                                                                                                                                                                                                                                                                           |
|-------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>Pattern</code>    | Der reguläre Ausdruck in Form eines Strings. Die Syntax der regulären Ausdrücke ist im nächsten Kapitel dokumentiert.                                                                                                                                                                                 |
| <code>IgnoreCase</code> | <i>True</i> bedeutet, dass zwischen Groß- und Kleinschreibung unterschieden wird. <i>False</i> ist die Standardeinstellung und macht diese Unterscheidung nicht.                                                                                                                                      |
| <code>Global</code>     | <i>False</i> (Standardeinstellung) bedeutet, dass nur das erste Vorkommen in die <code>Matches</code> -Collection aufgenommen wird. <i>True</i> bedeutet, dass alle Vorkommen aufgezeichnet werden. Dieser Parameter hat daher nur Bedeutung in Zusammenhang mit der Methode <code>Execute()</code> . |

Tabelle 22.3:  
Attribute der Klasse  
*RegExp*

## 22.2 Einfacher Mustervergleich mit Test()

**test()** Das erste Beispiel ist sehr einfach gehalten – so einfach, dass es dazu nicht einmal eines regulären Ausdrucks bedurft hätte: Die VBScript-Funktion InStr() hätte ausgereicht, denn das Pattern enthält gar keinen Platzhalter.

```
' Enthält der String "von"?
Set objRA = New RegExp
objRA.pattern = "von"
MsgBox objRA.test("Mein Name ist Alexandra von Künsberg-Langenstadt.")
End Sub
```

*Listing 22.1: Beispiel für die Test()-Methode [RA\_test1.wsf]*

Im zweiten Beispiel wird jedoch der erste Platzhalter (das Dach »^«) verwendet, um zu prüfen, ob der String mit einem bestimmten Muster beginnt. Der erste Test ist erfolgreich, der zweite nicht.

```
' Beginnt der String mit LDAP:// ?
Set objRA = New RegExp
objRA.pattern = "^LDAP://"
say objRA.test("LDAP://sonne") ' ist Wahr
say objRA.test("LDAP-Beispiel: LDAP://sonne")' ist Falsch
```

*Listing 22.2: Beispiel für die Test()-Methode [RA\_test2.wsf]*

### Funktion RAtest()

**RAtest()** Für die weiteren Tests sollte man besser eine eigene Funktion verwenden, die einen prägnanteren Zugriff ermöglicht und einen boolschen Wert zurückliefert.

```
Function RAtest(str, pattern, ignorecase) ' As BOOLEAN
Set objRA = New RegExp
objRA.pattern = pattern
objRA.ignorecase = ignorecase
RAtest = objRA.test(str)
End Function
```

*Listing 22.3: Hilfsroutine RAtest()*

Es folgen einige Anwendungsbeispiele. Die Konstante s steht dabei für die Eingabe "https://www.xy.de/bla/x.htm".

## RA-Symbole

| Vergleich                                                            | Ergebnis | Erläuterung                                                                                                                                                                                                                                                                                                                                           |
|----------------------------------------------------------------------|----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>RAtest(s, "^http://", True)</code>                             | False    | String enthält https statt http                                                                                                                                                                                                                                                                                                                       |
| <code>RAtest(s, "^https?://", True)</code>                           | True     | Das Fragezeichen steht bei regulären Ausdrücken nicht für ein Zeichen, sondern für das kein- oder einmalige Vorkommen des vorherigen Ausdrucks.                                                                                                                                                                                                       |
| <code>RAtest(s, "^HTTPS?://", False)</code>                          | False    | Groß- und Kleinschreibung passen nicht.                                                                                                                                                                                                                                                                                                               |
| <code>RAtest(s, "^https?://(\w+\.)+\w+/\w+/\w+\.htm\$", True)</code> | True     | Dieser Ausdruck beschreibt die Eingabe ganz genau. \w steht für Buchstaben, der Backslash vor dem Punkt wird als Escape-Kennzeichen benötigt, da ein Punkt ein RA-Platzhalter ist. Die Klammer markiert einen Block, das Pluszeichen bedeutet, dass der vorhergehende Platzhalter bzw. Block ein- oder mehrmals vorkommen muss. \$ markiert das Ende. |

Tabelle 22.4:  
Beispiele für  
Mustervergleiche  
[RA\_test3.wsf]

Der letzte RA ist aber immer noch nicht perfekt, denn auch die folgende Überprüfung ist erfolgreich, obwohl derart kurze Domainnamen nicht gültig sind:

Noch nicht perfekt

```
say RAtest("http://w.x.e/bla/x.htm", _
"^https?://(\w+\.)+\w+/\w+/\w+\.htm$", True)
```

## 22.3 RA-Symbole

Die nachfolgenden Tabellen erläutern alle verfügbaren Platzhalter in vier Gruppen: Platzhalter für Zeichen, Blockbildung und Alternativen, Wiederholungszeichen und Escape-Befehle, die die Bedeutung von Befehlszeichen außer Kraft setzen.

Symbole

| RA-Symbol           | Erläuterung                                                                                                                                                                                                                                                                                                   |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>[abc]</code>  | Die eckigen Klammern definieren eine Menge von Zeichen, von denen genau eines vorkommen muss. Dieses Symbol ist zeichenweise zu interpretieren: Z.B. passt <code>[XU]ML</code> auf XML und UML, aber nicht auf XUML. Hinweis: Innerhalb einer solchen Menge können keine anderen RA-Symbole verwendet werden! |
| <code>[^abc]</code> | Definiert ebenso wie das vorherige Symbol eine Menge, allerdings ist durch die Verwendung des Dachses diese Zeichenmenge ausgeschlossen. Beispiel <code>[^UX]</code> passt auf AML, BML, CML, etc., nicht aber auf UML und XML.                                                                               |

Tabelle 22.5:  
Platzhalter für  
Zeichen in regulären  
Ausdrücken

Tabelle 22.5:  
Platzhalter für  
Zeichen in regulären  
Ausdrücken  
(Forts.)

| RA-Symbol | Erläuterung                                                                                                                                                                                                                                                                                                                                        |
|-----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| [a-z]     | In einer Menge können auch Zeichenbereiche angegeben werden. Dabei können innerhalb einer eckigen Klammer mehrere Wertebereiche angegeben werden und auch einzelne Zeichen zusätzlich auftreten.<br>Beispiel 1: [a-h0-9] passt auf das hexadezimale Alphabet.<br>Beispiel 2: [a-k135] passt auf die Buchstaben a bis j und die Ziffern 1, 3 und 5. |
| [^m-z]    | Auch Zeichenbereiche können mit dem Dach ausgeschlossen werden.                                                                                                                                                                                                                                                                                    |
| \d        | Passt auf eine Ziffer und ist äquivalent zu [0-9] (siehe unten)                                                                                                                                                                                                                                                                                    |
| \D        | Passt auf eine Nicht-Ziffer (an dieser Stelle darf keine Ziffer stehen). \D ist äquivalent zu [^0-9] (siehe unten).                                                                                                                                                                                                                                |
| \w        | Passt auf alle alphabetischen Zeichen einschließlich Unterstrich und ist äquivalent zu [A-Za-z0-9_].                                                                                                                                                                                                                                               |
| \W        | Ist das Gegenteil zu \w und passt daher auf alle Zeichen, auf die \w nicht passt                                                                                                                                                                                                                                                                   |
| .         | Passt auf alle Zeichen außer einem Zeilenumbruch                                                                                                                                                                                                                                                                                                   |
| \f        | Passt auf einen Seitenvorschub                                                                                                                                                                                                                                                                                                                     |
| \n        | Passt auf einen Zeilenumbruch                                                                                                                                                                                                                                                                                                                      |
| \r        | Passt auf einen Wagenrücklauf                                                                                                                                                                                                                                                                                                                      |
| \t        | Passt auf einen Tabulator                                                                                                                                                                                                                                                                                                                          |
| \v        | Passt auf einen vertikalen Tabulator                                                                                                                                                                                                                                                                                                               |
| \s        | Passt auf das Leerzeichen und andere Leerräume (Tabulator, Seitenvorschub usw.) und ist äquivalent zu [\f\n\r\t\v]                                                                                                                                                                                                                                 |
| \S        | Ist das Gegenteil zu /s und passt auf alle Nicht-Leerräume (entspricht daher [^\f\n\r\t\v])                                                                                                                                                                                                                                                        |
| ^         | Passt auf den Anfang der Eingabe                                                                                                                                                                                                                                                                                                                   |
| \$        | Passt auf das Ende der Eingabe                                                                                                                                                                                                                                                                                                                     |
| \b        | Passt auf eine Wortgrenze                                                                                                                                                                                                                                                                                                                          |
| \B        | Passt auf alle Zeichen außer einer Wortgrenze                                                                                                                                                                                                                                                                                                      |

Tabelle 22.6:  
Blockbildung und  
Alternativen in  
regulären Aus-  
drücken

| RA-Symbol | Erläuterung                                                                                                                                                                                                                                                                                                                                                |
|-----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| (Muster)  | Ermöglicht die Bildung von Blöcken aus Symbolen und Literalen. Dies ist für die Verwendung des ODER-Symbols ( ) und der Wiederholungszeichen (*, +, ? sowie { }) wichtig. Mit Hilfe von Submatches können diese Blöcke jeweils einzeln extrahiert werden. Bei der Verwendung von Replace() kann mit \$Zahl auf die einzelnen Blöcke Bezug genommen werden. |

## RA-Symbole

| RA-Symbol                       | Erläuterung                                                                                                                                                           |
|---------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| $x y$                           | Entspricht entweder x oder y<br>Beispiel: $W(95) (98) (NT) (2000)$ passt auf »W95«, »W98«, »WNT« und »W2000«                                                          |
| $\backslash \text{BlockNummer}$ | Rückbezug auf einen durch Klammern markierten Block.<br>Damit kann ausgedrückt werden, dass ein zuvor gefundener Teilausdruck sich an dieser Stelle wiederholen muss. |

Tabelle 22.6:  
Blockbildung  
und Alternativen in regulären Ausdrücken  
(Forts.)

| RA-Symbol | Erläuterung                                                                                                                                                                                                         |
|-----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| *         | Das Zeichen, das Symbol oder der Block vor dem Stern kann mehrmals vorkommen, muss es aber nicht. Das Fehlen (keinmal) ist also auch erlaubt.                                                                       |
| +         | Das Zeichen, das Symbol oder der Block vor dem Plus muss mindestens einmal vorkommen. Das Fehlen (keinmal) ist also nicht erlaubt.                                                                                  |
| ?         | Das Zeichen, das Symbol oder der Block vor dem Fragezeichen darf höchstens einmal vorkommen. Das Fehlen (keinmal) ist erlaubt, die mehrmalige Wiederholung aber nicht.                                              |
| {n}       | Das Zeichen, das Symbol oder der Block vor der geöffneten geschweiften Klammer muss genau n-mal vorkommen.                                                                                                          |
| {n,}      | Das Zeichen, das Symbol oder der Block vor der geöffneten geschweiften Klammer muss mindestens n-mal vorkommen. Die Obergrenze kann also unendlich groß sein. {1,} ist äquivalent mit +. {0,} ist äquivalent mit *. |
| {n,m}     | Das Zeichen, das Symbol oder der Block vor der geöffneten geschweiften Klammer muss zwischen n- und m-mal vorkommen.                                                                                                |

Tabelle 22.7:  
Wiederholungszeichen in regulären Ausdrücken

| RA-Symbol       | Erläuterung                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| \               | Unterdrückt die Interpretation des folgenden Zeichens als ein RA-Symbol und gibt dem Zeichen damit seine literale Bedeutung zurück. Beispiele:<br><ul style="list-style-type: none"> <li>▶ <math>(x)?</math> markiert eine Wiederholung von »x«.</li> <li>▶ <math>(x)\backslash?</math> entspricht dem Literal »x?».</li> <li>▶ <math>\backslash\backslash</math> entspricht einem Backslash.</li> <li>▶ <math>\backslash(x\backslash)</math> entspricht »(x)«.</li> </ul> |
| $\backslash n$  | Folgt dem Backslash eine Zahl, so wird diese als ein ASCII-Code interpretiert. Beispiel: $\backslash 34$ entspricht einem Anführungszeichen.                                                                                                                                                                                                                                                                                                                               |
| $\backslash xn$ | Folgt dem Backslash ein x, so wird die dem x folgende Zahl als Hexadezimaldarstellung eines ASCII-Codes interpretiert. Beispiel: $\backslash x22$ entspricht einem Anführungszeichen.                                                                                                                                                                                                                                                                                      |

Tabelle 22.8:  
Escape-Charaktere in regulären Ausdrücken

## 22.4 Ausführlicher Mustervergleich mit Execute()

**Execute()** Für den ausführlichen Test mit `Execute()` stellt die `RegExp`-Komponente eine Liste der Fundstellen in Form einer `Collection` bereit. `Execute()` liefert den Zeiger auf eine `Matches`-Collection, die `Match`-Objekte enthält.

Tabelle 22.9:  
Attribute der Klasse  
`Match`

| Attributname            | Erläuterung                                                                                      |
|-------------------------|--------------------------------------------------------------------------------------------------|
| <code>FirstIndex</code> | Position des ersten Zeichens der Fundstelle im Eingabestring. Das erste Zeichen hat den Index 0. |
| <code>Length</code>     | Länge der Fundstelle in Form einer Zahl                                                          |
| <code>Value</code>      | Exakter Inhalt der Fundstelle (String)                                                           |

**MatchesCollection** Die Klasse `MatchesCollection` verfügt nur über die üblichen Attribute `Item` und `Count`. `Add()` und `Remove()` werden nicht unterstützt, da die `Collection` vom Anwender nicht verändert werden soll. Die Herangehensweise ist intuitiv zu verstehen und im nachstehenden Beispiel dokumentiert.

```
Dim objMatches, objM
' -- Beispiel
Const muster = "\.[a-z]{3,4}\b"
Const eingabe = ".htm wsf vbs .js .123 .html .zulang .abl .asp"
' -- Instanziierung und Wertübergabe
Set objRA = New RegExp
objRA.pattern = muster
objRA.Global = True
' Ausführung des ausführlichen Tests
Set objMatches = objRA.Execute(eingabe)
say "Anzahl der Fundstellen:" & objMatches.Count
' Rekursion über die Fundstellen
For Each objM In objMatches
 say objM.Value & " an Position " & objM.FirstIndex & _
 " Länge: " & objM.Length
Next
```

*Listing 22.4: Beispiel zur Verwendung von `Execute()` und `MatchesCollection` [Sub `RA_Exec1.wsf`]*

Das Beispiel liefert als Ausgabe:

```
.htm an Position 0 Länge: 4
.wsf an Position 11 Länge: 4
.html an Position 29 Länge: 5
.asp an Position 48 Länge: 4
```



### Ausführlicher Mustervergleich mit Execute()

Der reguläre Ausdruck "\.[a-z]{3,4}\b" ist so zu verstehen:

|            |                                                                                                                                                                                                                                                  |
|------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| \. :       | Am Anfang des Teilausdrucks muss ein Punkt stehen. (Hier darf nicht das Dach »^« verwendet werden, da ja ein Teilausdruck gesucht wird, der auch innerhalb der Eingabe vorkommen darf. Das Dach passt nur auf den absoluten Anfang der Eingabe.) |
| [a-z]{3,4} | Danach folgen mindestens drei, aber höchstens vier Buchstaben.                                                                                                                                                                                   |
| \b         | Dann kommt eine Wortgrenze.                                                                                                                                                                                                                      |

Folgende Teile der Eingabe passen daher nicht:

- ▶ »....«: Nach dem Punkt folgt kein Buchstabe.
- ▶ »vbs«: Es fehlt der Punkt am Anfang.
- ▶ »js«: Ausdruck besteht nicht aus mindestens drei Buchstaben.
- ▶ ».123«: Die Extension darf keine Zahlen enthalten.
- ▶ ».zulang«: Extension ist zu lang.
- ▶ ».ab1«: Die Extension darf keine Zahlen enthalten.

### Funktion RAfilter()

Sehr nützlich ist die nachstehende Funktion RAfilter(): Sie entnimmt alle passenden Muster aus einer Eingabe und erzeugt daraus wahlweise ein Array oder einen String, in dem alle Fundstellen mit einem bestimmten Trennzeichen getrennt sind. Sofern das Trennzeichen den Wert *False* enthält, wird ein Array erzeugt, sonst wird dieser Wert als Trennzeichen für den String benutzt.

RAfilter()

```
' Rückgabewert: String oder Array of String
Function RAfilter(eingabe, muster, trennzeichen)
Dim objRA, objMatches, objM ' Objekte
Dim filter_array() ' Array
Dim i ' Zähler
Set objRA = New RegExp
RAfilter = ""
objRA.pattern = muster
objRA.Global = True
' Ausführung des ausführlichen Tests
Set objMatches = objRA.Execute(eingabe)
If trennzeichen = False Then
 ' --- Rückgabe als Array
 i = 0
 ReDim filter_array(objMatches.Count - 1)
 For Each objM In objMatches
 filter_array(i) = objM.Value
 i = i + 1
 Next
```

### Reguläre Ausdrücke mit RegExp

```
RAfilter = filter_array
Else
' --- Rückgabe als String
For Each objM In objMatches
 If RAfilter <> "" Then RAfilter = RAfilter & trennzeichen
 RAfilter = RAfilter & objM.Value
Next
End If
End Function
```

*Listing 22.5: RAfilter() ist eine nützliche Funktion zum Extrahieren von passenden Teilausdrücken.*

#### Dynamischer Array als Rückgabewert

Ein Hinweis zur Implementierung: Die Erstellung des Rückgabewerts unterscheidet sich bei der Variante »Array of String« insofern von der Variante »String mit Trennzeichen«, als es hier zwingend notwendig ist, eine lokale Variable `filter_array` anzulegen, in der das Ergebnis zwischengespeichert wird. Eine direkte Zuweisung an den Funktionsnamen wie in der zweiten Variante ist in Visual Basic nicht möglich, da ein `ReDim` auf einem Funktionsnamen verboten ist. Die spätere Zuweisung an den Funktionsnamen ist jedoch möglich und liefert dem Aufrufer auch das Array korrekt zurück.

Der folgende Code testet `RAfilter()`.

```
Dim ergebnis, i
' -- Beispiel 1
muster = "[a-z]{3,4}\b"
eingabe = ".htm wsf vbs .js .123 .html .zulang .abl .asp"
' -- Variante 1
say "Rückgabe als String mit Semikolon als Trenner"
say RAfilter(eingabe, muster, ";")
' -- Variante 2
say "Rückgabe als Array"
ergebnis = RAfilter(eingabe, muster, False)
For i = LBound(ergebnis) To UBound(ergebnis)
 say "Fund " & i + 1 & ":" & ergebnis(i)
Next
```

*Listing 22.6: Testbeispiel für die Filter()-Funktion [RA\_Exec2.wsf]*

Die Ausgabe für dieses Beispiel ist:

```
Rückgabe als String mit Semikolon als Trenner
.htm;.wsf;.html;.asp
Rückgabe als Array
```

## Submatches

Fund 1:.htm  
Fund 2:.wsf  
Fund 3:.html  
Fund 4:.asp

## 22.5 Submatches

Neu in VBScript 5.5 gegenüber VBScript 5.1 ist die Unterstützung von Submatches. Submatches sind Teilstrings innerhalb einer Fundstelle. Diese werden durch runde Klammern »( )« abgegrenzt. Submatches sind sinnvoll, wenn ein Muster in verschiedene Teilmuster aufgespalten werden soll. Grundsätzlich lässt sich diese Aufgabe auch ohne Submatches erfüllen, indem zunächst das Gesamtmuster und dann die Fundstelle mit jedem Teilmuster einzeln geprüft wird; Submatches machen diese Aufgabe jedoch einfacher.

### Submatches

Ein Match-Objekt enthält eine Collection Submatches, die eine Menge von Strings enthält. Die Routine RA\_submatch(eingabe, muster) liefert alle Teilausdrücke innerhalb der ersten Fundstelle eines Musters in einer Eingabe.

```
Sub RA_submatch(eingabe, muster)
Dim s, re, matches
Set objRA = New VBScript_RegExp_55.RegExp
' Muster für eine E-Mail-Adresse
objRA.pattern = muster
objRA.ignorecase = True ' Ignoriert die Schreibweise.
objRA.Global = False ' Mehrere Fundstellen zulassen.
Set objMatches = objRA.Execute(eingabe)
' --Prüfung
If objMatches.Count <> 1 Then
 say "Muster nicht oder mehr als einmal gefunden!"
 Exit Sub
End If
' -- Nur 1. Fundstelle betrachten...
Set objMatch = objMatches(0)
say "Fundstelle: " & objMatch.Value
say "Teilausdrücke: " & objMatch.SubMatches.Count
For i = 0 To objMatch.SubMatches.Count - 1
 say "- Teilausdruck #" & i & " = " & objMatch.SubMatches(i)
Next
End Sub
```

*Listing 22.7: Liste aller Teilausdrücke in einem Muster (nur für die erste Fundstelle!)*

Tabelle 22.10:  
Beispiel für die Ver-  
wendung von  
RA\_submatch()

| Aufruf                                                                                            | Ausgabe                                                                                                                                                                                                                   |
|---------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| RA_submatch<br>"hs@windows-<br>scripting.de",<br>"(\w+)\@(\w+-?\w+).(\w+)"                        | Fundstelle: hs@windows-scripting.de<br>Teilausdrücke: 3<br>- Teilausdruck #0 = hs<br>- Teilausdruck #1 = windows-scripting<br>- Teilausdruck #2 = de                                                                      |
| RA_submatch "Schwichten-<br>berg@kom. wi-inf.uni-<br>essen.de", "(\w+)\@((\w+-<br>?\w+\.)+)(\w+)" | Fundstelle: Schwichtenberg@kom.<br>wi-inf.uni-essen.de<br>Teilausdrücke: 4<br>- Teilausdruck #0 = Schwichtenberg<br>- Teilausdruck #1 = kom.wi-inf.uni-essen.<br>- Teilausdruck #2 = uni-essen.<br>- Teilausdruck #3 = de |

#### Unzulänglich- keiten

Das zweite Beispiel zeigt bereits die Grenzen der Submatches: Bei Blöcken, die mit dem Wiederzeichen »+« versehen sind, wird nur der erste Teilausdruck, der auf `(\w+-?\w+\. )+` passt, in die Submatches-Collection aufgenommen. Da runde Klammern auch zur Blockbildung genutzt werden, erscheint zudem mit »kom.wi-inf.uni-essen.« ein Teilausdruck, den man eigentlich gar nicht in dieser Form haben wollte. Gegen diese Unzulänglichkeit hilft aber eine wiederholte Anwendung eines regulären Ausdrucks auf die Teilergebnisse. Grundsätzlich lässt sich jedes Musterproblem mit den regulären Ausdrücken lösen.

## 22.6 Musterersetzung mit Replace()

**Replace()** Ein einfacher Einsatz der Ersetzungsmethode `Replace()` aus der Klasse `RegExp`: Diese Aufgabe hätte auch die VB-Funktion `Replace()` erfüllen können.

```
Dim s
s = "<HTML>Dies ist Fettdas nichtDas ist auch Fett
</html>"
Set objRA = New RegExp
objRA.Global = True
objRA.pattern = ""
s = objRA.Replace(s, "<i>")
objRA.pattern = ""
s = objRA.Replace(s, "</i>")
objRA.pattern = "Fett"
s = objRA.Replace(s, "Kursiv")
say s
```

Listing 22.8: Beispiele zur Verwendung der `Replace()`-Methoden [RA\_Replace1.wsf]

Die Ausgabe lautet

```
<HTML><i>Dies ist Kursiv</i>das nicht<i>Das ist auch Kursiv</i></html>
```

### *Musterersetzung mit Replace()*

Auf elegantere Weise aus *Thomas Rünker* den String *Rünker, Thomas* zu machen bleibt jedoch den regulären Ausdrücken vorbehalten:

```
s = "Thomas Rünker"
Set objRA = New RegExp
objRA.pattern = "(\\w+) (\\w+)"
s = objRA.Replace(s, "$2, $1")
say s
```

*Listing 22.9: Umformung eines Namens mit Replace() [RA\_Replace3.wsf]*

Der Zauber ist schnell enthüllt: Zwar können in der Ersetzungszeichenkette nicht die üblichen RA-Platzhalter verwendet werden, dafür kann aber mit `$Zahl` auf einen im Muster mit runden Klammern umrandeten Block Bezug genommen werden.

### **Funktion RAClean()**

Auch hierzu soll eine hilfreiche Funktion präsentiert werden: `RAClean()` **RAClean()** löscht aus einer Eingabe alle passenden Teilausdrücke.

```
Function RAclean(eingabe, muster) ' As String
Dim objRA
Set objRA = New RegExp
objRA.pattern = muster
objRA.Global = True
RAclean = objRA.Replace(eingabe, "")
End Function
```

*Listing 22.10: RAClean() löscht alle Vorkommen eines Musters in einem Ausdruck.*

```
Function test_RAClean()
Const muster = "<[>]+>"
Const eingabe = "<p align='center'><h2 class='buch'>Windows- und
BackOffice-Scripting</h2></p>"
say RAclean(eingabe, muster)
End Function
```

*Listing 22.11: Beispiel für den Einsatz von RAClean(): Entferne alle HTML-Tags. [RA\_Replace3.wsf]*



# 23 WindowsScripting-Komponente

In diesem Buch darf natürlich auch eine eigene Komponente nicht fehlen. Diese Komponente stellt nur einige wesentliche nützliche Funktionen in drei Klassen bereit.

**Komponente zu diesem Buch**

|                                        |                                                                                                                                                   |
|----------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name und Abkürzung</b>              | WindowsScripting-Komponente                                                                                                                       |
| <b>Name der Komponentendatei</b>       | WindowsScripting.dll                                                                                                                              |
| <b>Interner Name der Typbibliothek</b> | WindowsScripting                                                                                                                                  |
| <b>Helpstring der Typbibliothek</b>    | WindowsScripting Komponente                                                                                                                       |
| <b>Hersteller</b>                      | Holger Schwichtenberg,<br><a href="http://www.IT-Visions.de">http://www.IT-Visions.de</a>                                                         |
| <b>Lizenzierung</b>                    | Freeware                                                                                                                                          |
| <b>Besprochene Version</b>             | 1.0                                                                                                                                               |
| <b>alle Windows-Versionen</b>          | Einzelregistrierung der DLL:<br>regsvr32 WindowsScripting.dll<br>[CD:/install/komponenten/windows-scripting]<br>Benötigt Visual Basic 6.0 Runtime |
| <b>Dokumentation</b>                   | Nur in diesem Buch                                                                                                                                |

*Tabelle 23.1:  
Schnellinfo  
WindowsScripting-Komponente*

## Klassen

Die Komponente besitzt kein Objektmodell, sondern besteht nur aus einzelnen Klassen.

**Klassen**

| Klassenname | Erläuterung                                                                                     |
|-------------|-------------------------------------------------------------------------------------------------|
| NTUser      | Die Klasse ermöglicht die Arbeit mit Benutzerkonten unter NT4. Sie kapselt Funktionen von ADSI. |
| IniFile     | Diese Klasse ermöglicht das Lesen und Setzen von Informationen in INI-Dateien.                  |
| CSV         | Diese Klasse ermöglicht das Lesen von CSV-Dateien.                                              |
| ADSI        | Hilfsroutinen zur Arbeit mit der ADSI-Komponente                                                |
| Util        | Diese Klasse realisiert den Zugriff auf die in diesem Buch vorgestellten Standardroutinen.      |

*Tabelle 23.2:  
Klassen der Komponente "WindowsScripting"*

## 23.1 Klasse »IniFile«

Die Klasse IniFile bietet genau zwei Methoden an:

- ▶ ReadFromIni(IniDat, Rubrik, Attribut): Auslesen eines einzelnen Werts in einer INI-Datei
- ▶ WriteToIni(IniDat, Rubrik, Attribut, Wert): Schreiben eines einzelnen Werts in eine INI-Datei

Dabei ist IniDat Pfad und Name einer INI-Datei, Rubrik eine in eckigen Klammern eingeschlossene Rubrik und Attribut der vor dem Gleichheitszeichen stehende Attributname. Nur beim Schreiben muss natürlich ein Wert angegeben werden.

```
[Autor]
Name=Holger Schwichtenberg
eMail=hs@windows-scripting.de
[Website]
URL=www.windows-scripting.com
```

Listing 23.1: Typischer Aufbau einer INI-Datei

```
Const INIDAT = "d:\buch\docs\buch.ini"
Set o = CreateObject("WindowsScripting.IniFile")
o.WriteToIni "Autor", "Name", "Holger Schwichtenberg", INIDAT
o.WriteToIni "Autor", "eMail", "hs@windows-scripting.de", INIDAT
o.WriteToIni "Website", "URL", "www.windows-scripting.com", INIDAT
say o.ReadFromIni("Autor", "Name", INIDAT)
say o.ReadFromIni("Autor", "eMail", INIDAT)
say o.ReadFromIni("WebSite", "URL", INIDAT)
```

Listing 23.2: Erzeugung und Auslesen der obigen INI-Datei [ws\_iniFile.wsf]

## 23.2 Klasse »WinNTUser«

Die Klasse WinNTUser kapselt das Anlegen und Löschen von NT-Benutzerkonten via ADSI.

Tabelle 23.3:  
Methoden der  
COM-Klasse  
WinNTUser

| Methode                                            | Beschreibung                                     |
|----------------------------------------------------|--------------------------------------------------|
| Function About() As String                         | Gibt Informationen über diese Komponente zurück  |
| Function ExistsUser(Username As String) As Boolean | Prüft, ob ein Benutzer in einer Domain existiert |



Klasse »WinNTUser«

| Methode                                                                                                               | Beschreibung                                                                                                                                |
|-----------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------|
| Function CreateNTAccount(Username As String, Password As String, Fullname As String, Description As String) As String | Anlegen eines Benutzers                                                                                                                     |
| Function DeleteNTAccount(Username As String) As String                                                                | Löschen eines Benutzers                                                                                                                     |
| Function UserState(Username As String, state As Boolean) As String                                                    | Ändern des Aktivierungszustands:<br>State=True bedeutet, das Konto kann benutzt werden.<br>State=False bedeutet, das Konto ist deaktiviert. |
| Function ChangePWD(Username As String, password As String) As String                                                  | Änderung des Passworts eines Benutzerkontos                                                                                                 |

Tabelle 23.3:  
Methoden der  
COM-Klasse  
WinNTUser  
(Forts.)

Außerdem besitzt die Klasse WinNTUser ein Attribut mit dem Namen Container. In dieses Attribut muss zu Beginn der Arbeit mit der Klasse der Name der Domain bzw. des Computers geschrieben werden, auf den sich die nachfolgenden Operationen beziehen sollen. Das folgende Skript zeigt, wie die Klasse verwendet werden soll:

```
Dim s ' As String
Dim u 'As WindowsScripting.WinNTUser
Set u = CreateObject("WindowsScripting.WinNTUser")
say u.About
u.Container = "Mars"
s = u.CreateNTAccount("D.Scully", "geheim", "Dana Scully", _
"FBI-Agentin")
say "Benutzer erzeugen: " & s
s = u.ExistsUser("D.Scully")
say "Existiert Benutzer?: " & s
s = u.ChangePWD("D.Scully", "bla")
say "Passwort ändern: " & s
s = u.UserState("D.Scully", False)
say "Benutzer deaktivieren: " & s
s = u.UserState("D.Scully", True)
say "Benutzer aktivieren: " & s
s = u.DeleteNTAccount("D.Scully")
say "Benutzer löschen: " & s
```

Listing 23.3: Verwendung der zu implementierenden Klasse WindowsScripting.  
WinNTUser [ws\_ntuser.wsf]

## 23.3 Klasse »Util«

**Util** Die Klasse Util enthält die im Rahmen dieses Buches besprochenen Standardroutinen.

- ▶ Sub WriteTo(FilePath, Text) hängt einen String an eine Datei an.
- ▶ Function ExistsObject(moniker) prüft auf die Existenz einer COM-Instanz.
- ▶ Function CheckCreate(progid) testet, ob eine angegebene COM-Klasse mit CreateObject() instanziiert werden kann.
- ▶ Function getcol(objcol As Object) liefert einen String mit einer durch Semikola getrennten Liste der Namen der Unterobjekte einer Collection.
- ▶ Function flat(var) macht einen String aus einem Array.
- ▶ Function CSVadd(s1, s2) fügt zwei Strings durch Semikolon getrennt zusammen.
- ▶ Function get\_from\_array(wert, feld) entnimmt aus einem zweidimensionalen Array einen passenden Wert (ermittelt Wert von Spalte1 anhand eines Werts aus Spalte2).
- ▶ Function get\_from\_array\_mult(wert, feld) entnimmt aus einem zweidimensionalen Array alle passenden Werte (ermittelt Werte von Spalte1 anhand eines Werts aus Spalte2).

## 23.4 Klasse »ADSI«

**Klasse ADSI** Die Klasse ADSI enthält die Implementierung der nachfolgenden vier im Kapitel über ADSI (Kapitel 13) vorgestellten Hilfsroutinen:

- ▶ Function ADSFlat(var)
- ▶ Function binGUIDtoSTR(guid)
- ▶ Sub ADSI\_Clear(this)
- ▶ Sub ADSI\_DeleteThis(this)

## 23.5 Klasse »CSV«

Die Klasse bietet Unterstützungsfunktionen beim Auslesen von Comma Separated Value (CSV)-Dateien an. Das Trennzeichen dabei kann aber ein beliebiges Zeichen sein.

**CSV**

CSV steht für *Comma Separated Value*. Eine CSV-Datei speichert Datensätze in Form einer Textdatei ab. Jede Zeile ist ein Datensatz. Innerhalb eines Datensatzes gibt es einen eindeutigen Feldbegrenzer, in der Regel ein Semikolon (»;<«).

*Klasse »CSV«*

- Die Methode `GetField(strLine)` liest aus einem übergebenen String alle Dateien bis zum nächsten Semikolon aus. Auf Basis eines Call by Reference wird die übergebene Variable so manipuliert, dass das ausgelesene Feld abgeschnitten ist.
- Die Methode `Count(strLine)` liefert die Anzahl der Felder.
- Mit dem Attribut `Separator` kann ein anderes Trennzeichen als das standardmäßige Semikolon festgelegt werden.

**Zwei Methoden,  
ein Attribut**

```
s = "Holger:Schwichtenberg:Essen"
Set o = CreateObject("WindowsScripting.CSV")
o.Separator = ":"
say o.Count(s)
' -- Ausgabe aller Felder
Do While s <> ""
 say o.GetField(s)
Loop
```

*Listing 23.4: Testroutine für die CSV-Klasse [ws\_csv.wsf]*



# 24 Microsoft Common Dialog Control

Das *Common Dialog Control* ermöglicht den Zugriff auf Windows-Standarddialoge (Datei öffnen, Datei speichern, Druckerauswahl, Schriftartauswahl, Farbauswahl, Hilfe). Das Common Dialog Control (COMDLG32.OCX) ist ein Satz von ActiveX-Steuerelementen, die den Zugriff auf die Betriebssystem-Bibliothek COMDLG32.DLL kapseln. Im Gegensatz zu vielen anderen ActiveX-Steuerelementen besitzen die Common Dialogs ihre eigenen Fenster – sie benötigen also keinen Form-Container. Daher sind diese Standarddialoge auch aus Skriptumgebungen heraus nutzbar.

Standarddialoge

Ein Common Dialog stellt dem Benutzer lediglich eine Frage – er führt die Aktion jedoch nicht aus. Das aufrufende Programm erhält die Auswahl des Benutzers und kann diese weiterverarbeiten.

|                                        |                                            |
|----------------------------------------|--------------------------------------------|
| <b>Name und Abkürzung</b>              | Microsoft Common Dialog Control            |
| <b>Name der Komponentendatei</b>       | COMDLG32.OCX                               |
| <b>Interner Name der Typbibliothek</b> | MSComDlg                                   |
| <b>Helpstring der Typbibliothek</b>    | Microsoft Standarddialog-Steuerelement 6.0 |
| <b>Hersteller</b>                      | Microsoft                                  |
| <b>Lizenzierung</b>                    | Bestandteil von Visual Studio 6.0          |
| <b>Besprochene Version</b>             | 6.0.84.18                                  |
| <b>NT4 und Windows 95/98/ME</b>        | regsvr32 COMDLG32.OCX                      |
| <b>Windows2000</b>                     | regsvr32 COMDLG32.OCX                      |
| <b>Dokumentation</b>                   | MSDN Library [MSL00]                       |

## 24.1 Objektmodell

Einzigste Klasse ist `MSComDlg.CommonDialog`.

Das Vorgehen ist wie folgt:

- ▶ Instanzieren der Klasse
- ▶ Festlegen der relevanten Attribute für den jeweiligen Standarddialog
- ▶ Anzeige der Dialoge durch Aufruf einer spezifischen Methode für jeden einzelnen Dialog
- ▶ Auslesen der durch den Dialog gesetzten Werte

## Microsoft Common Dialog Control

**Methoden** Die Klasse `CommonDialog` stellt die in der nachfolgenden Tabelle genannten sechs Methoden für jeweils eine Art von Standarddialogen bereit. Die Methoden besitzen weder Parameter noch Rückgabewerte. Alle notwendigen Einstellungen müssen durch Zuweisung an die Attribute vorgenommen werden. Dabei hat jeder der sechs Standarddialoge eigene Attribute. Die Klasse `CommonDialog` speichert alle Rückgabewerte in Attributwerten, die nach der Rückkehr aus der Methode abgefragt werden können.

Tabelle 24.1:  
Methoden der Klasse  
»CommonDialog«

| Methode       | Erläuterung                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ShowOpen()    | Zeigt das Dialogfenster »Öffnen« zur Auswahl einer Datei. Diese Methode kann ohne das vorherige Setzen von Attributwerten aufgerufen werden.                                                                                                                                                                                                                                                                                                                                                                                                        |
| ShowSave()    | Zeigt das Dialogfenster »Speichern« zur Auswahl eines Ordners und Dateinamens. Diese Methode kann ohne das vorherige Setzen von Attributwerten aufgerufen werden.                                                                                                                                                                                                                                                                                                                                                                                   |
| ShowColor()   | Zeigt das Dialogfenster »Farbe« zur Druckerauswahl. Diese Methode kann ohne das vorherige Setzen von Attributwerten aufgerufen werden.                                                                                                                                                                                                                                                                                                                                                                                                              |
| ShowFont()    | Zeigt das Dialogfenster »Schriftart« zur Auswahl von Schriftart, Schriftschnitt und Schriftgröße an. Vor dem Aufruf muss im Attribut <code>Flags</code> festgelegt werden, ob nur Bildschirmschriftarten, nur Druckerschriftarten oder beide Arten angezeigt werden sollen.<br><br>CONST <code>cdlCFScreenFonts</code> = 1<br>CONST <code>cdlCFPrinterFonts</code> = 2<br>CONST <code>cdlCFBoth</code> = 3                                                                                                                                          |
| ShowPrinter() | Ohne diese Angabe erscheint eine Dialogbox, die meldet, es seien keine Schriftarten installiert.<br>Zeigt das Dialogfenster »Drucken« zur Druckerauswahl. Diese Methode kann ohne das vorherige Setzen von Attributwerten aufgerufen werden.                                                                                                                                                                                                                                                                                                        |
| ShowHelp()    | Zeigt eine Hilfedatei an. Es können nur die alten Windows-Hilfedateien (Extension <code>.hlp</code> ), nicht jedoch die neuen kompilierten HTML-Dateien ( <code>.chm</code> ) angezeigt werden. Es müssen vorher im Attribut <code>HelpFile</code> der Dateiname der Hilfedatei und im Attribut <code>HelpCommand</code> die Art der Anzeige näher spezifiziert werden.<br>Ohne diese Angaben scheint nichts zu passieren. Die Anwendung <code>WINHLP32.EXE</code> wird jedoch im Hintergrund gestartet und bleibt als verwaister Prozess bestehen. |

## Microsoft Common Dialog Control



Abbildung 24.1:  
Anzeige der Klasse  
»CommonDialog«  
im VB6-Objekt-  
katalog

## 24.2 Einfaches Beispiel

Das nachfolgende Beispiel zeigt die jeweils schlichteste Form der Dialogfenster.

```
Set objDialog = CreateObject("MSComDlg.CommonDialog")
' --- Dialogfenster ohne weitere Angaben
objDialog.ShowOpen
objDialog.ShowSave
objDialog.ShowColor
objDialog.ShowPrinter
' --- Dialogfenster mit Voraussetzungen
' ----- Schriftarten wählen
objDialog.Flags = cdLCFBoth
objDialog.ShowFont
' ----- Hilfe zu Windows anzeigen
objDialog.HelpFile = "windows.hlp"
objDialog.HelpCommand = cdLHelpContents
objDialog.showHelp
```

Listing 24.1: Aufruf minimal konfigurierter Dialogfenster

Abbildung 24.2:  
Druckerauswahl-  
Dialog



Abbildung 24.3:  
Farbauswahl-Dialog

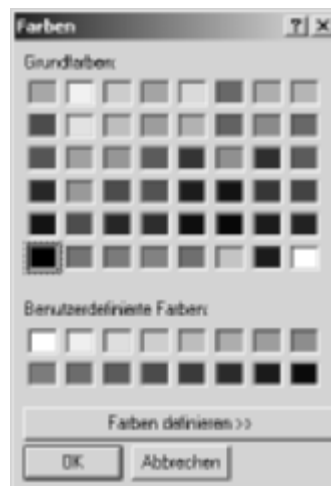
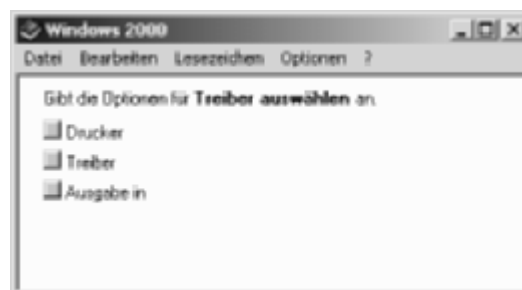


Abbildung 24.4:  
Windows-Hilfe





## 24.3 Öffnen-Dialog

Das nächste Listing zeigt den Öffnen-Dialog ausführlicher. Dabei werden folgende Werte festgelegt:

- ▶ das Startverzeichnis im Attribut `InitDir`
- ▶ zwei Dateitypfilter im Attribut `Filter`. Dafür gibt es eine spezielle Syntax. Zu jedem Filter gehören zwei Angaben: Zum einen die Anzeige im Dialogfeld, davon mit einem vertikalen Strich »|« abgetrennt als zweites das Filterkriterium. Sofern mehrere Extensionen gefiltert werden sollen, müssen diese durch ein Semikolon getrennt sein. Dateifilter werden untereinander wieder durch einen vertikalen Strich abgetrennt.
- ▶ die maximale Länge des Dateinamens (`MaxFileSize`). Die Angabe dieses Attributs ist normalerweise bereits auf 255 Zeichen eingestellt. VBS hat jedoch ein Problem mit dem Standardwert und verlangt eine explizite Zuweisung.
- ▶ mit einer Zuweisung an das Attribut `Flags` wird die Mehrfachauswahl von Dateien zugelassen. Bitte beachten Sie, dass Windows in diesem Fall nicht den modernen Dateidialog im Windows32-Look&Feel, sondern einen Dialog älteren Typs hervorbringt.
- ▶ Da die `Show()`-Methoden der Klasse `CommonDialog` keine Rückgabewerte liefern und es auch kein Attribut gibt, das anzeigt, ob das Dialogfenster vom Anwender abgebrochen wurde, ist es etwas umständlich, den Abbruch festzustellen. Mit `objDialog.CancelError = True` wird das Control angewiesen, im Fall des Abbruchs einen Fehler Nummer 32755 zu erzeugen, der dann im Programm abgefangen werden muss.

```
' === Demonstration des Open-Dialogs
Sub Dialog_open()
Const cd10FNAIAllowMultiselect = 512
Dim objdialog ' As MSComDlg.CommonDialog
Dim strfilter
' --- Instanzieren
Set objdialog = CreateObject("MSComDlg.CommonDialog")
' --- Festlegen des Dateifilters
strfilter = "Textdateien (*.txt;*.csv;*.asc)|*.txt;*.csv;*.asc|"
strfilter = strfilter & "Alle Dateien (*.*)|*.*"
objdialog.Filter = strfilter
' --- Startverzeichnis
objdialog.InitDir = "D:\buch"
' --- Maximale Dateilänge
objdialog.MaxFileSize = 200
' --- Mehrfachauswahl zulassen
objdialog.Flags = cd10FNAIAllowMultiselect
' --- Wirkung von Abbruch festlegen
objdialog.CancelError = True
```

## Microsoft Common Dialog Control

```
' --- Anzeigen
On Error Resume Next
objdialog.ShowOpen
say "Ausgewählt wurde: " & objdialog.FileName
' --- Fehlerabfrage
If Err.Number = 32755 Then
 say "Auswahl wurde abgebrochen!"
ElseIf Err.Number <> 0 Then
 say "Es ist folgender Fehler aufgetreten:" & Err.Description
Else
 say "Ausgewählt wurde: " & objdialog.FileName
End If
```

End Sub

Listing 24.2: Anzeige eines Öffnen-Dialogs [dialog\_open.wsf]

Abbildung 24.5:  
Öffnen-Dialog ohne  
Mehrfachauswahl

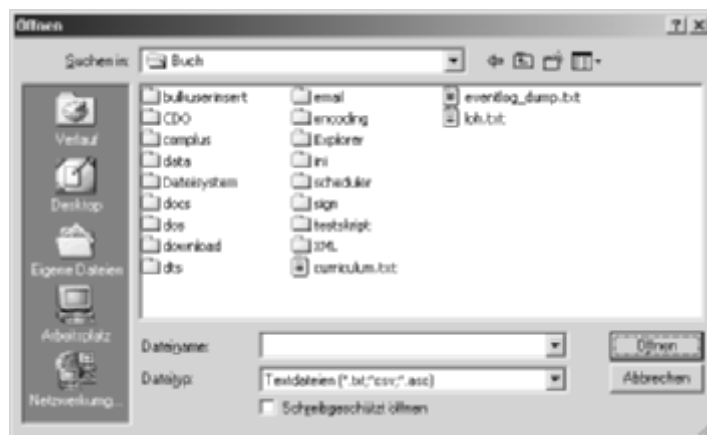
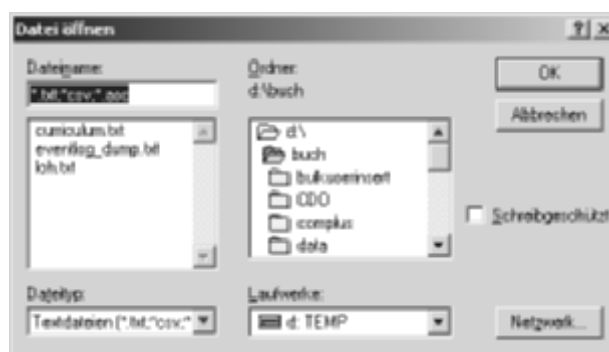


Abbildung 24.6:  
Öffnen-Dialog mit  
Mehrfachauswahl



### Rückgabewerte

Das Ergebnis der Auswahl steht nach der Rückkehr im Attribut `FileName`. Wenn der Benutzer eine Mehrfachauswahl hatte, dann stehen die Dateinamen alle in diesem Attribut, jeweils getrennt durch ein Leerzeichen. Dies führt natürlich dann zu einem Problem, wenn Dateinamen ausgewählt wurden, die Leerzeichen enthalten. Aus diesem Grund zeigt der »alte« Standarddialog mit Mehrfachauswahl zwar lange Dateinamen mit mehr als acht Zeichen korrekt an, jedoch nicht solche mit Leerzeichen. Diese werden durch die übliche Tilde ~ abgekürzt, die Sie kennen, wenn Sie mit alten Windows-16-Bit-Anwendungen auf einem Dateisystem eines 32-Bit-Betriebssystems arbeiten.

Das Attribut `FileName` ist auch dann belegt, wenn der Benutzer zwar einen Dateinamen ausgewählt, dann aber auf Abbruch geklickt hat. Ein leeres `FileName`-Attribut ist also kein hinreichendes Kriterium für die Feststellung des Dialogabbruchs. Dies muss mit einem Test auf Fehler 32755 festgestellt werden.

Das Attribut `FileTitle` enthält den Namen der ausgewählten Datei ohne Pfad.

## 24.4 Speichern-Dialog

Das Beispiel des Speichern-Dialogs sieht sehr ähnlich wie die Anwendung des Öffnen-Dialogs aus. Neu ist das Attribut `FilterIndex`, das festlegt, welcher der angegebenen Dateifilter zu Beginn gewählt ist.

Datei speichern  
unter

```
Const intOverwritePrompt = &H2&
Const intPathMustExist = &H800&
' -- Instanzieren
Set objDialog = CreateObject("MSComDlg.CommonDialog")
' -- Festlegen des Dateifilters
strfilter = "HTML (*.htm;*.html)|*.htm;*.html|"
strfilter = "ASP-Dateien (*.asp)|*.asp|"
strfilter = strfilter & "Alle Dateien (*.*)|*.*"
' Vorbereitung der Anzeige
With objDialog
' -- Startverzeichnis
.InitDir = "c:\\"
' -- Flags
.Flags = intOverwritePrompt + intPathMustExist
' -- Dateityp-Filter auf ersten Filter setzen
.Filter = strfilter
.FilterIndex = 1
' -- Namensvorschlag
.FileName = "neueDatei"
' -- Maximale Dateilänge
```

## Microsoft Common Dialog Control

```
.MaxFileSize = 255
' -- Wirkung von Abbruch festlegen
.CancelError = True ' triggert einen Fehler im Fall des Abbruchs!
On Error Resume Next
' -- Anzeigen
.ShowSave
End With
' Auswertung des Ergebnisses
If Err.Number = 32755 Then
 MsgBox "Auswahl wurde abgebrochen!"
ElseIf Err.Number <> 0 Then
 MsgBox "Es ist folgender Fehler aufgetreten:" & Err.Description
Else
 MsgBox "Ausgewählt wurde: " & objDialog.FileName
End If
```

Listing 24.3: Anzeige eines Speicher-Dialogs mit Filter [dialog\_save.vsf]

Abbildung 24.7:  
Speichern-Dialog



## 24.5 Schriftarten-Dialog

### Schriftart wählen

Da sich diese Beispiele sehr ähnlich waren, zeigt das letzte Beispiel den Font-Dialog, der zahlreiche, jedoch intuitiv verständliche Ein- und Ausgabeattribute hat.

```
' *** FontsConstants
Const cd1CFScreenFonts = 1
Const cd1CFPrinterFonts = 2
Const cd1CFBoth = 3
Dim objdialog, strfilter
' -- Instanziieren
```

### Microsoft Common Dialog Control

```
Set objdialog = CreateObject("MSComDlg.CommonDialog")
With objdialog
' -- Vorbereitung der Anzeige
.Flags = cd1CFBoth
.FontName = "Arial"
.FontSize = 12
.FontItalic = True
.FontStrikethru = True
On Error Resume Next
' -- Anzeige
.ShowFont
' Auswertung des Ergebnisses
If Err.Number = 32755 Then
 MsgBox "Auswahl wurde abgebrochen!"
ElseIf Err.Number <> 0 Then
 MsgBox "Es ist folgender Fehler aufgetreten:" & Err.Description
Else
 MsgBox "Ausgewählt wurde: " & .FontName & "(" & .FontSize & _
 " Punkt)"
End If
End With
```

Listing 24.4: Anzeige des Schriftarten-Dialogs [dialog\_font.wsf]

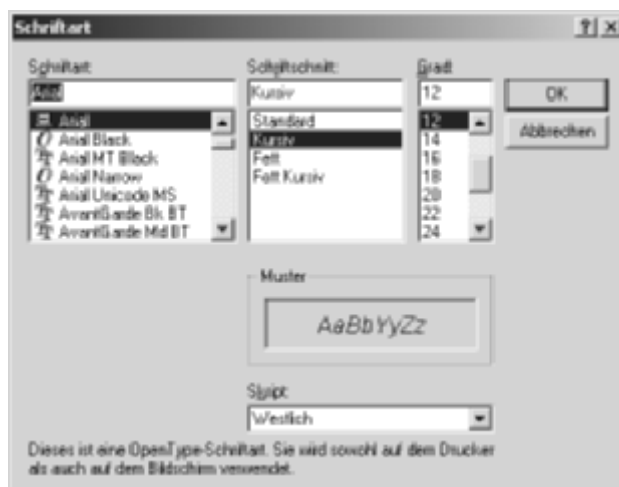


Abbildung 24.8:  
Schriftarten-Dialog



# 25 Microsoft Operations Manager-Hilfsobjekte (OnePoint Utility Objects)

Zusammen mit dem Microsoft Operations Manager (MOM) 2000 liefert Microsoft drei scriptingfähige COM-Komponenten aus, die auch außerhalb des MOM verwendet werden können. Die Komponenten haben nicht den Begriff »Microsoft«, sondern den Begriff »OnePoint« im Namen, da sie (wie auch der ganze MOM selbst) von der Firma NetIQ entwickelt wurden. Microsoft hat den MOM von NetIQ gekauft.

Hilfskomponenten für den MOM

Die drei Komponenten sind:

- ▶ *OnePoint Utility Objects for Scripting*
- ▶ *OnePoint Utility Objects for Active Directory*
- ▶ *OnePoint Utility Objects for Exchange*

## Gemeinsamkeiten

Allen Klassen in diesen Komponenten ist gemein,

**GetError()**

- ▶ dass sie eine *GetError()*-Methode besitzen, die die Fehlerbeschreibung des letzten in dem Objekt aufgetretenen Fehlers liefern
- ▶ dass die Methode nur ein *True* oder *False* zurückliefert
- ▶ dass die Rückgabewerte in Form von Referenzparametern (»Call by Reference«) übermittelt werden.

## 25.1 OnePoint Utility Objects for Scripting

Diese Komponente ist eine bunte Sammlung von COM-Klassen rund um das Systemmanagement.

|                                        |                                        |
|----------------------------------------|----------------------------------------|
| <b>Name und Abkürzung</b>              | MOM Scripting Helper Objects           |
| <b>Name der Komponentendatei</b>       | OpScUtil.dll                           |
| <b>Interner Name der Typbibliothek</b> | OpScrUtil                              |
| <b>Helpstring der Typbibliothek</b>    | OnePoint Utility Objects for Scripting |
| <b>Hersteller</b>                      | NetIQ (vertrieben durch Microsoft)     |

Tabelle 25.1:  
Schnellinfo  
»OnePoint Utility  
Objects for  
Scripting«

## Microsoft Operations Manager-Hilfsobjekte (OnePoint Utility Objects)

Tabelle 25.1:  
Schnellinfo  
»OnePoint Utility  
Objects for  
Scripting«  
(Forts.)

|                               |                                                                                  |
|-------------------------------|----------------------------------------------------------------------------------|
| <b>Lizenzierung</b>           | Teil des Microsoft Operations Manager (MOM)                                      |
| <b>Besprochene Version</b>    | 4.0.0.12                                                                         |
| <b>Alle Windows-Versionen</b> | Durch Installation des MOM 2000 oder der MOM 2000-Client-Werkzeuge               |
| <b>Dokumentation</b>          | OPERATIONS.CHM<br>[CD:/install/Weitere Informationen/<br>MOM2000/operations.chm] |

Abbildung 25.1:  
Ansicht der  
Typbibliothek im  
Objektkatalog



Tabelle 25.2:  
Überblick über die  
Klassen

| Klassenname   | Funktionsumfang                                                                                               |
|---------------|---------------------------------------------------------------------------------------------------------------|
| Inet          | HTTP-Get, FTP-Get und Ping                                                                                    |
| SystemManager | Basisinformation über das Betriebssystem und Funktionen des Task-Managers (Neustart, Herunterfahren, Sperren) |
| EventLog      | Sichern und Leeren eines NT-Ereignisprotokolls                                                                |
| PSAPI         | Prozessliste einlesen, Prozess stoppen                                                                        |
| Utility       | Verschiedene Hilfsfunktionen (z.B. Prozesse starten, Umgebungsvariablen, Protokollierung, Skript anhalten)    |
| SCManager     | Verwaltung von NT-Diensten: Zustand auslesen und Zustand ändern                                               |
| TermServer    | Ermitteln des Benutzers einer Terminalserveransicht                                                           |
| PagerJobQueue | Übermitteln einer Pager-Nachricht via Modem                                                                   |



### 25.1.1 SCManager-Klasse

Diese Klasse bietet Methoden zur Kontrolle über NT-Dienste:

**Dienstverwaltung**

- ▶ obj.BounceService(ServiceName String, [ComputerName String], [Arguments String])
- ▶ obj.ContinueService (ServiceName String, [ComputerName String])
- ▶ obj.GetServiceConfig(ServiceName String, Request String, OutInfo Variant, [ComputerName String])
- ▶ obj.GetServiceStatus(ServiceName String, ServiceStatus String, [ServiceStatusNumber Variant] [ComputerName String])
- ▶ obj.PauseService (ServiceName String, [ComputerName String])
- ▶ obj.StartService(ServiceName String, [ComputerName String], [Arguments String])
- ▶ obj.StopService(ServiceName String, [ComputerName String])

#### Beispiel mit BounceService()

Interessant in dieser Klasse ist BounceService(). Diese Methode beendet einen Dienst und startet ihn neu. Wenn der Dienst schon beendet war, wird er einfach gestartet. Diese Methode ist wertvoll, wenn Dienste »hängen«.

**Dienste neu  
starten**

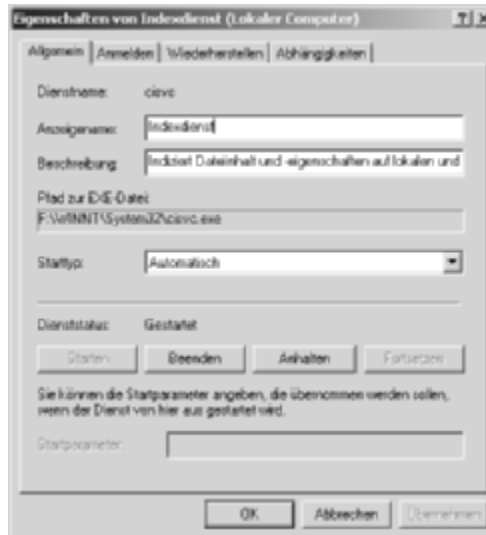
```
' === Neustart eines Dienstes
Sub mom_service()
Set obj = CreateObject("OpScrUtil.SCManager")
Status = obj.BounceService("cisvc", "")
If Status Then
 say "Dienst erfolgreich gestartet!"
Else
 say "Fehler beim Neustart: " & obj.GetError()
End If
End Sub
```

*Listing 25.1: [mom\_service.wsf]*

Wichtig ist, dass Sie nicht den deutschen Anzeigenamen des Dienstes, sondern den internen Namen angeben. Diese Information erhalten Sie im Dienstmanager.

**Dienstname**

Abbildung 25.2:  
Eigenschaften eines  
Dienstes im  
Windows2000-  
MMC-Snap-In  
»Dienste«



## 25.1.2 EventLog

Diese Klasse bietet die folgenden drei selbsterklärenden Methoden zur Verwaltung von NT-Ereignisprotokollen:

- ▶ BackupEventLog(EventLogName string, OutputFolderName string, [MachineName string])
- ▶ ClearEventLog(EventLogName string, OutputFolderName string, [MachineName string])
- ▶ GetNumberOfEventLogRecords(EventLogName string, NumRecords variant, [MachineName string])

### Beispiel

#### Arbeit mit dem Ereignisprotokoll

Das Beispiel liest die Anzahl der Protokolleinträge aus und erstellt danach ein Backup des Logs in eine Datei. Den Dateinamen vergibt die Klasse. Die Methode BackupEventLog() liefert im dritten Parameter als Referenzparameter den vergebenen Namen zurück.

```
' == EventLog
Sub mom_log()
Const RECHNER = "mars"
Const LOG = "Application"
Dim Count
Dim name

Dim obj ' As OpScrUtil.EventLog
Set obj = CreateObject("OpScrUtil.EventLog")
```

### *OnePoint Utility Objects for Scripting*

```
' --- Anzahl auslesen
Status = obj.GetNumberOfEventLogRecords(LOG, Count, RECHNER)
If Status Then
 say "Anzahl der Protokolleinträge: " & Count
Else
 say "Protokoll nicht auslesbar: " & obj.GetError
End If

' --- Sichern
Status = obj.BackupEventLog(LOG, "c:\", name, RECHNER)
If Status Then
 say "Backup erfolgreich gespeichert in " & name
Else
 say "Backup nicht erfolgreich: " & obj.GetError
End If

' --- OPTIONAL: Sichern + Leeren
' Status = obj.ClearEventLog(LOG, "c:\", name & "x", RECHNER)
' If Status Then
' say "Log gesäubert"
' Else
' say "Log nicht gesäubert: " & obj.GetError
' End If
End Sub
```

*Listing 25.2: [mom\_log.wsf]*

## **25.1.3 SystemManager-Klasse**

Diese Klasse liefert Informationen über das Betriebssystem und ermöglicht das Herunterfahren und Neustarten eines Rechners sowie die Sperrung der Arbeitsstation.

**Betriebssystem-  
Informationen**

### **Attribute**

- ▶ OS
- ▶ OSMajorVersion
- ▶ OSMinorVersion
- ▶ OSBuildNumber
- ▶ Domain
- ▶ LoggedOnUsers
- ▶ ServicePackInfo

#### Methoden (alle ohne Parameter)

- ▶ ForceLogoff()
- ▶ ForceReboot()
- ▶ ForceShutdown()
- ▶ LockWorkstation()
- ▶ RequestLogoff()
- ▶ RequestReboot()
- ▶ RequestShutdown()

#### Vergleich mit WMI

Diese Funktionalität bietet auch WMI. Die Installation dieser Klasse ist aber von Vorteil, wenn die Installation des umfangreichen WMI nicht erwünscht ist. Allerdings kann diese Klasse nur lokal agieren.

#### Beispiel

```
' == liefert Informationen über das Betriebssystem
Sub mom_sysinfo()
Dim obj ' As OpScrUtil.SystemManager
Set obj = CreateObject("OpScrUtil.SystemManager")
say obj.OS
say obj.OSMajorVersion
say obj.OSMinorVersion
say obj.OSBuildNumber
say obj.DOMAIN
say obj.LoggedOnUsers
say obj.ServicePackInfo
End Sub
```

Listing 25.3: [mom\_sysinfo.wsf]

### 25.1.4 INet-Klasse

#### TCP/IP

Die Klasse bietet neben `GetError()` drei weitere Methoden aus dem Bereich der Internet-Protokolle:

##### ▶ Laden einer Datei per FTP

```
obj.GetFTPFile(Host String, RemoteFile String, DestinationLocalFile
String, [AsciiText Boolean], [PortNumber Long], [LogonUserID
String],
[LogonPassword String], [AccessType Long])
```

##### ▶ Laden einer Datei per HTTP

```
obj.GetHTTPFile(URL String, [Data Variant],
[BytesRead Long], [AccessType Long])
```

► **Ausführen eines Ping**

*obj.Ping(Address String, Data String, RoundTripMs Long)*

**Beispiel für Ping()**

```
' === Ausführung eines Ping mit der Ping()-Methode
Sub Mom_ping1()
Dim obj ' As OpScrUtil.INet
Dim Status

Set obj = CreateObject("OpScrUtil.INet")
' --- Ping starten
Status = obj.Ping("www.windows-scripting.de", Data, ms)

' --- Auswertung
If Status Then
 say "Rechner erreichbar!"
Else
 say "Fehler bei der Ausführung von ping(): " & _ obj.GetError()
End If
End Sub
```

*Listing 25.4: [mom\_ping1.wsf]*

## **25.1.5 Utility-Klasse**

Die Utility-Klasse ist eine bunte Mischung von Funktionen, die vergleichbar sind mit den Funktionen der WSH Runtime-Komponente. Jedoch sind die von NetIQ implementierten Methoden zum Teil besser als die von Microsoft.

► **Externe Anwendungen starten**

*CreateProcess(ExePath String, CommandLine String,  
Wait Boolean, [Timeout Long],  
[ForceTermination Boolean], [ExitCode Long])*

► **Zugriff auf Umgebungsvariablen**

*GetEnviron(EnvironmentVariable String)  
ExpandEnvironmentStrings(Instring string)*

► **Schreiben eines Eintrags in das Anwendungsprotokoll**

*LogEvent(Description String, [Severity Variant])*

► **das Skript in eine Warteschleife setzen**

*Sleep(Seconds Long)  
SleepMs(MilliSeconds Number)*

### Beispiel für CreateProcess()

Das Beispiel zeigt die Anwendung der CreateProcess()-Methode zur Ausführung eines Ping.

```
' === Ausführung eines Ping mit ping.exe
Sub mom_ping2()
' --- Rückgabewerte
Dim ExitCode
Dim Status
Dim bForceTerminate

Set Util = CreateObject("OpScrUtil.Utility")

' --- Ping starten
Status = Util.CreateProcess("", "ping www.windows-scripting.com",
True, 60, bForceTerminate, ExitCode)

' --- Auswertung
If Not Status Then
 say "Prozess konnte nicht erzeugt werden."
Else
 If bForceTerminate Then say "Ping wurde vorzeitig beendet!"
 If ExitCode = 0 Then
 say "Ping erfolgreich!"
 Else
 say "Ping nicht erfolgreich: " & ExitCode
 End If
End If
strMsg = "The exit code was " & ExitCode
End Sub
```

Listing 25.5: [mom\_ping.wsf]

## 25.2 OnePoint Utility Objects for Active Directory 1.0

Diese Komponente enthält genau eine Klasse (McActiveDir.ActiveDirectory) mit zahlreichen Funktionen zur Verwaltung des Windows2000-Active Directory.

Tabelle 25.3:  
Schnellinfo  
»OnePoint Utility  
Objects for  
Scripting«

|                                            |                                                      |
|--------------------------------------------|------------------------------------------------------|
| <b>Name und Abkürzung</b>                  | MOM Scripting Helper Objects<br>for Active Directory |
| <b>Name der Komponentendatei</b>           | OOMADs.dll                                           |
| <b>Interner Name der<br/>Typbibliothek</b> | OOMADSLib                                            |

# OnePoint Utility Objects for Active Directory 1.0

|                                            |                                                                                  |
|--------------------------------------------|----------------------------------------------------------------------------------|
| <b>Helfstring der Typbibliothek</b>        | OnePoint Utility Objects for Active Directory 1.0                                |
| <b>Abweichender Name der Typbibliothek</b> | McActiveDir.ActiveDirectory                                                      |
| <b>Hersteller</b>                          | NetIQ (vertrieben durch Microsoft)                                               |
| <b>Lizenzierung</b>                        | Teil des Microsoft Operations Manager (MOM)                                      |
| <b>Besprochene Version</b>                 | 4.0.0.12                                                                         |
| <b>Alle Windows-Versionen</b>              | Durch Installation des MOM 2000 oder der MOM 2000-Client-Werkzeuge               |
| <b>Dokumentation</b>                       | OPERATIONS.CHM<br>[CD:/install/Weitere Informationen/<br>MOM2000/operations.chm] |

Tabelle 25.3:  
Schnellinfo  
»OnePoint Utility  
Objects for  
Scripting«  
(Forts.)



Abbildung 25.3:  
Ansicht der Typ-  
bibliothek im VB6-  
Objektkatalog

**Attribute der Klasse »ActiveDirectory«**

- Attribute**
- ▶ BindCount
  - ▶ BindLast
  - ▶ BindTotal
  - ▶ Domain
  - ▶ DomainNamingMaster
  - ▶ DomainNamingMasterBind
  - ▶ GlobalCatalogSearchTime
  - ▶ InfrastructureMaster
  - ▶ InfrastructureMasterBind
  - ▶ LastError
  - ▶ LostAndFoundCount
  - ▶ PDCMaster
  - ▶ PDCMasterBind
  - ▶ Password
  - ▶ RIDMaster
  - ▶ RIDMasterBind
  - ▶ SchemaMaster
  - ▶ SchemaMasterBind
  - ▶ Server
  - ▶ UserName

**Methoden der Klasse »ActiveDirectory«  
(ohne Angabe der Parameter)**

- Methoden**
- ▶ BindDomainNamingMaster()
  - ▶ BindInfrastructureMaster()
  - ▶ BindLostFoundContainer()
  - ▶ BindObject()
  - ▶ BindPDCMaster()
  - ▶ BindRIDMaster()
  - ▶ BindSchemaMaster()
  - ▶ GetDatabaseInfo()
  - ▶ GetLogFileInfo()
  - ▶ NextDomainController()



- ▶ NextGlobalCatalog()
- ▶ NextReplicationPartner()
- ▶ RefreshDomainControllers()
- ▶ RefreshGlobalCatalogs()
- ▶ RefreshReplicationPartners()
- ▶ ReplicationCheckDomainController()
- ▶ ReplicationLastModifyInfo()
- ▶ ReplicationModifyObject()
- ▶ SearchGlobalCatalog()

## 25.3 OnePoint Utility Objects for Exchange

Diese Komponente bietet eine wichtige Funktion, die man in allen Microsoft-Automatisierungskomponenten vergeblich sucht: Mit dieser Komponente ist man in der Lage zu ermitteln, wie groß die einzelnen Postfächer und öffentlichen Ordner auf einem Exchange Server sind.

Diese Komponente funktioniert sowohl mit dem Exchange Server 5.5 als auch mit dem Exchange Server 2000. Die Ermittlung der Daten ist sehr schnell (genauso schnell wie die Anzeige im Exchange 5.5-Administrator bzw. im Exchange 2000-System-Manager).



|                                        |                                             |
|----------------------------------------|---------------------------------------------|
| <b>Name und Abkürzung</b>              | MOM Scripting Helper Objects for Exchange   |
| <b>Name der Komponentendatei</b>       | McExchDG.dll                                |
| <b>Interner Name der Typbibliothek</b> | McExchg                                     |
| <b>Helpstring der Typbibliothek</b>    | keiner                                      |
| <b>Hersteller</b>                      | NetIQ (vertrieben durch Microsoft)          |
| <b>Lizenzierung</b>                    | Teil des Microsoft Operations Manager (MOM) |
| <b>Besprochene Version</b>             | 4.0.0.12                                    |

Tabelle 25.4:  
Schnellinfo »One-  
Point Utility  
Objects for  
Exchange«

Tabelle 25.4:  
Schnellinfo  
»OnePoint  
Utility Objects  
for Exchange«  
(Forts.)

|                               |                                                                                                                                                                                                                               |
|-------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Alle Windows-Versionen</b> | <p>Einzelregistrierung der DLL<br/>regsvr32 McExchDG.dll</p> <p>Außerdem muss die Datei McEXCH.DLL im System-Verzeichnis abgelegt werden. Dies ist keine COM-DLL, so dass eine Registrierung nicht möglich und nötig ist.</p> |
| <b>Dokumentation</b>          | <p>OPERATIONS.CHM<br/>[CD:/install/Weitere Informationen/<br/>MOM2000/operations.chm]</p>                                                                                                                                     |

Tabelle 25.5:  
Klassen in der  
McExchDG.dll

| Klassenname          | Funktionsumfang                                                                                                   |
|----------------------|-------------------------------------------------------------------------------------------------------------------|
| MTALogAnalyzer       | Bietet Methoden zur Analyse des Message Transfer Agent-Protokolls                                                 |
| PublicFolderAnalyzer | Bietet eine Methode, die die Größe in Megabytes und die Anzahl der Nachrichten aller öffentlichen Ordner ausliest |
| MailboxAnalyzer      | Bietet eine Methode, die die Größe in Megabytes und die Anzahl der Nachrichten aller Mailboxen ausliest           |

### 25.3.1 Klasse »MailboxAnalyzer«

#### GatherMailboxData()

Diese Klasse liefert Informationen über die Postfächer eines Microsoft Exchange Servers.

Die Methode

GatherMailboxData(*DataOut* Variant, *RowCount* Long, [*TopNN* Long])

liefert ein zweidimensionales Array.

Während die zweite Dimension die einzelnen Postfächer unterscheidet und prinzipiell unendlich groß sein kann, besteht die erste Dimension aus drei Feldern:

- Feld 0 enthält den Postfachnamen
- Feld 1 enthält den Füllstand in MB (als Fließkommazahl)
- Feld 2 enthält die Anzahl der Nachrichten in dem Postfach

#### Beispiel

Das folgende Skript liefert eine CSV-Liste der einhundert am meisten gefüllten Postfächer. Vor dem Aufruf von `GatherMailboxData()` muss zunächst mit

`LogonToMAPI(Server String, Mailbox String)`

eine Verbindung zum Server aufgebaut werden.

### OnePoint Utility Objects for Exchange

```
' === Ermittelt den Füllstand der Postfächer
' auf einem Exchange Server
Sub mom_mailbox_fuellstand()
Dim data()
Dim lngrowcount
Set obj = CreateObject("McExchg.MailboxAnalyzer")
' --- Anmeldung
Status = obj.LogonToMAPI("Sonne2000", "hs")
If Not Status Then
 say "Fehler: " & obj.GetError()
Else
 ' --- Lies die vollsten 100 Postfächer
 Status = obj.GatherMailboxData(data, Count, 100)
 If Not Status Then
 say "Fehler: " & obj.GetError()
 Else
 ' --- Array ausgeben

 For i = 0 To Count - 1
 mailbox = data(0, i)
 KB = Int(data(1, i) * 1000)
 Anzahl = data(2, i)
 say mailbox & ":" & KB & ":" & Anzahl
 Next
 End If
End If
End Sub
```

Listing 25.6: [mom\_mailbox\_fuellstand.wsfl]

#### Fehlermeldungen

Die Fehlermeldungen, die `GetError()` liefert, sind überraschend ausführlich, z.B.

#### Fehlermeldungen

*Fehler: Could not access mailbox information: -2147467259 (0x80004005)  
Unbekannter Fehler*

*Accessing Exchange mailbox information requires administrative access to the Exchange server.*

*Ensure that the Windows NT account hs has admin permissions on the Exchange server Sonne2000 and that the account hs has permission to access mailbox bh.*

Die Meldung »Datei nicht gefunden: mcexch.dll« weist darauf hin, dass Sie die Datei MCEXCH.DLL in das System-Verzeichnis des Rechners kopieren müssen, auf dem Sie diese Routine ausführen.



### 25.3.2 Klasse »PublicFolderAnalyzer«

Die Klasse `PublicFolderAnalyzer` liest den Füllstand aller öffentlichen Ordner eines Exchange Servers aus. Die Klasse arbeitet analog zur Klasse `MailboxAnalyzer`.

Sie bietet die Methoden

- ▶ `GetError()`
- ▶ `LogonToMAPI(Server String, Mailbox String)`
- ▶ `GatherPublicFolderData(DataOut Variant, RowCount Long, [TopNN Long])`

Das Beispiel sieht bis auf den geänderten Klassennamen und den geänderten Methodennamen genauso aus wie das Beispiel im letzten Kapitel.

Abbildung 25.4:  
Ansicht des Füll-  
stands der öffent-  
lichen Ordner im  
System-Manager  
des Exchange  
Server 2000

[illegible]

## Einschränkung



Diese Klasse liefert beim Exchange Server 2000 leider nur die Ordner der Standardhierarchie der öffentlichen Ordner. Selbstangelegte Speichergruppen bleiben unberücksichtigt. Dies ist allerdings keine Beschränkung der Komponente, sondern eine Beschränkung von MAPI.

### Beispiel

```
' === Ermittelt den Füllstand der öffentlichen Ordner auf einem
Exchange
```

Server

Sub mom folder fuellstand()

```
Dim data()
```

Dim lngrowcount

### *OnePoint Utility Objects for Exchange*

```
Dim Ordner
Set obj = CreateObject("McExchg.PublicFolderAnalyzer")
' --- Anmeldung
Status = obj.LogonToMAPI("Sonne2000", "hs")
If Not Status Then
 say "Fehler: " & obj.GetError()
Else
 ' --- Lese die vollsten 100 Postfächer
 Status = obj.GatherPublicFolderData(data, Count, 100)
 If Not Status Then
 say "Fehler: " & obj.GetError()
 Else
 ' --- Array ausgeben

 For i = 0 To Count - 1
 Ordner = data(0, i)
 KB = Int(data(1, i) * 1000)
 Anzahl = data(2, i)
 say Ordner & ":" & KB & ":" & Anzahl
 Next
 End If
End If
End Sub
```

*Listing 25.7: Ermitteln des Füllstands öffentlicher Ordner  
[mom\_folder\_fuellstand.wsf]*



# 26 Microsoft Office-Komponenten

Die Kernprodukte der Microsoft Office-Produktfamilie (Word, Excel, Access, PowerPoint, FrontPage, Outlook) sind komplett über COM-Komponenten automatisierbar. In Office 2000 gibt es insgesamt rund 600 Klassen. Diese Komponenten sind gut dokumentiert in der MSDN Library [MSL00]. Die Dokumentation enthält auch grafische Darstellungen der Objektmodelle. Sie erhalten auch Hilfe zu den Objektmodellen, wenn Sie innerhalb der VBA-Entwicklungsumgebung der jeweiligen Anwendung die Hilfe aufrufen.

```
Set wo = CreateObject("Word.Application")
wo.Visible = False
wo.Documents.Open "d:\buch\docs\test.doc"
wo.ActiveDocument.PrintOut
wo.Quit
```

*Listing 26.1: Dieses Skript druckt ein Word-Dokument aus, ohne das Word-Fenster sichtbar zu machen. [office\_word.wsf]*

Die MS Office-Komponenten sind implementiert in den jeweiligen .EXE-Dateien der Anwendungen. Die Typbibliotheken sind jeweils extern gespeichert in Dateien mit der Extension .OLB, z.B. EXCEL9.OLB, MSWORD9.OLB, GRAPH9.OLB, MSPPT9.OLB, MSOUTL9.OLB.

**Implementierung  
und  
Typbibliotheken**





# 27 Komponenten- erforschung

Dieses Buch hat Ihnen zahlreiche COM-Komponenten vorgestellt, die Ihnen den Zugriff auf viele Funktionen des Betriebssystems und von (Server-)Anwendungen ermöglichen.

Hilfe zur  
Selbsthilfe

Zwar haben Sie nach der Lektüre dieses Buches ein fundiertes Grundwissen über verschiedene Komponententypen und verschiedene Objektmodelle, dieses Buch kann jedoch unmöglich eine vollständige Referenz aller Komponenten sein. Sie kommen also nicht umhin, in Zukunft selbst Komponenten zu erforschen. Dieses Kapitel stellt Ihnen ein Vorgehensmodell zur Suche nach und Analyse von Komponenten vor.

## 27.1 Suche nach Komponenten

Es ist falsch, erst auf die Suche nach einer Komponente zu gehen, wenn Sie eine konkrete Herausforderung haben. Sowohl als Programmierer als auch als Administrator sollten Sie sich regelmäßig darüber informieren, was die bei Ihnen vorhandenen Systeme und der Markt insgesamt an Komponenten hergeben. Sie werden feststellen, dass Ihnen dabei Komponenten in die Hände fallen, die Ihre Arbeit an Stellen vereinfachen, an die Sie bisher gar nicht zu denken gewagt haben.

### 27.1.1 Lokale Suche

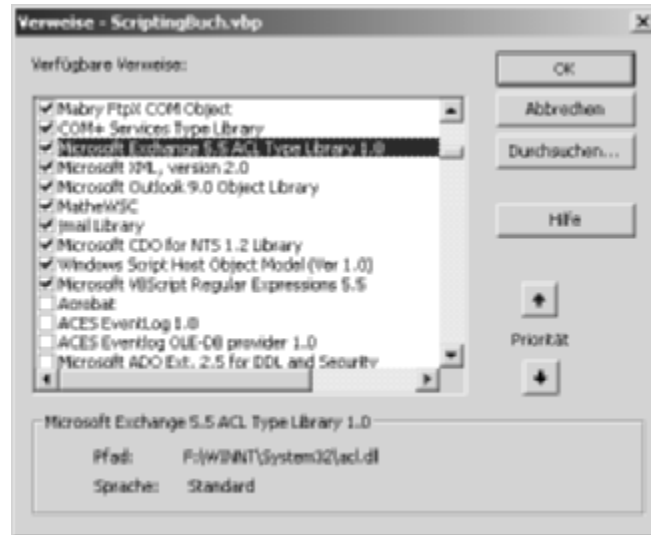
Der erste und wichtigste Ort, den Sie nach einer hilfreichen Komponente durchsuchen sollten, sind Ihre eigenen Windows-Installationen. Machen Sie sich damit vertraut, welche Komponenten auf Ihren Systemen installiert sind. Betrachten Sie keineswegs nur Ihre eigene lokale Workstation, sondern schauen Sie sich Ihre Server an und gehen Sie zu anderen Arbeitsplätzen, auf denen andere Software installiert ist.

Suche auf Ihren  
eigenen Systemen

Leider gibt es noch kein umfassendes Komponentenrepository, das Sie allumfassend über die installierten Komponenten informiert. Sie müssen damit leben, die Kombination mehrerer Werkzeuge einzusetzen, um sich über die installierten Komponenten zu informieren.

Der erste Suchplatz ist der VERWEISE-Dialog einer Entwicklungsumgebung wie Visual Basic oder VBA. Dort erhalten Sie eine Liste der installierten Typbibliotheken anhand ihrer Helpstrings. Die Helpstrings vermitteln in der Regel einen recht guten Eindruck davon, welche Aufgaben eine Komponente erfüllt. Es gibt aber auch genügend Fälle, wo der Helpstring keinen für sich selbst sprechenden Text enthält (z.B. werden Sie auch Einträge wie »vs« oder »tom« finden).

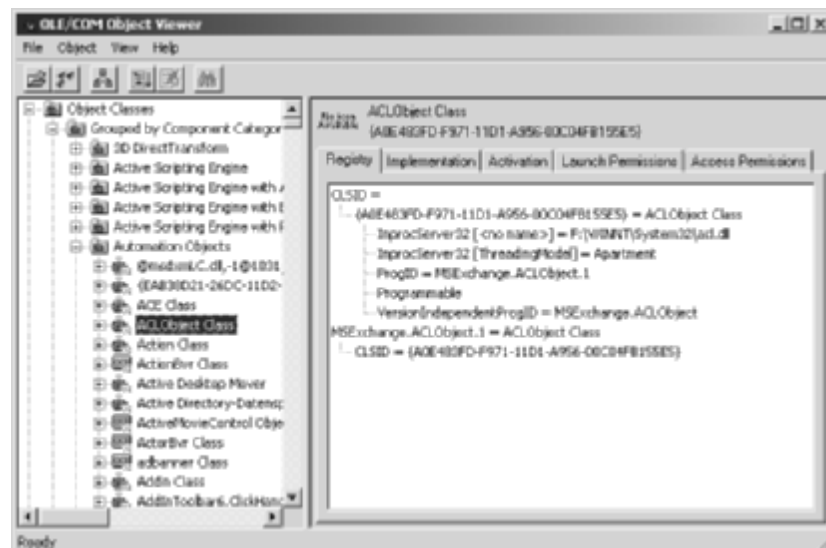
Abbildung 27.1:  
Dialog »Verweise«  
in der Visual Basic  
6.0-IDE



#### Automations- fähige Klassen

Der nächste Ansatzpunkt ist eine Liste der installierten automationsfähigen Klassen. Dafür sollten Sie sowohl den *Microsoft COM-Viewer* (Komponentenkategorie AUTOMATION OBJECT) als auch das Shareware-Werkzeug *COM-Explorer* zu Rate ziehen, da beide Auflistungen zwar unvollständig, aber keineswegs an den gleichen Stellen unvollständig sind. Im COM-Explorer sollten Sie auch die Möglichkeit nutzen, die Klassen nach Typbibliotheken sortieren zu lassen. Damit erhalten Sie wichtige Erkenntnisse über den Zusammenhang der Automationsklassen zu den im VERWEISE-Dialog gezeigten Typbibliotheken.

Abbildung 27.2:  
Automationsfähige  
Klassen im  
COM-Viewer  
von Microsoft



### Suche nach Komponenten

Da die Auflistung der automationsfähigen Klassen in beiden Tools unzureichend ist, sollten Sie im dritten Schritt die Liste ALL OBJECTS im COM-Viewer aufklappen. Schauen Sie sich bei Ihren interessanten Funden aber zuerst die unterstützten Schnittstellen an. Wenn dort nicht `IDispatch` auftaucht, ist für Sie als Skriptentwickler der Spaß vorbei, bevor er richtig angefangen hat.

Erst im vierten Schritt sollten Sie dann noch einmal auf der untersten Ebene ansetzen, der Registry selbst. Öffnen Sie den `HKEY_CLASSES_ROOT` mit `REGEDIT.EXE` und durchsuchen Sie die ProgIDs. Wenn Sie interessante ProgIDs gefunden haben, versuchen Sie diese im COM-Viewer oder COM-Explorer wiederzufinden, um die Information über die `IDispatch`-Unterstützung zu erhalten.

#### Alle Klassen

#### Registry

#### Unterschiedliche Sichten

In den Abbildungen wird deutlich, wie unterschiedlich die Sicht der verschiedenen Tools ist. In allen drei Abbildungen geht es um die *ACL-Komponente* aus dem Exchange Resource Kit:

- ▶ Der VERWEISE-Dialog zeigt den Helpstring der Typbibliothek der Komponente.
- ▶ Der COM-Viewer ist dagegen klassenorientiert; der Fokus ist hier auf der Klasse `ACLObject` aus der *ACL-Komponente*. Angezeigt wird der Friendly Class Name dieser Klasse.
- ▶ In der Registry ist die Sicht ebenfalls klassenorientiert. Hier sind die Klassen allerdings anhand ihrer ProgIDs angeordnet. Der Friendly Class Name erscheint erst nach Auswahl einer Klasse in der rechten Fensterhälfte des Registry-Editors.

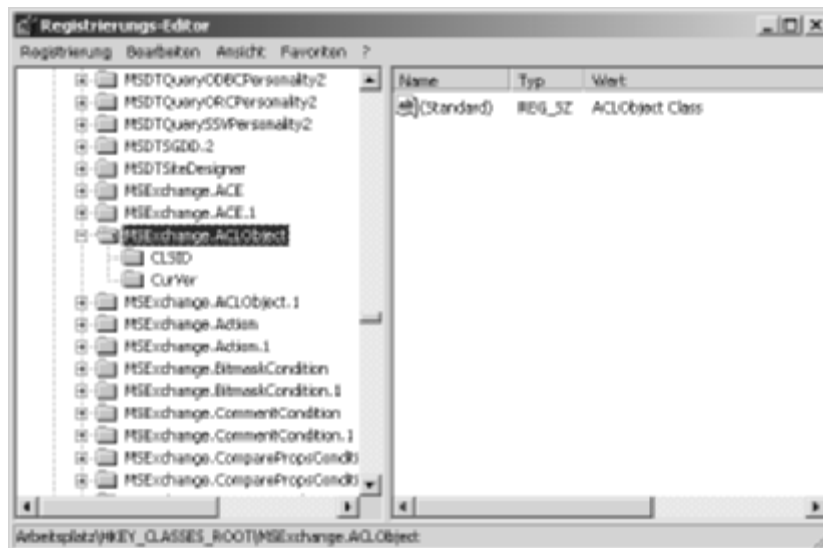


Abbildung 27.3:  
Ansicht der ProgID  
in der Registry



Achten Sie darauf, in welchem Verzeichnis die Komponente und die Typbibliothek gespeichert sind. Sofern die Komponente nicht in dem allgemeinen Verzeichnis System bzw. System32 liegt, kann Ihnen der Standort der Komponente wichtige Informationen darüber liefern, wie die Komponente auf Ihr System gekommen ist und mit welcher Anwendung sie in Verbindung steht.

### 27.1.2 Suche bei Microsoft

#### Suche bei Microsoft

Wenn Sie die bei Ihnen vorhandenen Installationen ausgereizt haben, sollten Sie als Nächstes die Microsoft-Komponenten suchen, die noch nicht bei Ihnen installiert sind. Stöbern Sie in der MSDN Library (entweder auf der MSDN Library-CD/-DVD oder im WWW [MSL00]), der Knowledge Base, den Downloads und im Rest der Microsoft-Homepage unter den Stichwörtern, die Ihr Problem beschreiben. Immer wenn Sie die Begriffe COM, ActiveX oder Scripting lesen, sollten Sie aufmerksam sein.

Abbildung 27.4:  
Hauptgliederungs-  
punkt der MSDN  
Library



Regelmäßig lesen sollten Sie die Kolumne »Scripting Clinic« [CLI00] von Andrew Clinick, der als Programmmanager bei Microsoft maßgeblich an der Weiterentwicklung der ActiveX Scripting-Architektur beteiligt ist.

Scripting Clinic

### 27.1.3 Suche bei anderen Anbietern

Microsoft ist längst nicht mehr der einzige Anbieter von COM-Komponenten. Es gibt zahlreiche kommerzielle Anbieter (z.B. [CMS00], [ACE00], [MAR00]), aber auch genügend Enthusiasten, die mit ihren selbst geschriebenen Komponenten die Menschheit ohne Forderung einer monetären Gegenleistung erfreuen (z.B. [WAR00], [BOR00]). Einige Sites (z.B. [CPS00], [NTF00], [CWA00], [SEC00], [IAN00]) führen Listen verfügbarer COM-Komponenten. Lassen Sie sich nicht abschrecken, wenn einige Sites nur von ASP-Komponenten reden; die meisten dieser Komponenten lassen sich auch von anderen Scripting Hosts aus nutzen.

Andere Anbieter

Der Zeitaufwand, sich regelmäßig intensiv mit diesen Sites zu beschäftigen, amortisiert sich spätestens dann, wenn Sie sich bei Ihrem nächsten Automatisierungsprojekt die Arbeit mit dem Win32-API ersparen können.



### 27.1.4 Newsgroups

Für Scripting und COM-Komponenten gibt es inzwischen einige spezielle Diskussionsgruppen (siehe Anhang E). Diese Themen werden aber auch in plattform- und anwendungsspezifischen Newsgroups besprochen. Das regelmäßige Studium aller relevanten Newsgroups ist ein aufwendiger und von vielen gestressten IT-Fachleuten kaum zu bewältigender Vorgang. Zumindest sollten Sie die Newsgroups aber dann aktiv nutzen, wenn im Hinblick auf eine konkrete Herausforderung keiner der ersten drei Schritte zum Erfolg geführt hat. Eine Suche in Newsgroup-Beiträgen aus der Vergangenheit ist durch die begrenzte Speicherdauer auf den meisten News-Servern erschwert. Am besten suchen Sie über den Internet-Dienst *Deja.com* [DEJ00], der alle Beiträge in vielen tausend Diskussionsgruppen auf unbestimmte Zeit archiviert. Wenn Sie nicht fündig werden, dann scheuen Sie sich nicht und stellen Ihre Frage durch einen eigenen Beitrag. Achten Sie aber bitte darauf, dass Sie in der richtigen Newsgroup und nicht »off-topic« sind. Und in einer ruhigeren Phase in Ihrem Arbeitsalltag sollten Sie sich die Zeit nehmen, auch ohne konkretes Ziel in den Beiträgen der letzten Tage oder Wochen zu stöbern. Bei dieser Gelegenheit können Sie dann der Internet-Gemeinschaft Ihren Tribut zollen, indem Sie selbst Ihnen leicht von der Hand gehende Antworten auf die ein oder andere Frage eines Kollegen vom anderen Ende der Erdkugel beantworten.

Newsgroups

**Andere APIs**

Wenn Sie trotz aller Bemühungen nicht fündig werden, dann sollten Sie einen Blick in die MSDN Library bzw. die Dokumentation der jeweiligen Anwendung werfen, um festzustellen, ob es überhaupt eine Programmierschnittstelle für die gewünschte Funktionalität gibt. Auch heute werden noch Anwendungen auf den Markt gebracht, die sich von der Außenwelt abschirmen und eine Integration in größere Lösungen verhindern. Wenn es eine nur API-basierte Programmierschnittstelle gibt, dann sollten Sie in Erwägung ziehen, dieses API selbst in eine Komponente zu kapseln. Einen Einblick in die Entwicklung von COM-Komponenten erhalten Sie in Kapitel 3.

## 27.2 Analyse von Komponenten

**Analyse** Wenn Sie (endlich) eine interessante oder gar die lang gesuchte Komponente gefunden haben, dann gilt es, ihre Funktionalität zu analysieren. Bei Komponenten, die Sie im Internet oder in einer Dokumentation wie der MSDN Library finden, werden Sie in der Regel schon bei der Suche wenigstens eine minimale Zusatzinformation über die Ziele der Komponente bekommen haben. Anders sieht dies bei den auf dem eigenen System gefundenen Komponenten aus. Dort hilft Ihnen nur die Aussagekraft des Helpstrings der Typbibliothek.

### 27.2.1 Installation

**Installation** Wenn Sie die Komponente nicht auf Ihrem lokalen System gefunden haben, dann müssen Sie sie nun installieren. Kommerzielle Produkte verfügen oft über eine Setup-Routine, viele Komponenten aus dem Free- und Sharewarebereich werden oft einfach durch REGSVR32.EXE installiert. Problematisch ist die Installation dann, wenn Sie die Komponente auf einem anderen System gefunden haben und nun woanders nachvollziehen wollen. Versuchen Sie zuerst die Einzelregistrierung der Komponente (siehe Erläuterungen zur Komponentenregistrierung in Kapitel 2). Wenn dies fehlschlägt, betrachten Sie mit dem Microsoft Dependency Walker, ob alle DLLs, von denen die Komponente abhängig ist, vorhanden sind. Viele Komponenten, gerade solche, die den Zugriff auf größere Anwendungen ermöglichen, können nur dann funktionieren, wenn auch die komplette Anwendung installiert wird.

### 27.2.2 Typbibliothek betrachten

**Typbibliothek** Es mag Sie überraschen, aber der erste Ansatzpunkt für Ihre Analyse sollte nicht unbedingt die Dokumentation der Komponente sein. In der Regel können Sie die Dokumentation wesentlich besser verstehen, wenn Sie sich vorher einen ersten Überblick über die Klassen, Schnittstellen, Attribute, Methoden und Konstantenlisten verschafft haben. Meistens haben diese Bausteine sprechende Namen, und Sie können bereits erkennen, ob die Komponente grund-

sätzlich die von Ihnen gesuchte Funktionalität bereitstellt. Der Microsoft Objektkatalog ist für den ersten Eindruck besser geeignet als der comTLBrowser, da letzterer Details offenbart, die Sie in diesem Stadium noch nicht benötigen.

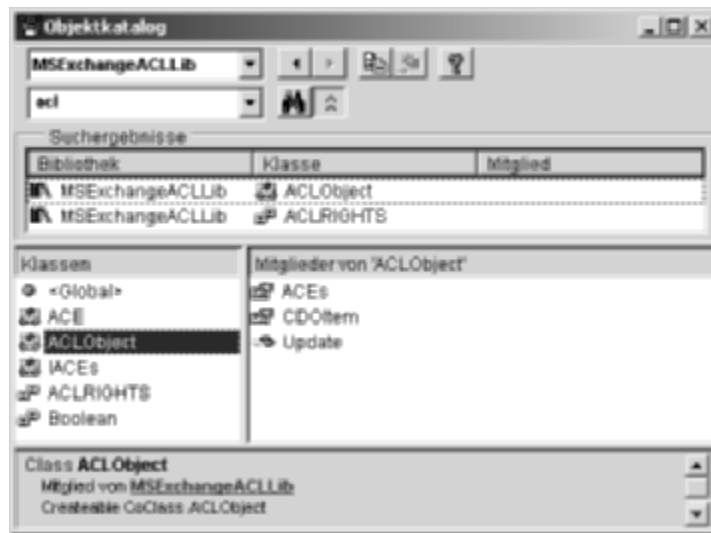


Abbildung 27.5:  
Die Klasse ACLObject aus der ACL-Komponente im Objektkatalog von der VB6-IDE

Wenn Sie die Komponente im VERWEISE-Dialog gefunden haben, dann brauchen Sie sie nur auszuwählen, um sie zur Betrachtung im Objektkatalog zu aktivieren. Haben Sie dagegen die Komponente im COM-Viewer oder COM-Explorer oder gar in der Registry gefunden, dann schauen Sie nach, mit welcher Typbibliothek diese Komponente verbunden ist. Nicht immer werden Sie den Namen der Typbibliothek im VERWEISE-Dialog finden. Verwenden Sie dann den DURCHSUCHEN-Button, um den VERWEISE-Dialog auf die Komponentendatei zu stoßen. Wenn dies mit der Meldung »Verweis auf die angegebene Datei kann nicht hinzugefügt werden.« quittiert wird, dann enthält die Komponentendatei keine Typbibliothek. Suchen Sie in diesem Fall nach einer Datei mit dem gleichen Namen und der Extension .TLB, .OLB oder .RLL bzw. nach einer Datei mit einer dieser Extensionen im gleichen Verzeichnis. Allein durch die Registry bekommen Sie leider nicht heraus, welche Schnittstellen eine Klasse unterstützt.

### 27.2.3 Dokumentation lesen

Wenn Sie nun am liebsten direkt losprogrammieren möchten, anstatt Dokumentationen zu lesen, dann sind Sie keineswegs allein. Die Abneigung gegen das Dokumentationsstudium zieht sich durch alle Altersgruppen von Softwareentwicklern. Auch wenn es inzwischen eine starke Vereinheitlichung beim Aufbau und Vorgehen gibt, werden Sie doch in der Regel Zeit sparen, wenn Sie die Dokumentation zumindest überfliegen, bevor Sie die erste Zeile

#### Dokumentation

kodieren. Verschaffen Sie sich einen Überblick über den Aufbau der Dokumentation und suchen Sie allgemeine Erläuterungen zur Arbeitsweise der Komponente. Wenn die Dokumentation Codebeispiele enthält, dann freuen Sie sich darüber, Ihre ersten Gehversuche mit diesen Beispielen bestreiten zu können. Bei Microsoft können Sie nach Codebeispielen im MSDN Code Center [MCC00] suchen.

**Suche nach  
Hilfdateien**

In einigen Fällen (gerade bei Funden auf der eigenen Festplatte) werden Sie keine Dokumentation zur Komponente zur Hand haben. Suchen Sie in diesem Fall zunächst nach einer Hilfdatei. Diese können Sie häufig anhand des Namens, des Standorts (im selben oder benachbarten Verzeichnis) oder anhand des Eintrags HELPDIR im Registry-Eintrag einer Typbibliothek lokalisieren. Oft ist aber gar keine Hilfdatei vorhanden. Schauen Sie dann in die MSDN Library und suchen Sie auf der Microsoft Homepage.

**Hilfe aus dem  
Internet**

Manchmal werden Sie aber gar keine Dokumentation finden. Schöpfen Sie in diesem Fall das Internet als Quelle aus: Suchen Sie nicht nur in Newsgroups, sondern über das ganze Web. Nicht nur PC-Zeitschriften berichten über undokumentierte Komponenten, sondern auch eifrige Anwender stellen ihre Erkenntnisse in solchen schweren Fällen besonders gerne auf ihre Homepage.



Viele Komponenten werden Sie mit den in diesem Buch vorgestellten Tools und Vorgehensmodellen auch ohne Dokumentation zu einem befriedigenden Grad erkunden können. Manche Feinheiten und Tricks bleiben allerdings ohne eine gute Dokumentation verborgen.

## 27.2.4 Schrittweise Testcodierung

**Testcodierung**

Wenn Sie Beispielprogramme zur Hand haben, dann sollten Sie zuerst versuchen, diese zum Laufen zu bringen. Fangen Sie aber mit den einfacheren Beispielen an. Wenn Sie Code in den Beispielen nicht verstehen, dann sollten Sie – genau wie in diesem Buch an vielen Stellen vorgemacht – die Objekthierarchie einer Komponente schrittweise vom Stammobjekt herab erkunden. Geben Sie sich zuerst damit zufrieden, die Objekthierarchie anhand der Objektnamen auszugeben. Gehen Sie dann zu den weiteren Eigenschaften der Objekte über. Erst danach testen Sie die Methoden und die Möglichkeiten, die Objekthierarchie zu verändern. Gerade bei großen Objektmodellen, bei denen Sie auf bestimmte Äste der Objekthierarchie fokussieren müssen, sollten Sie Ihre konkrete Problemstellung bei der Auswahl Ihrer Testkodierungen im Hinterkopf haben.



Zu einzelnen Attributen und Methoden einen kurzen Blick in die Dokumentation zu werfen ist effizienter als das eigene Herumprobieren.



### 27.2.5 Ausführliches Studium der Dokumentation

Nach einigen erfolgreichen und einigen sicherlich erfolglosen Gehversuchen mit den neuen Komponenten sollten Sie sich nun noch einmal viel Zeit nehmen, die Dokumentation (sofern vorhanden) intensiv zu lesen. Wenn keine Dokumentation vorhanden ist, recherchieren Sie im Internet und werfen Sie die Codebeispiele, die nicht funktionieren, in die Runde der passenden Newsgroups.

**Studium der  
Dokumentation**

Unterschätzen Sie die Bedeutung dieser Phase für das Gelingen eines Automatisierungsprojekts nicht. Nachdem Sie erste Testkodierungen unternommen haben, werden Sie die Quellen mit ganz anderen Augen lesen.

### 27.2.6 Implementierung

Sofern Sie die bisherigen Schritte nicht nur aus allgemeinem Interesse, sondern aufgrund einer bestimmten Problemstellung durchlaufen haben, ist nun endlich der Zeitpunkt gekommen, mit der Kodierung der Lösung zu beginnen.

**Implementierung**

Natürlich werden Sie in dieser Phase auch immer wieder auf die Dokumentation bzw. auf Internet-Quellen zurückgreifen müssen. Vielleicht sind Sie auch in der glücklichen Lage, einen Support-Vertrag mit dem Hersteller der Komponente zu haben, wobei diese Möglichkeit nicht immer schneller zum Ziel führt als eine Frage in der Internet-Gemeinde.



# A Grundlagen objektorientierter Komponenten- architekturen

Das objektorientierte Paradigma hat sich bereits in vielen Bereichen der Softwareentwicklung etabliert. Der Komponentengedanke ist die konsequente Weiterentwicklung des Prinzips der Objektorientierung; er durchzieht auch immer mehr die Windows-Betriebssysteme. Dieser Anhang bietet Personen, die noch keine Kenntnisse über Objektorientierungen und Komponentenarchitekturen haben, einen kompakten Überblick über die zugehörige Begriffswelt.

## A.1 Objektorientierung

EDV-Fachleute sind heute in der Regel bereits mit dem objektorientierten Paradigma vertraut. Da in diesem Buch die Objektorientierung eine zentrale Rolle spielt, soll diese Kapitel eine gemeinsame Begriffswelt sicherstellen. Das vorliegende Buch kann aber keine umfassende Einführung in dieses weitreichende Thema geben. Daher sollen an dieser Stelle nur einige wesentliche Definitionen gegeben werden.

**Das objektorien-  
tierte Paradigma**

Die Kürzel OO oder OOx sind Sammelbegriffe für die Nutzung objektorientierter Methoden und Techniken im Softwarelebenszyklus: Objektorientierte Analyse (OOA), Design (OOD), Programmierung (OOP) und Datenbank-mManagementsysteme (OODBMS) (vgl. [QUI94], S. 264).

### A.1.1 Objekte

*»Ein Objekt ist ein Modell eines (meist sehr genau) definierten Teils der Realität, sei es nun in der Umgebung von Alltag und Umwelt oder derjenigen von Daten. Ein Objekt kann als individuelle, identifizierbare Einheit mit folgenden Eigenschaften bezeichnet werden: Die Einheit ist real oder abstrakt, die Grenzen der Einheit sind genau definiert und die Einheit verhält sich innerhalb dieser Grenzen nach genau definierten Regeln.« ([BÖH96], S. 490)*

**Objekte**

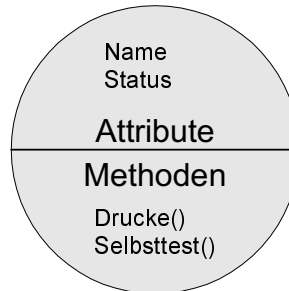
*»Ein Objekt ist eine im laufenden System konkret vorhandene und agierende Einheit.« ([OES97], S. 221)*

Ein Objekt besteht aus Attributen und Methoden. Attribute enthalten Informationen (Daten) über das Objekt. Methoden sind Operationen, die das

**Attribute und  
Methoden**

Verhalten eines Objekts bestimmen. Der Begriff Mitglied (Member) wird als Oberbegriff für Attribute und Methoden verwendet.

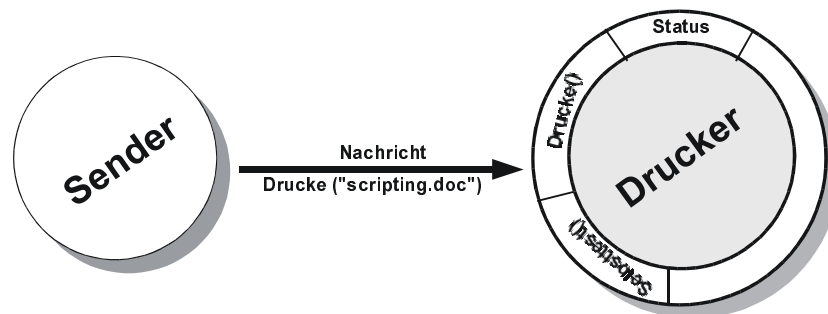
Abbildung A.1:  
Beispiel für ein  
Objekt mit vier Mit-  
gliedern (jeweils  
zwei Attributen und  
zwei Methoden), das  
einen Drucker  
repräsentiert



#### Kapselung

Ein wichtiges Prinzip der Objektorientierung ist das *Information Hiding* durch *Kapselung*: Daten und Programmcode bilden in einem Objekt eine geschlossene Einheit (Kapsel), die für die Außenwelt nur über voll definierte Schnittstellen zugänglich ist. Objekte kommunizieren untereinander über den gegenseitigen Aufruf von Methoden; man spricht in der Objektorientierung davon, dass Objekte über Botschaften (Nachrichten) miteinander kommunizieren. Die Methoden eines Objekts bestimmen, welche Nachrichten ein Objekt empfangen kann. Die Daten eines Objekts sind im Idealfall nicht direkt, sondern nur über den Aufruf von Methoden zugänglich.

Abbildung A.2:  
Ein Client sendet  
einem Objekt, das  
einen Drucker  
repräsentiert, die  
Nachricht, eine  
Datei zu drucken.



#### Ereignisse

Einige objektorientierte Sprachen (z.B. Visual Basic) verwenden darüber hinaus auch noch *Ereignisse*. Ein Ereignis ist eine Nachricht, die ein Objekt an einen Nutzer sendet, wenn bestimmte Bedingungen eingetreten sind. Im Gegensatz dazu ist eine Methode eine Routine, die der Nutzer in dem Objekt aufruft. Auf der Implementierungsebene steckt hinter einem Ereignis ein Methodenaufruf in umgekehrter Richtung, also von dem Objekt an seinen Nutzer. Dies setzt oft voraus, dass der Nutzer selbst ein Objekt ist.

#### Objektzugriff

Objekte werden – ebenso wie elementare Datentypen – mit Hilfe von Variablen verwendet. Diese Variablen heißen oft auch *Objektvariablen*. Eine Objektvariable enthält entweder das Objekt oder stellt einen Zeiger auf den

Speicherbereich dar, in dem das Objekt liegt. Die in diesem Buch vorgestellte Sprache Visual Basic kennt nur die letztere Form.

Als Nutzer eines Objekts wird in der Regel kein Mensch bezeichnet, sondern ein Programm, das ein Objekt über eine Objektvariable verwendet. Alternativ spricht man auch von *Client*.

**Client**

## A.1.2 Schnittstellen (Interfaces)

»Eine Schnittstelle umfasst die ausführbaren Dienste eines Objekts. In der Schnittstelle können der Typ eines formalen Parameters festgelegt und das Über-alles-Verhalten der Anfrage beschrieben sein.« ([QUI94], S. 226)

Bei einem Objekt unterscheidet man zwischen der Schnittstelle und der Implementierung. Eine Schnittstelle beschreibt eine Menge von Mitgliedern (also Attribute und Methoden) eines Objekts mit den zugehörigen Parametern und Rückgabewerten; wie diese Mitglieder implementiert sind, ist dabei ohne Bedeutung. Eine Schnittstelle repräsentiert die Tür zu der Funktionalität eines Objekts und ist eine Untermenge seiner Funktionalität. Eine Schnittstelle ist auch 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 Möglichkeit zur formalen Beschreibung von Schnittstellen ist die Interface Definition Language (IDL); sie wird auch in COM eingesetzt.

**Interfaces**

In Umgebungen, in denen jedes Objekt nur eine Schnittstelle haben darf, ist die Schnittstelle gleichzusetzen mit der Vereinigungsmenge aller Mitglieder des Objekts. Wenn ein Objekt Mehrfachschnittstellen haben kann, dann fasst eine Schnittstelle eine Menge von Mitgliedern eines Objekts zu einer Einheit zusammen. Jede dieser Schnittstellen repräsentiert eine Teilmenge der Funktionalität des Objekts. Bei Mehrfachschnittstellen können die Teilmengen nach beliebigen Kriterien gebildet werden; in der Regel erfolgt die Mengengbildung jedoch nach semantischen Gesichtspunkten.

**Einfach versus Mehrfachschnittstellen**

Mehrfachschnittstellen ermöglichen einen höheren Abstraktionsgrad als Einfachschnittstellen. Für einen Nutzer ist es sehr wichtig zu wissen, ob ein Objekt eine bestimmte Funktionalität bereitstellt. Im Fall der Mehrfachschnittstellen kann ein Nutzer das Objekt auf das Bestehen einer bestimmten Schnittstelle hin abfragen und sich im Falle einer positiven Antwort darauf verlassen, dass alle zugehörigen Methoden und Attribute angeboten werden. Im Fall von Einfachschnittstellen muss der Nutzer dagegen die Existenz jeder einzelnen Methode und jedes einzelnen Attributs bei dem Objekt erfragen. Durch Mehrfachschnittstellen wird der Aufruf einer Funktionalität von dem Test auf Bestehen der Funktionalität entkoppelt.

**Vorteil von Mehrfachschnittstellen**

### A.1.3 Klassen

»Eine Klasse ist die Definition der Attribute, Operationen und der Semantik für eine Menge von Objekten. Alle Objekte einer Klasse entsprechen dieser Definition« ([OES97], S. 157).

|                                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|-----------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Instanzen und Instanzieren</b> | Klassen dienen als Schablonen (auch: Templates, Vorlagen, Objektfabriken) für Objekte. Das Verhältnis von Objekt zu Klasse wird durch den Begriff <i>Instanz</i> beschrieben: Ein Objekt ist eine <i>Instanz</i> einer Klasse. Den Vorgang der Bildung von Instanzen nennt man <i>Instanzieren</i> (zum Teil auch <i>Instantiieren</i> geschrieben). Einige Autoren (vgl. z.B. [OES97], S. 221 und [BLA97], S. 421) weisen in diesem Zusammenhang darauf hin, dass die korrekte Übersetzung des englischen Begriffs <i>Instance</i> das deutsche Wort <i>Exemplar</i> ist. In der Fachsprache hat sich aber der Anglizismus <i>Instanz</i> durchgesetzt. |
| <b>Typ</b>                        | Häufig wird eine Klasse auch als der <i>Typ</i> eines Objekts bezeichnet. Der feine Unterschied zwischen den Begriffen Klasse und Typ soll hier nicht näher diskutiert werden (vgl. dazu [UNL95], S. 34ff.).                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Schablone</b>                  | Eine Klasse gibt vor, welche Attribute und Methoden jede Instanz der Klasse haben wird. Durch das Instanzieren einer Klasse entsteht ein Objekt mit genau diesen Attributen und Methoden. Das durch die Methoden definierte Verhalten ist bei allen Instanzen einer Klasse gleich; die durch die Attributwerte repräsentierten Daten ändern sich jedoch individuell für jedes Objekt.                                                                                                                                                                                                                                                                    |

Eine Klasse ist wie eine Form für Weihnachtsplätzchen. Man stanzt damit Objekte aus, wobei die grundlegende Form vorgegeben ist. Danach ist es aber immer noch möglich, den einzelnen Plätzchen unterschiedliche Zuckergüsse zu geben.

Abbildung A.3:  
Darstellung eines  
Instanzierungsvorgangs



**Abstrakte Klassen** Normalerweise besteht eine Klasse aus einer Schnittstellendefinition und einer Implementierung. Eine Klasse, die keine Implementierung liefert, heißt abstrakte Klasse. Eine abstrakte Klasse kann nicht instanziiert werden, sondern dient der Vererbung.

#### Beispiel

**Gebäude in VBScript** Das Konzept der Klasse sei an einer konkreten Realisierung einer Klasse in der Programmiersprache Visual Basic Script gezeigt. Bitte achten Sie auf das Grundprinzip der Definition einer Klasse, der Instanzierung und der Nutzung.

## Objektorientierung

```
' === Klassendefinition

Class Gebaeude
' --- Attribute der Gebäude-Klasse
Dim Name
Dim Strasse
Dim PLZ
Dim Ort
Dim Status einweihen
Dim Architekt
Dim Bezugsdatum
' --- Nicht öffentliche globale Variablen
Private AnzahlNutzer
' --- Methoden der Gebäude-Klasse
Sub Einweihung(Jahr)
 Bezugsdatum = Jahr
 Status = "Eingeweiht"
End Sub
Sub NeuerNutzer(Name)
 AnzahlNutzer = AnzahlNutzer + 1
' Weitere Implementierung...
End Sub
Function ErmittleAnzahlNutzer(Stand)
 ErmittleAnzahlNutzer = AnzahlNutzer
End Function
Sub Abreissen()
 Status = "nicht mehr existent"
End Sub

End Class
```

*Listing 1.1: Beispiel für eine Klassendefinition in VBScript  
[CD: /code/Anhang/gebaeude.vbs]*

**Im Folgenden werden zwei Instanzen dieser Klasse unter den Namen WBHaus und KAHaus erzeugt und verwendet.**

```
' === Klassennutzung der Gebäude-Klasse

' --- Instanziierung!
Dim WBHaus
Set WBHaus = new Gebaeude
' --- Nutzung der Attribute
WBHaus.Name = "WillyBrandtHaus"
WBHaus.Strasse = "Wilhelmstraße 140"
WBHaus.PLZ = 10963
WBHaus.Ort = "Berlin"
WBHaus.Architekt = "Prof. Helge Bofinger"
' --- Nutzung der Methoden
WBHaus.Einweihung(1996)
```

```
WBHaus.NeuerNutzer("SPD")
WBHaus.NeuerNutzer("speedGroup")
Msgbox WBHaus.ErmittleAnzahlNutzer(#6/20/2000#)
' --- Instanziierung eines zweiten Gebäudes
Dim KAHaus
Set KAHaus = new Gebaeude
' --- Nutzung der Attribute
KAHaus.Name = "KonradAdenauerHaus"
KAHaus.Ort = "Bonn"
' usw.
' --- Objektvernichtung
KAHaus.Abreissen

Set KAHaus = Nothing
```

Listing 1.2: Beispiel für die Nutzung der zuvor definierten Klasse Gebaeude [CD: /code/Anhang/gebaeude.vbs]

### Objekt versus Klasse

Ein Objekt ist eine Instanz einer Klasse. Objekte werden immer zur Laufzeit gebildet, während eine Klasse ein statisches Konzept repräsentiert, das zur Entwicklungszeit im Quellcode festgelegt wird.

Nicht immer ist es einfach, die Begriffe zu trennen und sich in einem konkreten Fall zwischen der Verwendung des Begriffs *Klasse* und des Begriffs *Objekt* zu entscheiden. Wenn man beispielsweise eine Methode beschreibt, soll man dann von der »Methode der Klasse *x*« oder der »Methode des Objekts *x*« sprechen? Grundsätzlich sind beide Ausdrücke richtig. Welchen Begriff Sie im konkreten Fall verwenden, sollten Sie vom Kontext abhängig machen. Verwenden Sie *Objekt* nur dann, wenn es um das Verhalten einer konkreten Instanz zur Laufzeit geht. Beschreiben Sie jedoch allgemein die bereitgestellten Funktionen, so verwenden Sie besser die *Klasse* als Bezugspunkt. Viele stellen den Begriff *Objekt* viel zu sehr in den Vordergrund und sprechen in der Beschreibung der Klassen meistens von Objekten.

### Abgrenzungsschwierigkeiten

#### A.1.4 Vererbung (Inheritance)

»Unter Vererbung versteht man die Übernahme der Merkmale einer (Ober-)Klasse A in eine (Unter-)Klasse B. Die Unterklasse besitzt zusätzlich zu den eigenen Merkmalen auch alle Merkmale der Oberklasse.« ([GOO97], S. 398)

»Vererbung bezeichnet die gemeinsame Verwendung von Attributen und Operationen durch verschiedene Klassen auf Basis einer hierarchischen Relation. Eine Klasse kann sehr allgemein definiert sein und dann in immer detaillierteren Unterklassen verfeinert werden. Jede Unterklasse übernimmt oder erbt alle Eigenschaften ihrer Oberklasse und fügt ihre eigenen, individuellen Eigenschaften hinzu. Die Eigenschaften der Oberklasse müssen nicht in jeder Unterklasse wiederholt werden.« ([RUM93], S. 3f.)



## Objektorientierung

Die Oberklasse wird auch Basisklasse, Superklasse oder Elternklasse, die Unterklasse abgeleitete Klasse, Subklasse oder Kinderklasse genannt. Vererbung ermöglicht es, Attribute und Methoden, die mehreren Klassen gemein sind, an einer zentralen Stelle zu definieren. Unterklassen können wiederum Oberklassen für andere Klassen sein. Daraus ergibt sich eine baumartige *Vererbungshierarchie* (auch Klassenhierarchie genannt).

### Basisklassen und abgeleitete Klassen

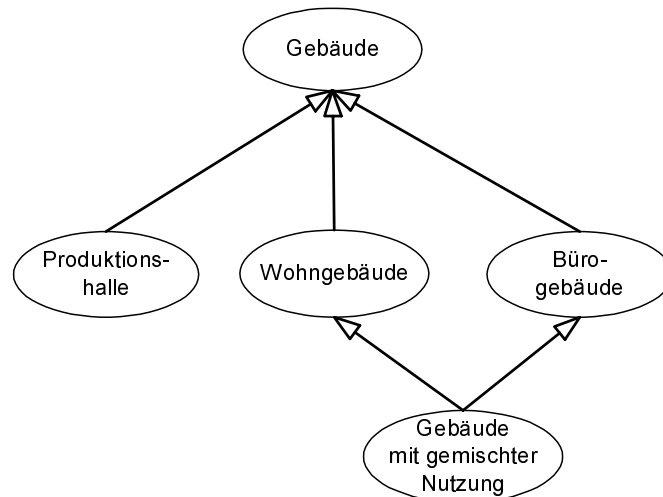


Abbildung A.4:  
Beispiel für eine  
Vererbungs-  
hierarchie

Mit Vererbung ist in der Regel eine Implementierungsvererbung gemeint, d.h., die Unterklasse erbt den Code der Oberklasse. Wird in einer erbenden Unterklasse eine Methode aufgerufen, die von der Oberklasse geerbt wurde, so wird der in der Oberklasse implementierte Programmcode aufgerufen. Eine redundante Implementierung in der Unterklasse ist nicht notwendig. Gleichwohl kann eine Unterklasse eine neue Implementierung einer Methode der Oberklasse liefern. Man spricht dann von *Überschreibung* (*Overriding*).

### Implementierungsvererbung

Eine andere Form der Vererbung ist die Schnittstellenvererbung. Dabei erbt die Unterklasse nur die Schnittstellendefinitionen mit Namen und Typ der Attribute sowie Namen, Parameter und Rückgabewerte der Methoden. Die eigentliche Implementierung der Methoden wird nicht vererbt. Eine Unterklasse muss also für eine von der Oberklasse geerbte Methode eine eigene Implementierung bereitstellen.

### Schnittstellenvererbung

Die Erstellung einer Vererbungshierarchie ist eine Aufgabe im Rahmen der Entwicklung von objektorientierten Anwendungen und Komponenten. Als Nutzer von Softwarekomponenten haben Sie in der Regel mit der Vererbungshierarchie wenig zu tun. Bei einigen Softwarekomponenten (z.B. dem *Windows Management Instrumentarium*, kurz: *WMI*) erleichtert jedoch die Kenntnis der Vererbungshierarchie das Verständnis der Verwendung der Klassen.



### A.1.5 Beziehungen zwischen Objekten und Klassen

Zwischen Klassen kann es zwei Arten von Beziehungen geben: Vererbungsbeziehungen und Nutzungsbeziehungen.

**Klassenhierarchie** *Vererbungsbeziehungen* drücken aus, welche Klassen von anderen Klassen erben, und sie sind ein Indikator für die Ähnlichkeit der Klassen. Aus den Vererbungsbeziehungen ergibt sich die Vererbungshierarchie (auch: Klassenhierarchie). Diese Beziehungsart wurde bereits im letzten Unterkapitel besprochen.

**Objekthierarchie** *Nutzungsbeziehungen* (auch: Containment-Beziehungen) beziehen sich auf die möglichen Verbindungen der Instanzen zur Laufzeit. Eine Instanz kann entweder durch eine Aggregation oder durch eine Assoziation mit einem anderen Objekt verbunden sein.

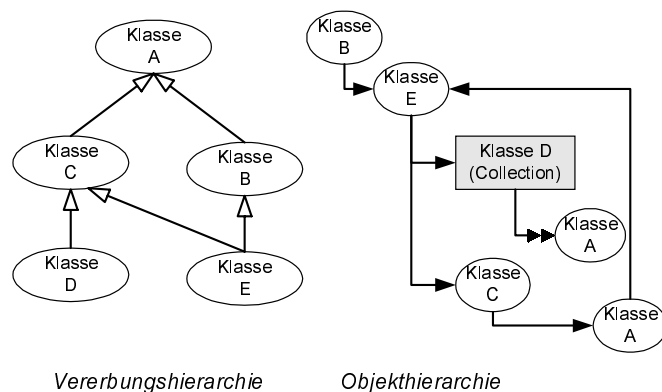
- Bei einer *Aggregation* (Is-part-of-Beziehung) ist ein Objekt Bestandteil eines anderen Objekts.
- Bei einer *Assoziation* bestehen die Objekte unabhängig voneinander, es gibt jedoch einen Verweis von einem Objekt auf das andere. Aus den Nutzungsbeziehungen ergibt sich eine *Objekthierarchie* (auch: Containment-Hierarchie). Eine Objekthierarchie bildet ein *Objektmodell*.

#### Definitionen

Rauh und Stickel ([RAS97], S. 260ff.) weisen darauf hin, dass die Semantik der Beziehungsarten nicht in allen objektorientierten Ansätzen die gleiche ist. Dieses Buch orientiert sich an den in der COM-Welt üblichen Definitionen:

- »Eine Klassenhierarchie beschreibt die Vererbung. Dies bedeutet, dass die Klassenhierarchie zeigt, wie Objekte von einfacheren Objekten abgeleitet werden, indem sie ihr Verhalten erben.« ([MIC98], S. 66)
- »Das Objektmodell definiert eine Objekthierarchie, die einem objektbasierten Programm seine Struktur gibt. Das Objektmodell legt die Beziehungen zwischen den Programmobjekten fest und organisiert die Objekte so, dass die Programmierung erleichtert wird.« ([MIC98], S. 66).

Abbildung A.5:  
Vererbungs-  
hierarchie versus  
Objekthierarchie



### Kardinalität von Beziehungen zwischen Objekten

Eine Nutzungsbeziehung besitzt eine Kardinalität. Ähnlich wie bei der Modellierung relationaler Datenbanken kann es auch zwischen Objekten 1-zu-1-, 1-zu-n- und n-zu-m-Beziehungen geben:

- ▶ Bei einer 1-zu-1-Beziehung nutzt eine Instanz einer Klasse genau eine (andere) Instanz einer (anderen) Klasse.
- ▶ Bei einer 1-zu-n-Beziehung nutzt eine Instanz einer Klasse mindestens eine (andere) Instanz einer (anderen) Klasse.
- ▶ Bei einer n-zu-m-Beziehung gibt es eine beidseitige 1-zu-n-Nutzungsbeziehung: eine Instanz der Klasse A nutzt n Instanzen der Klasse B. Eine Instanz der Klasse B nutzt andererseits n Instanzen der Klasse A.

**Kardinalitäten**

**1-zu-1**

**1-zu-n**

**n-zu-m**

Diese Bedingungen können weiter aufgefächert werden, wenn man zwischen einer Muss- und einer Kann-Beziehung unterscheidet. So kann zwischen einer 1-zu-1- und einer 1-zu-0/1-Beziehung unterschieden werden. Im ersten Fall muss eine Instanz der Klasse A mit einer Instanz der Klasse B verbunden sein. Im Fall der 1-zu-0/1-Beziehung ist die Nutzungsbeziehung eine optionale Beziehung. Entsprechend gibt es auch 1-zu-0/n-Beziehungen und 0/n-zu-0/m-Beziehungen.

**1-zu-0/1,  
1-zu-0/n,  
0/n-zu-0/m**

### Objektmengen

Eine 1-zu-1- oder 1-zu-0/1-Beziehung kann durch ein Mitglied (ein Attribut oder eine Methode) der Ausgangsklasse A, das einen Zeiger auf eine Instanz der Zielklasse B liefert, modelliert werden. Dies ist bei den anderen Beziehungsarten nicht möglich. Instanzen können daher zu *Objektmengen* zusammengefasst werden.

Eine Objektmenge (auch: Objektsammlung) fasst n Instanzen zusammen. Es gibt in der Objektorientierung zwei weit verbreitete Verfahren, eine Objektmenge zu realisieren:

- ▶ Das erste Verfahren sind (doppelt) verkettete Listen. Dabei verweist ein Objekt der Objektmenge durch einen Zeiger auf das nächste Objekt. Bei einer doppelten Verkettung weist auch ein Objekt auf das vorherige Objekt. Eine Objektmenge wird dann über einen Zeiger auf das erste Objekt der verketteten Liste identifiziert. Der Nachteil dieses Verfahrens ist, dass die für die Verkettung notwendigen Attribute und Methoden (z.B. Next, Previous, AddNew(), Remove() etc.) in den Klassen der in die Liste aufzunehmenden Objekte implementiert sein müssen. Eine Instanz einer Klasse, die diese Mitglieder nicht anbietet, kann nicht in die Liste aufgenommen werden. Um Implementierungsaufwand zu sparen, werden die für die Listenverwaltung notwendigen Funktionalitäten üblicherweise in einer Basisklasse implementiert, von der die Klassen erben, die in einer Liste aufgenommen werden sollen. Diese Form der Implementierung von Objektmengen ist in objektorientierten Programmiersprachen wie C++ und Java üblich.

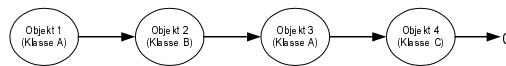
**Verkettete Listen**

## Mengenverwaltungs-klassen

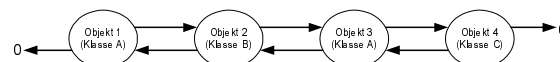
- Das zweite Verfahren basiert auf einer separaten *Objektmengen-Verwaltungs-klasse*. Diese Klasse implementiert eine Möglichkeit, Zeiger auf eine beliebige Menge von Instanzen zu speichern und zu verwalten. Zur Laufzeit wird eine Instanz einer Objektmengen-Verwaltungs-klasse (*Verwaltungsobjekt*) erzeugt, die Zeiger auf Instanzen anderer Klassen aufnehmen kann. Die einzelnen Instanzen, die in die Objektmenge aufgenommen werden, müssen über keinerlei Verwaltungsfunktionen verfügen. Visual Basic verwendet dieses Verfahren zur Implementierung von Objektmengen. Zur Abgrenzung einer Objektmengen-Verwaltungs-klasse sei eine normale Klasse, die nicht in der Lage ist, eine Objektmenge zu verwalten, eine *Einzelklasse* genannt. Diese Form der Implementierung von Objektmengen verwendet Visual Basic

Abbildung A.6:  
Grafische Darstellung der verschiedenen Verfahren zur Implementierung von Objektmengen.

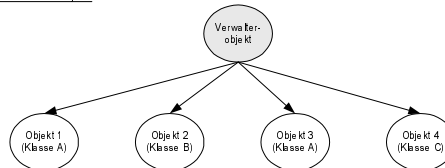
Einfach verkettete Liste



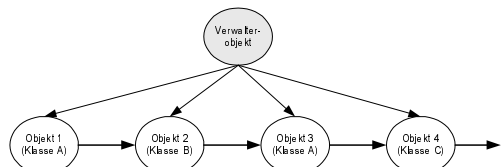
Doppelt verkettete Liste



Liste mit Verwaltungsobjekt



Kombination aus Verwaltungsobjekt und einfach verketteter Liste



## Modellierungen von mehrwertigen Nutzungsbeziehungen durch Objektmengen

### Modellierung durch Objektmengen

1-zu-n- und n-zu-m-Beziehungen zwischen Objekten können durch die Verwendung von Objektmengen modelliert werden. Dabei ist der 1-zu-n-Fall einfacher. Ähnlich wie bei relationalen Datenbanken kann eine n-zu-m-Beziehung nicht direkt modelliert werden, sondern muss auf zwei 1-zu-n-Beziehungen zurückgeführt werden. Die folgende Tabelle unterscheidet die vier Fälle zur Modellierung mehrwertiger Nutzungsbeziehungen.



Im Component Object Model (COM) werden fast ausschließlich Verwalterklassen eingesetzt.

|                         | 1-zu-n-Beziehung                                                                                                               | n-zu-m-Beziehung                                                                                                                                                                                                                                              |
|-------------------------|--------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Verkettete Liste</b> | Klasse A enthält einen Zeiger auf das erste Element einer verketteten Liste von Instanzen der Klasse B.                        | Klasse A enthält einen Zeiger auf das erste Element einer verketteten Liste von Instanzen der Klasse B; Klasse B enthält einen Zeiger auf das erste Element einer verketteten Liste von Instanzen der Klasse A.                                               |
| <b>Verwalterklasse</b>  | Klasse A enthält einen Verweis auf eine Verwalterklasse C. Die Verwalterklasse C kann dann n Instanzen der Klasse B aufnehmen. | Klasse A enthält einen Verweis auf eine Verwalterklasse C. Die Verwalterklasse C kann dann n Instanzen der Klasse B aufnehmen; Klasse B enthält einen Verweis auf eine Verwalterklasse D. Die Verwalterklasse D kann dann n Instanzen der Klasse A aufnehmen. |

Tabelle A.1:  
Fallunterscheidung  
bei der Modellierung  
mehrwertiger Nut-  
zungsbeziehungen

### Grafische Darstellung

In der in diesem Buch verwendeten Notation (siehe Anhang B) werden Einzelobjekte durch Ovale und Objektmengen-Verwaltungsobjekte durch Rechtecke dargestellt. Ein einfacher Pfeil bedeutet eine 1-zu-1-Beziehung; ein doppelter Pfeil eine 1-zu-n-Beziehung. Die folgende Abbildung zeigt die Modellierung und grafische Darstellung der wichtigsten Beziehungstypen. Eine Unterscheidung zwischen Muss- und Kann-Beziehungen findet nicht statt. Die Beschriftung mit »Klasse xy« bedeutet, dass es sich um ein Objekt der Klasse xy handelt.

### Grafische Notation

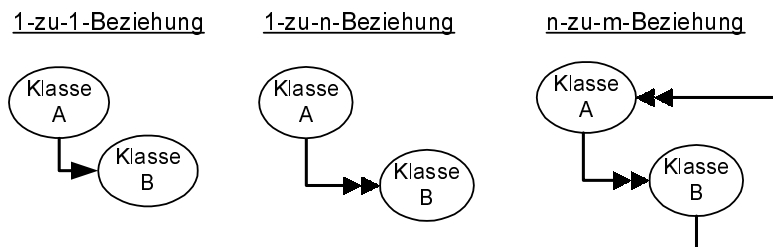


Abbildung A.7:  
Nutzungsbeziehungen auf Basis  
verketteter Listen

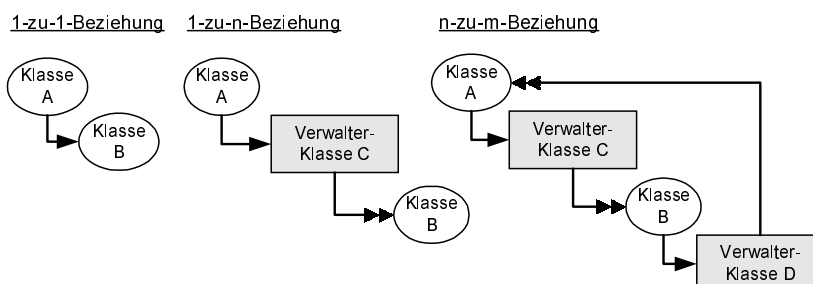


Abbildung A.8:  
Nutzungsbeziehungen auf Basis von  
Mengenverwaltungs-  
klassen

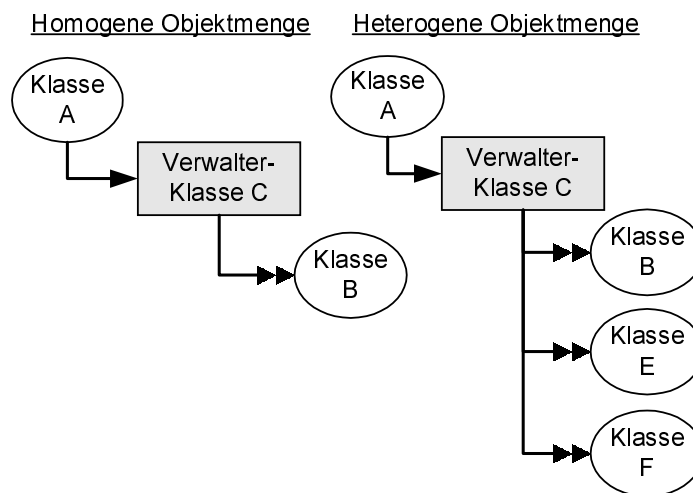
### Zusammensetzung von Objektmengen

Ein weiteres Unterscheidungskriterium zwischen Objektmengen sind die Bedingungen an die Zusammensetzung der in der Menge enthaltenen Objekte. Man unterscheidet drei Typen:

- Set** ▶ *Sets* dürfen nur Objekte einer Klasse und jedes konkrete Objekt jeweils nur einmal enthalten.
- Bag** ▶ *Bags* dürfen nur Objekte einer Klasse enthalten, Duplikate sind jedoch erlaubt.
- Collection** ▶ *Collections* dürfen Objekte verschiedener Klassen enthalten.

Sets und Bags sind homogene Objektmengen, während Collections heterogene Objektmengen darstellen. Als deutsche Übersetzung für den Begriff *Collection* findet man *Kollektion*.

Abbildung A.9:  
Grafische Darstellung homogener und heterogener Objektmengen mit Verwalterklasse



In COM wird die diskutierte Unterscheidung zwischen Set, Bag und Collection in der Regel nicht vorgenommen. In der COM-Welt werden Objektmengen unabhängig davon, ob die enthaltenen Instanzen homogen oder heterogen sind, Collections genannt.

### A.1.6 Objektmodelle

#### Objektmodelle und Objektdiagramme

Im Rahmen der *Object Modelling Technique (OMT)* wird folgende Definition für Objektmodell verwendet: »Das Objektmodell beschreibt die statische Struktur der Objekte in einem System und ihre Relationen. Das Objektmodell enthält Objektdiagramme. Ein Objektdiagramm ist ein Graph, dessen Knoten Objektklassen sind und dessen Linien (Kanten) Relationen zwischen Klassen sind.« ([RUM93], S. 7)

Objektmodelle werden eingesetzt, um den gewünschten Ausschnitt aus der Realwelt auf natürliche Weise zu modellieren. Die Realwelt kann aus der Sicht eines Objektmodells auch selbst wieder eine Software sein. Sofern Objektmodelle Programmierschnittstellen zu Anwendungen darstellen, spiegeln diese die Benutzeroberfläche und/oder das konzeptionelle Design der Anwendung wider. Während die Klassenhierarchie durch den Quellcode konkret vorgegeben ist, sind Objektmodelle zunächst abstrakt; sie werden erst zur Laufzeit realisiert und können sich zur Laufzeit verändern. Den Rahmen für die Veränderung bilden die durch den Quellcode vorgegebenen möglichen Nutzungsbeziehungen. Natürlich können die Vorgaben des Quellcodes so starr sein, dass es nur eine mögliche Anordnung der Instanzen gibt.

Ein Objektmodell kann aus beliebig vielen Klassen bestehen. Seine Struktur muss kein echter Baum sein, sondern kann auch Rekursionen enthalten. Dabei wird die Klasse, die den Einstieg in das Objektmodell darstellt, *Stammklasse* (auch: *Wurzelklasse*) genannt. In vielen Fällen sind auch Klassen, die in der Realwelt nicht in Beziehung zu anderen Klassen der Komponente stehen, so modelliert, dass sie über die Stammklasse erreichbar sind.

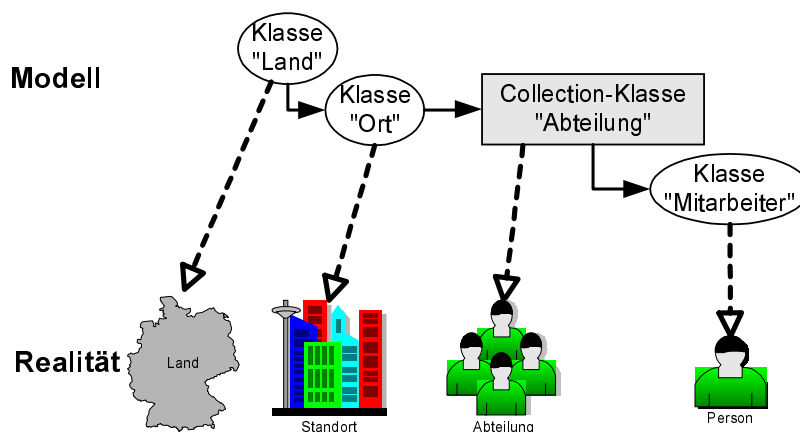
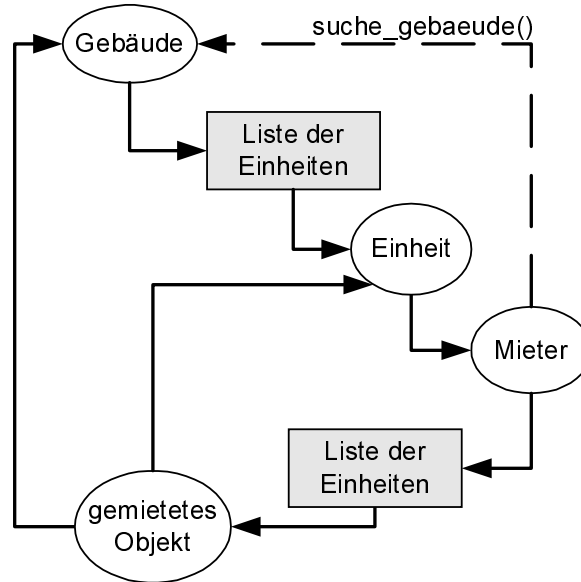


Abbildung A.10:  
Ein Objektmodell ist  
die Abbildung von  
Elementen und  
deren Beziehungen  
aus der Realwelt.

Ein Objektmodell ermöglicht dem Nutzer ein wesentlich intuitiveres Verständnis der abgebildeten Funktionalität als eine Menge »frei im Raum schwebender« Objekte. Komponenten, die eine Menge von Klassen darstellen, enthalten daher in der Regel ein oder mehrere Objektmodelle. Die grafische Darstellung eines Objektmodells in Form eines Objektdiagramms ermöglicht einem Menschen die schnelle Erfassung des Objektmodells. Sie werden in diesem Buch zahlreiche Objektdiagramme zur Veranschaulichung der vorgestellten COM-Komponenten finden. Die dazu verwendete Notation ist in Anhang B beschrieben.

Abbildung A.11:  
Beispiel für  
ein durch ein  
Objektdiagramm  
verbildlichtes  
Objektmodell



#### Mehrdeutigkeit des Begriffs Objektmodell

##### Objektmodell ist nicht gleich Objektmodell

Der Begriff Objektmodell kommt in der Informatik auch noch in einer anderen Bedeutung vor. In der Modellierung wird ein Objektmodell als eine abstrakte Beschreibung der verfügbaren Elemente betrachtet. Ein *Objektmodell auf Modellierungsebene* beschreibt also, dass es Komponenten, Klassen, Objekte, Attribute, Methoden, Assoziationen etc. gibt und in welchen Beziehungen diese Elemente zueinander stehen können. Diese Definition von Objektmodell ist gemeint, wenn man von dem *Object Model der Object Management Group* oder dem *Component Object Model* spricht (vgl. z.B. [SHW97], S. 283ff.). In Abgrenzung dazu kann man den Objektmodellbegriff, wie er von Komponentenherstellern verwendet wird, als ein *Objektmodell auf Anwendungsebene* bezeichnen. In der Begrifflichkeit von Microsoft besteht eine Inkonsistenz, weil die Verwendung des Begriffs Objektmodell im Begriff *Component Object Model* nicht zu der Verwendung dieses Begriffs in den konkreten Microsoft-Komponenten passt.

### A.1.7 Polymorphismus

**Polymorphismus** »Polymorphie (Vielgestaltigkeit) ist die Eigenschaft einer Variablen, für Objekte verschiedener Klassen stehen zu können.« ([BLA97], S. 425)

»Polymorphismus meint, dass sich die gleiche Operation in unterschiedlichen Klassen unterschiedlich verhalten kann.« ([RUM93], S. 2)



## Objektorientierung

Dabei kann die Variable auf all jene Nachrichten reagieren, die das Objekt, für das die Variable zu einem bestimmten Zeitpunkt steht, unterstützt. Die Implementierung, die durch eine konkrete Nachricht aufgerufen wird, ist über die Lebensdauer der Variablen nicht die gleiche; sie kann sich durch den Verweis auf ein anderes Objekt ändern.

Man nennt zwei Objekte polymorph hinsichtlich einer Nachricht, wenn sie beide auf diese Nachricht reagieren können. Sofern zwei Objekte die gleiche Schnittstelle besitzen, sind sie hinsichtlich aller zu dieser Schnittstelle gehörenden Mitglieder polymorph. Instanzen einer Unterklasse sind hinsichtlich aller geerbten Mitglieder zu den Instanzen der Oberklasse polymorph.

### Beispiele

Ein Beispiel aus dem Bereich der Gebäude wäre eine Methode `NeuerMieter()`, die sowohl in der Klasse `Wohngebäude` als auch in der Klasse `Bürogebäude` angeboten wird. Sofern eine Variable für ein Gebäude steht, wird in Abhängigkeit davon, um welchen Typ von Gebäude es sich handelt, entweder `Wohngebäude::NeuerMieter()` oder `Geschäftsgebäude::NeuerMieter()` aufgerufen. Dieser Polymorphismus beruht auf der Existenz einer gemeinsamen Oberklasse `Gebäude`.

#### Beispiel 1

Ein Beispiel aus dem Bereich der Betriebssysteme wären zwei Klassen `User` und `Group`, die insofern polymorph sind, als sie beide eine Schnittstelle `IGrunddaten` anbieten. `IGrunddaten` definiert ein Attribut `Name` und eine Methode `Deaktivieren()`. Es ist somit möglich, in einer Schleife über alle Elemente der Benutzerdatenbank auf die Namen der einzelnen Einträge zuzugreifen, ohne dabei eine Fallunterscheidung durchzuführen, um welche Art von Objekt es sich handelt. Das würde in Visual Basic so aussehen:

#### Beispiel 2

```
For Each obj in BenutzerDatenbank
 obj.Deaktivieren()
 MsgBox obj.Name & " wurde deaktiviert!"
Next
```

## A.1.8 Dynamische Bindung

»Unter Bindung versteht man die Verknüpfung eines Prozedur- oder Methodenaufrufs mit dem aufrufenden Code. In nicht-objektorientierten Sprachen kann diese Verknüpfung statisch vom Compiler oder Binder vorgenommen werden, so dass sich die Bindungen zur Laufzeit nicht mehr ändern. In objektorientierten Sprachen kann eine Variable wegen der Polymorphie für Objekte verschiedener Klassen stehen. Deshalb muss die Bindung dynamisch erfolgen, also zur Laufzeit.« ([BLA97], S. 428)

#### Dynamisches versus statisches Binden

Die dynamische Bindung ist die notwendige Voraussetzung für die Umsetzung des Polymorphismus.



## A.2 Komponentenarchitekturen

»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)

**Componentware** Betriebssysteme und Anwendungen waren bisher monolithisch aufgebaut. Mit dem Ansatz der *Componentware* werden sie in kleine(re) Einheiten (Komponenten) aufgeteilt, die einzeln oder in Zusammenarbeit mit anderen Komponenten einzelne Dienste erbringen. Das Prinzip der komponentenbasierten Softwareentwicklung besteht darin, Softwaresysteme aus voneinander unabhängigen Softwarebausteinen zusammenzusetzen. Diese Softwarebausteine können Produkte verschiedener Hersteller sein. Anwendungen werden somit nicht mehr von Grund auf neu entwickelt, sondern aus vorgefertigten Komponenten zu einem Endprodukt zusammengesetzt. Man spricht in diesem Zusammenhang auch von einem Plug&Play-Prinzip für Software. Componentware wird daher als der Schlüssel zur »Industrialisierung« der Softwareentwicklung bezeichnet.

Der Begriff Komponente wird für sehr viele unterschiedliche Dinge verwendet. In diesem Buch ist mit *Komponente* stets ein Softwarebaustein im objektorientierten Sinne gemeint. Eine objektorientierte Komponente enthält Klassen und bildet zur Laufzeit die Umgebung für die Instanzen dieser Klassen. Man sagt, dass eine bestimmte COM-Komponente ein bestimmtes Objektmodell *implementiert* oder *realisiert*.

### Abgrenzungen zu Objekten

#### Objekt versus Komponente

»Komponenten sind trotz zahlreicher Gemeinsamkeiten mit Objekten im Sinne der objektorientierten Programmierung von diesen erstens in Bezug auf die Größe abzugrenzen: Komponenten sind größere Bausteine, bestehen meist aus einer Menge von Objekten. Zweitens ist mit Komponenten und Objekten ein anderer Zweck verbunden; insbesondere sind Komponenten häufig in ihrer Funktionalität für den Anwender sichtbar. [...] Adressat von Objekten ist auf Grund ihrer feineren Granularität eher der Entwickler von Nicht-Standardlösungen. Schließlich muss drittens die Realisierung von Komponenten nicht notwendigerweise objektorientiert erfolgen, eine Komponente muss nur eine dem objektorientierten Paradigma folgende Schnittstelle besitzen.« ([EIC99], S. 363)

#### Die Grenzen von Plug&Play

Plug&Play bedeutet jedoch nicht, dass es für den Nutzer von Softwarekomponenten nichts mehr zu tun gäbe. Drei Aspekte sind – je nach Anwendungsfall in unterschiedlichem Grade – weiterhin von ihm zu leisten:

- Konfiguration**
- **Installation und Konfiguration der Komponenten:** Nach der Installation muss das Umfeld der Komponente konfiguriert werden. Dazu gehören insbesondere die Zugriffsrechte auf die Komponente und die Ressourcen, die die Komponente verwenden darf.

- **Parametrisierung der Komponenten:** Ähnlich wie Standardsoftware wird in der Regel auch eine Sammlung von Standardkomponenten nicht alle individuellen Ansprüche des Kunden erfüllen. Daher müssen die Komponenten mit Hilfe der bereitgestellten Schnittstellen parametrisiert werden. Insbesondere sind Standardwerte festzulegen.
- **Verbindung der Komponenten:** Die einzelnen Komponenten müssen durch verbindenden Programmcode (engl. Glue Code) integriert werden. Damit eine Integration von Komponenten unterschiedlicher Hersteller möglich ist, müssen Komponenten über eine standardisierte Schnittstelle verfügen. Wenn Softwarehersteller nicht nur einzelne Komponenten mit Teilfunktionalitäten, sondern ein komplettes Komponenten-Bundle für eine bestimmte Problemlösung vertreiben, liefern sie oft auch Glue Code mit. Aus der Sicht des Kunden ist die Bereitstellung einer Gesamtlösung aus Komponenten und mitgeliefertem Standard-Glue-Code zu begrüßen. Sie hat den Vorteil, dass ihm damit eine Test- und Lernumgebung für die Komponenten zur Verfügung steht. Immer häufiger werden dabei Skriptsprachen zur Verbindung von Komponenten eingesetzt. Diesem Trend folgt auch Microsoft mit der ActiveX Scripting-Architektur.

Parametrisierung

Glue Code

#### Komponente versus Klasse

Oft wird der Begriff *Komponente* mit dem Begriff *Klasse* gleichgesetzt. Jedoch sind diese Begriffe allenfalls dann gleichbedeutend, wenn eine Komponente nur eine Klasse enthält. Aber auch dann sollten Sie zwischen den beiden Konzepten sauber unterscheiden.

In der Regel besteht eine Komponente aus  $n$  Klassen. Eine Komponente ist zudem in der Regel die binäre Form einer Implementierung von Klassen. Verwirrung entsteht auch dadurch, dass manche von der Instanziierung von Komponenten sprechen und damit meinen, dass eine Komponente im Rahmen verschiedener Prozesse mehrfach in den Speicher geladen werden kann.

Leider macht Microsoft selbst in der Dokumentation und auch in Benutzer- sowie Programmierschnittstellen den Fehler, von Komponenten zu sprechen, wenn eigentlich von Klassen gesprochen werden müsste. Das ist nicht nur aus theoretischer Sicht unbefriedigend, sondern bringt auch in der Praxis viel Verwirrung mit sich.

Komponente  
versus Klasse

#### Schema einer Softwarekomponente

Jede Softwarekomponente im objektorientierten Sinn besitzt ein Schema. Ein Schema ist die Definition der in der Komponente enthaltenen Klassen und der Beziehungen zwischen den Klassen. Auch eine Schnittstellendefinition ist Teil des Schemas einer Komponente. Im Gegensatz zu einer Schnittstellendefinition beschreibt ein Schema eine Komponente vollständig.

Schema

Ein Schema definiert ein oder mehrere Objektmodelle. Dies gilt nicht in den Sonderfällen, dass eine Komponente nur aus einer Klasse besteht oder es keine Nutzungsbeziehungen zwischen den Klassen der Komponente gibt. Der Begriff Objektmodell wird häufig auch mit dem Schema einer Komponente gleichgesetzt, obwohl das Objektmodell eigentlich auch nur ein Teil des Schemas ist.

**Transparenz des Schemas**

Komponentenentwickler haben die Wahl, ob sie dem Komponentennutzer das Schema der Komponente (teilweise) offen legen oder aber es vollständig vor dem Benutzer verbergen. Im letzten Fall spricht man von einer Black-Box-Komponente. Eine Komponente, die ihr Schema offen legt und damit semantische Informationen über ihre Arbeit preisgibt, ist eine Grey-Box-Komponente. Sofern der komplette Quellcode der Komponente für den Nutzer einsehbar ist, handelt es sich um eine White-Box-Komponente (in Anlehnung an [BÜC97]). Die Offenlegung des Schemas ist jedoch nur eine Form der semantischen Beschreibung einer Grey-Box-Komponente. Ein transparentes Schema wird zweckmäßigerweise in der Regel selbst wieder objektorientiert in Form einer Objekthierarchie abgebildet.

**Middleware**

**Verteilungsplattform**

Eine Middleware (auch: Verteilungsplattform, Verteilungsinfrastruktur) ist eine Architektur zur Überbrückung von Rechengrenzen und zum Aufbau eines verteilten Systems. Die wichtigste Anforderung an eine Middleware ist die (Verteilungs-)Transparenz. Ein Benutzer (egal ob Mensch oder Anwendung) soll entfernte Ressourcen auf die gleiche Weise benutzen können wie lokale. Dazu muss die Middleware von den komplexen internen Aufgaben abschirmen, die nötig sind, um Verteiltheit zu überwinden. Middleware soll auch die Interaktion zwischen Anwendungskomponenten auf heterogenen Systemen unterstützen. In dieser Definition wurde bewusst der allgemeine Begriff »Ressource« verwendet, um von der Frage zu abstrahieren, ob es sich dabei um Objekte oder Komponenten oder nur um einfache Programmroutinen oder Daten handelt.

**Definition**

Heute verfügen alle wichtigen Komponentenarchitekturen auch über eine Middleware und sind daher zum Aufbau verteilter Systeme geeignet.

Österle definiert Middleware folgendermaßen: »Zusammenfassend gesagt ist Middleware eine Softwareschicht, welche auf Basis standardisierter Schnittstellen und Protokolle Dienste für eine transparente Kommunikation verteilter Anwendungen bereitstellt. Middledienste stellen eine Infrastruktur für die Integration von Anwendungen und Daten in einem heterogenen und verteilten Umfeld zur Verfügung.« ([ÖST96], S. 28). »Middleware ist im Rahmen des ISO/OSI-Referenzmodells für Rechnerkommunikation in offenen Systemen den anwendungsorientierten Schichten (Ebene 5-7) zuzuordnen.« ([ÖST96], S. 27)

Eine Komponente wird im Bereich der Componentware auch als Server bezeichnet, da sie einen bestimmten Dienst bereitstellt. Analog dazu ist der Nutzer einer Komponente ein Client. Diese Bezeichnungen sind unabhängig davon, ob die Komponenten sich auf demselben oder auf verschiedenen Computern befinden, und abhängig davon, welche Rolle der Computer in einem Client-/Server-Netzwerk besitzt. So kann auch auf einer Windows NT-Workstation eine Komponente ein Server für eine Komponente auf einem NT-Server sein.

Client und Server

## A.3 Verfügbare Komponentenarchitekturen

Es gibt inzwischen verschiedene Ansätze für Komponentenarchitekturen, die in der Praxis eingesetzt werden:

COM, CORBA, JAVA Beans

- ▶ das *Component Object Model (COM)* von Microsoft
- ▶ das *.NET-Framework* von Microsoft
- ▶ *Java Beans* und *Enterprise Java Beans (EJB)* von der Firma Sun
- ▶ die *Common Object Request Broker Architecture (CORBA)* der Object Management Group (OMG)

### Verbreitung der Komponentenarchitekturen

Das amerikanische Analyse-Unternehmen Cutter Consortium hat im Jahr 2000 die Verbreitung der verschiedenen Komponentenmodelle untersucht. Befragt wurden nordamerikanische und europäische Unternehmen. Paul Harmon dokumentiert in [CUT00], dass 68% der befragten Unternehmen Komponentenarchitekturen einsetzen. Die folgende Grafik zeigt die Verbreitung der verschiedenen Architekturen gemäß dieses Berichts. Das .NET-Framework wurde dabei nicht berücksichtigt, da es zum Zeitpunkt der Untersuchung noch nicht einmal eine Beta-Version der Software gab.

Umfrage

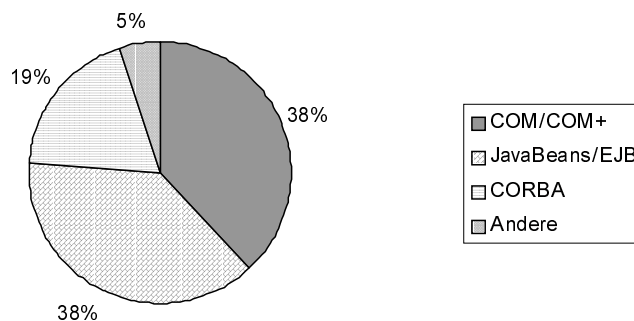


Abbildung A.12: Verbreitung der verschiedenen Komponentenarchitekturen gemäß [CUT00]

### Kriterien zur Bewertung der Komponentenarchitekturen

Zur Einstufung der Ansätze gibt es drei wichtige Kriterien:

- **Programmiersprachenabhängigkeit:** Ist die Architektur auf eine Sprache zugeschnitten oder können Komponenten, die in unterschiedlichen Sprachen erzeugt wurden, miteinander kommunizieren?
- **Plattformabhängigkeit:** Ist die Komponentenarchitektur grundsätzlich nur auf einem Betriebssystem lauffähig? Muss es für jede unterstützte Plattform eine eigene Binärform der Komponenten geben?
- **Herstellerbindung:** Wie sehr legt man sich bei der Verwendung der Komponentenarchitektur auf einen bestimmten Hersteller fest?

#### Verweis

Einen sehr ausführlichen Vergleich auf Basis zahlreicher Kriterien und unter Verwendung einer Beispielanwendung bietet das Buch von Gruhn/Thiel [GRU00].

### Bewertung der Komponentenarchitekturen

#### Programmiersprachenabhängigkeit

Während der Java Beans/Java Enterprise Beans-Ansatz programmiersprachenabhängig ist, sind COM, .NET und CORBA programmiersprachenneutrale Komponentenarchitekturen.

#### Plattformabhängigkeit

Der große Vorteil der Java-Ansätze ist jedoch die Plattformunabhängigkeit der Komponenten, die COM und CORBA hinsichtlich der Binärform der Komponenten nicht bieten. Immerhin gibt es inzwischen zahlreiche Portierungen der Basisdienste von COM und CORBA auf verschiedenen Betriebssystemplattformen. .NET ist grundsätzlich plattformunabhängig, es gibt aber bisher nur eine Implementierung für Microsoft Windows.

#### Herstellerbindung

Die Herstellerbindung (engl. Vendor Log-In) ist eindeutig bei COM/DCOM am größten. Allerdings unterstützen inzwischen sehr viele Windows-Softwarehersteller (z.B. Corel, Borland) und auch die Open Source-Szene (z.B. bei PHP4) dieses Komponentenmodell. Für .NET hat Microsoft von Beginn an die Unterstützung durch zahlreiche Hersteller als Werbeargument verwendet. Über die langfristige Unterstützung durch andere Hersteller lässt sich aber noch nichts sagen.

JavaBeans und EJB wurden im Schwerpunkt von Sun entwickelt, jedoch waren andere Hersteller auch daran beteiligt. Java Application Server für EJB gibt es von zahlreichen Herstellern. Da diese verschiedenen Implementierungen (nicht zuletzt durch proprietäre Erweiterungen) nicht vollständig kompatibel zueinander sind, begibt man sich auch mit dem Einsatz von EJB Application Servern in eine Abhängigkeit.

CORBA ist ein offener Standard, der aber noch mehr wie EJB darunter leidet, dass es zahlreiche verschiedene Hersteller mit inkompatiblen Implementierungen gibt.

# B Hinweise zum Buch

## B.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, Windows2000, Windows XP und Windows.NET. Windows2000, Windows XP und Windows.NET sind nur Marketing-Begriffe; das Betriebssystem heißt intern »Windows NT« (vgl. Registry-Schlüssel HKEY\_LOCAL\_MACHINE\SOFTWARE\MICROSOFT\WINDOWS NT\CURRENTVERSION). **NT**
- ▶ Windows2000 ist Windows NT 5.0.
- ▶ Windows XP ist Windows NT 5.1
- ▶ Der Begriff »Windows NT 4.0« (oder kurz »NT4«) umfasst dagegen Windows2000 und die Folgeversionen nicht.
- ▶ Windows 9x steht für Windows 95 und Windows 98. **Windows 9x**
- ▶ Windows ME steht für die Windows Millennium Edition.
- ▶ Bezüglich der Visual Basic-Sprachfamilie werden folgende Sprachregelungen angewandt:
  - ▶ Visual Basic (oder kurz »VB«) wird als Oberbegriff über die Sprachfamilie verwendet. **VB**
  - ▶ Die Bezeichnung *VB-Vollversion* wird für die kompilierungsfähige Programmiersprache im Rahmen von Visual Studio verwendet. VB6 bezeichnet davon die Versionen 6.0. **VB6**
  - ▶ Das Kürzel VB6/VBA (oder VB6/A) bedeutet: VB6 und Visual Basic for Applications. Dies ist sinnvoll, weil Aussagen für die VB-Vollversion häufig auch für VBA gelten. **VB6/A**
  - ▶ Die Ausdrücke Visual Basic Script, VBScript und VBS bezeichnen die Skriptsprache *Visual Basic Scripting Edition*. **VBS**
- ▶ 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**

**Automation  
versus Auto-  
matisierung**

► *Automation* und *Automatisierung*: Die Abgrenzung dieser beiden Begriffe ist bei vielen Autoren unklar. Einerseits meinen Autoren damit die Durchführung administrativer Aufgaben durch Programme, andererseits stehen die Begriffe für ein spezielles Verfahren innerhalb des Component Object Models (COM). Dabei sind diese Bedeutungen keineswegs gleichzusetzen. In diesem Buch wird das COM-Verfahren stets *Automation*, der allgemeine Begriff jedoch *Automatisierung* genannt. Zwar ist COM-Automation ein Hilfsmittel zur Automatisierung, aber keineswegs das einzige.

► Methodennamen werden stets – unabhängig von der Parameteranzahl – durch ein Klammernpaar »()« kenntlich gemacht und damit von Attributnamen abgegrenzt. Bitte beachten Sie aber, dass Visual Basic die Verwendung von Klammern nur bei Funktionen und Methoden mit Rückgabewert zulässt. Im konkreten Anwendungsfall müssen Sie daher prüfen, ob Sie die Routine mit oder ohne Klammern aufrufen müssen.

**MINFU**

► MINFU steht für **MI**crosoft **N**omenclature **F**oul-Up und bezeichnet Fälle, in denen Microsoft Probleme mit der adäquaten Benennung der eigenen Produkte und Konzepte hat [ROL00]. Urheber dieses Begriffs ist der amerikanische Autor David S. Platt, der sich um die Aufnahme von MINFU als *Wort* in das Oxford English Dictionary (OED) bemüht. Jede Verwendung von MINFU bringt ihn ein Stückchen näher an die Verewigung im OED.

## B.2 Hinweise zur Formatierung des Textes

Quellcode sowie die Namen von Klassen, Schnittstellen, Attributen, Methoden und Ereignissen erkennen Sie an der nicht-proportionalen Schrift.

Syntaxbeschreibungen sind zusätzlich in einen Kasten eingerahmt.

In *kursiver nicht-proportionaler Schrift* finden Sie Textausgaben von Programmen.

*Kursiv* gesetzt sind Namen und Werte, sofern der Begriff aus mehreren Wörtern besteht und es zu Missverständnissen kommen könnte, welche Wörter zu dem Begriff gehören. Oft wird ein Begriff nur bei der ersten Verwendung kursiv geschrieben. **Fett** geschrieben sind Wörter, die besonders betont werden sollen.

In KAPITÄLCHEN finden Sie jegliche Form von Ressourcenpfaden, also Dateinamen, Dateipfade, URLs, Registry-Schlüssel und Moniker sowie alle Namen, die Bildelementen entsprechen (z.B. Registerkarten, Menüeinträge, Schaltflächen).



### Querverweise

Wichtige Hinweise und Einschübe sind durch einen grauen Kasten hinterlegt.

Zusätzlich werden vier Icons verwendet, um Ihre Aufmerksamkeit zu erwecken:

Warnung vor Bugs oder  
möglichen Schwierigkeiten



Interessante Hintergrund-  
informationen



Verweis auf die CD



Tipp



## B.3 Querverweise

Leider ist es möglich, dass einige Querverweise auf andere Kapitel in diesem Buch nicht stimmen. Dies ist nicht das Verschulden des Autors, sondern des Setzers, der nicht in der Lage war, die dynamischen Verknüpfungen aus dem ursprünglichen Buch »Windows- und BackOffice-Scripting« (vgl. Vorwort) zu übernehmen. Da alle Querverweise »hardcodiert« waren, führte jedes zugefügte Kapitel, jedes zugefügte Listing, jede hinzugefügte Tabelle und jede zusätzliche Abbildung dazu, dass die Nummerierung nicht mehr stimmt. Autor und Verlag haben sich bemüht, alle durch den Satz verursachten Fehler manuell zu beheben. Soweit möglich, wurden die Nummern durch relative Verweise (»siehe nächste Abbildung«, »siehe die folgenden beiden Tabellen«) ersetzt, um dem Nummerierungsproblem bei der nächsten Auflage dieses Buchs aus dem Weg zu gehen. Dennoch ist leider nicht auszuschließen, dass es fehlerhafte Querverweise in diesem Buch gibt, da es erfahrungsgemäß trotz ordnungsgemäßer Manuskriptdateien und mehrmaliger manueller Nachbesserung der Satzfarben durch den Autor in der Endproduktion zu Fehlern bei den Verweisen kommt.

## B.4 Konventionen in den Code-Beispielen

Bei der Erstellung der Code-Beispiele wurden einige Konventionen verwendet, die nachfolgend dokumentiert sind.

### Umgebung für die Beispiele

- Kapitel 1,2,27** Kapitel 1, 2 und 27 enthalten keine Code-Beispiele.
- Kapitel 3** Die Code-Beispiele aus Kapitel 3 sind – sofern nicht anders erwähnt – Beispiele für Visual Basic 6.0 und innerhalb der Visual Studio 98-Entwicklungsumgebung lauffähig und kompilierbar.
- Kapitel 4 bis 26** Die im Buch abgedruckten Beispiele aus den Kapiteln 4 bis 26 sind – sofern nicht anders erwähnt – als Skript in der Sprache VBScript für den Windows Script Host (WSH) geschrieben. Dies ist der kleinste gemeinsame Nenner in der Visual Basic-Welt, da nicht alle Nutzer Visual Studio 6.0 oder ein Microsoft Office-Produkt mit integriertem Visual Basic for Applications (VBA) zur Verfügung haben. VBScript und der WSH werden seit Windows2000 mit dem Betriebssystem ausgeliefert. Für alle anderen Windows-Betriebssysteme sind VBScript und der WSH als kostenloses Add-on erhältlich [MSS00].

### Variablennamen

- Variablennamen** Aus Platzgründen wurde in den Listings in diesem Buch auf lange Variablennamen verzichtet. Die Listings sind in der Regel so kurz, dass dies nicht zu Nachteilen bei der Lesbarkeit führt. Für Laufvariablen wurden einbuchstabile 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*.

### Deklarationen und Typisierung

- Deklarationen und Datentypen** Variablendeklarationen erfolgen grundsätzlich ohne Typdefinitionen, da dies in VBS nicht unterstützt wird. Oft sind aber die Klassen- bzw. Schnittstellennamen in einem Kommentar angegeben, um Ihnen die Entwicklung in VB6/VBA zu erleichtern. In den meisten Listings sind die Variablen mit *Dim* deklariert, und es ist als Kommentar der Datentyp in der VB6/VBA-Form (*»As xy«*) angegeben. Sie können zwar diese Typisierung in VBScript nicht nutzen, jedoch unterstützt der Hinweis auf den Datentyp das Verständnis der Listings. Außerdem haben Sie es so leichter, die Beispiele der Buch-CD in typisierten Umgebungen wie VB6 und VBA zu verwenden.

## Konstanten

In der Regel werden in den Listings numerische Konstanten (also Zahlen, z.B. 4) verwendet und die zugehörigen symbolischen Konstanten (also Zeichenketten, z.B. `vbRetry`) nur in Klammern angegeben. Dies ermöglicht, dass die Beispiele auch ohne Einbindung einer Typlibibliothek lauffähig sind, zumal noch nicht alle Active Scripting Hosts die Einbindung von Typlibibliotheken erlauben. Alternativ werden Konstanten im Listing definiert:

```
Const vbRetry = 4
```

Konstanten

## Ausgaben mit say()

In fast allen Listings wird als Ausgaberroutine `say()` verwendet. `Say()` tritt an die Stelle der Ausgaberroutine der jeweiligen Ablaufumgebung (ein Active Scripting Host, VBA, VB6 etc.). `Say()` ist in der Skriptdatei `WS_VBWSHLIB.VBS` [CD:/code/\_bibliotheken/WS\_vbwshLIB.vbs] so implementiert, dass die Routine selbst erkennt, ob sie im Windows Script Host oder in VB6/VBA läuft. Wird das Objekt `WScript` (das Intrinsic Object des WSH) nicht gefunden, nimmt die Routine an, dass Sie in VB6/VBA läuft und bedient sich des dort vorhandenen Intrinsic Objects `Debug`, um eine Ausgabe in das Direktfenster der Entwicklungsumgebung zu schreiben. Sie können diese Ausgaberroutine leicht auf Ihre Umgebung anpassen. Die Verwendung von `MsgBox()` ist nicht empfehlenswert, da einige Beispiele viele Ausgaben machen und Sie dann sehr viele einzelne Dialogboxen erhalten werden.

Ausgabe wahlweise in WSH und VB6

```
' ### Universelle Ausgabe eines elementaren Datentyps
Sub say(s)
 If IsObject(Wscript) Then
 Wscript.echo s
 Else
 Debug.Print s
 End If
End Sub
```

*Listing 2.1: Universelle Ausgaberroutine say()*  
[CD:/code/\_bibliotheken/WS\_vbwshLIB.vbs]

Die Datei `WS_VBWSHLIB.VBS` enthält noch einige andere Kapselungen von Unterschieden zwischen dem WSH und VB6/VBA. Dies vereinfacht den Austausch von Quellcode zwischen dem WSH und VB6/VBA.



## Kommentare

Für die Kommentarzeilen gibt es eine bestimmte Notation, die in der folgenden Tabelle dokumentiert ist.

Tabelle B.1:  
Kommentarzeichen

| Kommentarzeichen | Bedeutung                                                                                                              |
|------------------|------------------------------------------------------------------------------------------------------------------------|
| ' ===            | Kommentar zu einem eigenständigen Beispiel                                                                             |
| ' ###            | Kommentar zu einer Hilfsroutine, die von einem eigenständigen Beispiel oder einer anderen Hilfsroutine aufgerufen wird |
| ' ---            | Zentraler Kommentar innerhalb einer Routine                                                                            |
| ' %%%            | Beginn einer Liste von Konstantendefinitionen                                                                          |
| '                | Sonstige Kommentare innerhalb einer Routine                                                                            |

## B.5 Die Umgebung für dieses Buch

### Aufbau der Testumgebung

Viele administrative Routinen benötigen Informationen über ihr Arbeitsumfeld wie zum Beispiel Servernamen und Anmeldeinformationen. Die Routinen in diesem Buch bzw. auf der CD zu diesem Buch enthalten die konkreten Namen für die Rechnerinfrastruktur, in der diese Routinen entwickelt und getestet wurden. Für den Test der Routinen haben Sie grundsätzlich zwei Möglichkeiten:

- Nachbau der Testumgebung oder
- Anpassung der Umgebungsinformationen in den Routinen

Der Nachbau der Testumgebung ist in der Regel aufwendig und daher eher ungeeignet. Die Anpassung der Routinen ist problemlos. Die folgende Beschreibung der Testumgebung hilft Ihnen zu erkennen, welche Angaben in den Routinen Schlüsselwörter sind und welche Eigennamen sind, die Sie ersetzen müssen. Wenn Sie aber sowieso gerade dabei sind, eine Testumgebung aufzubauen, erwägen Sie doch den Nachbau dieser Umgebung.

### Sprachversion

Bis auf wenige Ausnahmen, in denen deutsche Versionen nicht verfügbar waren, wurde die Software in einer lokalisierten deutschen Version verwendet.

Die Testumgebung besteht aus folgenden Geräten:

### Sonne

- Ein Windows NT4 Server mit dem Namen »Sonne«, der Domänencontroller für eine Domäne mit dem Namen »Sonnensystem« ist. In dieser Domäne gibt es drei Benutzer, die immer wieder in den Routinen vorkommen:
  - der Administrator mit dem Kennwort »ds9«
  - der Benutzer »HS«, der zur Domänen-Administratorengruppe gehört und das Kennwort »egal« hat
  - die Benutzer »Fox Mulder« und »Dana Scully«, die einfache Domänenbenutzer sind

Auf dem Server ist ein Microsoft Exchange Server Version 5.5 mit folgenden Parametern installiert:

- ▶ Organisationsname »Universum«
- ▶ Sitename »Sonnensystem«
- ▶ ein Client (Windows 98) mit Namen »Erde«
- ▶ ein Client (Windows2000) mit Namen »Mars«. Dort angeschlossen ist ein Drucker mit Namen »HP2100«
  - ▶ ein Client (Windows XP) mit Namen »Minbar«
- ▶ ein Windows2000 Server mit Namen »Sonne2000«. Auf dem Server ist ein Active Directory installiert mit der Domain »sonnensystem.it-visions.de«. In dieser Domäne gibt es eine Reihe weiterer Benutzer.

Auf Sonne2000 sind installiert:

- ▶ ein Microsoft SQL Server 7.0. Der Server hat für den Standardadministrator »sa« das Passwort »egal«.
- ▶ ein Microsoft Exchange Server 2000
- ▶ ein Microsoft Operations Manager 2000 Beta
- ▶ Ein Windows2000 Advanced Server mit Namen »Saturn«. Auf dem Server ist ein Active Directory installiert mit der Domain »komw2k.wi-inf.uni-essen.de«.

Auf Saturn sind installiert:

- ▶ ein Microsoft Exchange 2000 Server
- ▶ ein Microsoft SQL Server 2000. Der Server hat für den Standardadministrator »sa« das Passwort »egal«.
- ▶ Ein Windows Whistler Server Beta2 mit Namen »Narn«.

## **B.6 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.

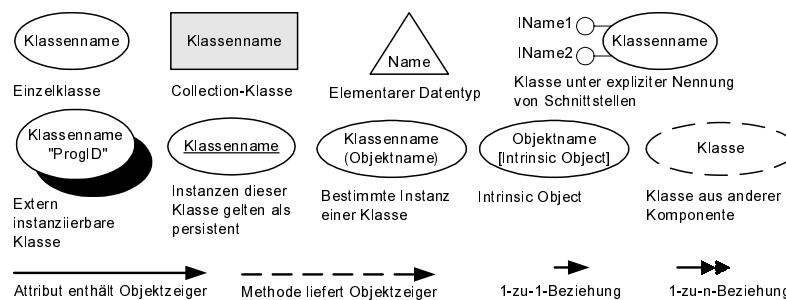
Da die Objektmodelle oft sehr umfangreich und die Navigationspfade komplex sind, beschränkt sich die Darstellung in der Regel auf die wichtigsten Klassen und deren Zusammenhänge.

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.

#### Eigene Notation

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.

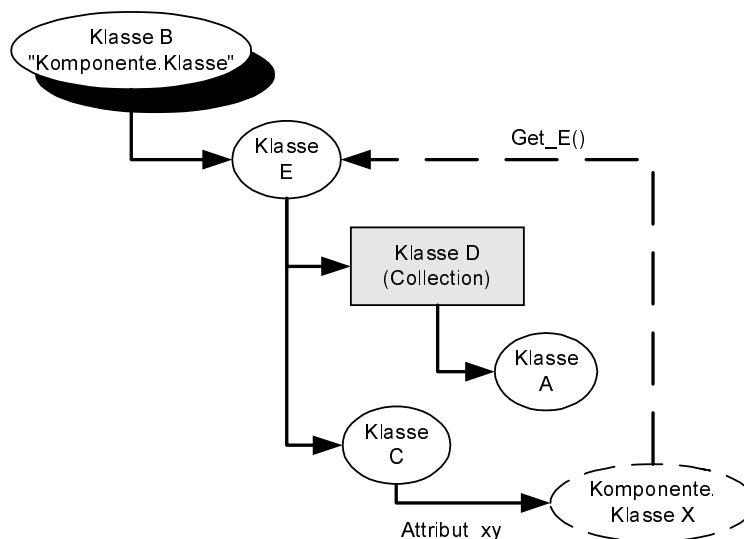
Abbildung B.1:  
Grafische Elemente  
zur Darstellung von  
Objektdiagrammen



#### 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 B.2:  
Beispiel für die grafische Darstellung eines Objektmodells in diesem Buch



### B.6.1 Knoten (geometrische Formen)

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
- ▶ Elementare Datentypen (z.B. String, Integer) sind Dreiecke

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.

### B.6.2 Zusätze

In den Knoten können folgende Zusätze vorkommen:

- ▶ Sofern ein Oval, ein Rechteck oder ein abgerundetes Rechteck eine gestrichelte Umrandung haben, bedeutet dies, dass die Klasse bzw. die Schnittstellen nicht in dieser, sondern in einer anderen Komponente definiert ist bzw. sind. Es gibt also eine Verbindung zu einem anderen Objektmodell. Der Name ist dann in der Form `Komponente.Klassenname` bzw. `Komponente.Klasse.Schnittstellename` angegeben.
- ▶ Eine mit einem Schatten hinterlegte Klasse ist eine von außen instanziierbare CoClass. Alle anderen Klassen können nur von anderen Klassen der Komponente erzeugt werden. **Schatten**
- ▶ Bei Klassen, deren Klassenname unterstrichen ist, ist es möglich, auf einzelne Instanzen direkt über COM-Moniker zuzugreifen. **Unterstreichung**
- ▶ Intrinsic Objects sind mit dem in eckigen Klammern stehenden Zusatz *[Intrinsic Object]* versehen. **Eckige Klammern**
- ▶ Namen in runden Klammern hinter dem Klassennamen bezeichnen eine einzelne, konkrete Instanz dieser Klasse (also ein Objekt) mit dem in runden Klammern angegebenen Namen. Der Name einer Instanz ist in der Regel nur dann angegeben, wenn es nur eine einzige Instanz dieser Klasse gibt. **Runde Klammern**
- ▶ Namen in Anführungszeichen sind von dem Klassennamen abweichende ProgIDs, die im HKEY\_CLASSES\_ROOT registriert sind. **Anführungszeichen**
- ▶ Sofern alle Mitglieder der Klasse über eine Standardschnittstelle mit IDispatch-Unterstützung angeboten werden, sind Schnittstellen nicht explizit eingezeichnet. In einigen wenigen Fällen, in denen die Mehrfachschnittstellen auch bei der Skriptprogrammierung von Bedeutung sind, sind diese in der üblichen Bullet&Stick-Form an Klassen angehängt. **Bullet&Stick**

- Icons** ► Die Objektdiagramme von Verzeichnisdienst-Objektmodellen sind mit zusätzlichen Icons versehen, die nur der Illustration dienen und keine inhaltliche Bedeutung haben.



In einigen Fällen sind in den Objektdiagrammen Klassen eingezeichnet, obwohl es sich auf der Implementierungsebene um Schnittstellen von ein und derselben Klasse handelt. Aus der Sicht von Visual Basic und den Skriptsprachen gibt es jedoch keine Schnittstellen, sondern nur Klassen. Da dieses Buch sich aber auf das *Rapid Application Development (RAD)* konzentriert, mögen die C++-Programmierer es verzeihen, wenn die Dokumentation der Objektmodelle an einigen Stellen nicht ihren Bedürfnissen entspricht.

### B.6.3 Kanten (Verbindungslinien)

**Kanten** 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.

**Attributzugriff  
versus Methoden-  
zugriff**

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:

- 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 einem Rechteck zu einem Dreieck bedeutet, dass eine Collection aus Werten eines elementaren Datentyps besteht.



### Grafische Notation in den Objektdiagrammen

- 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.

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.

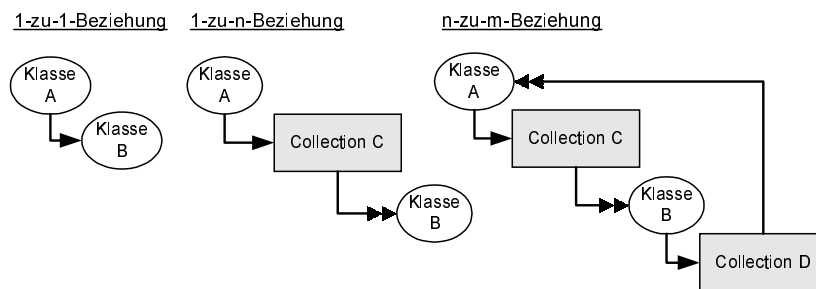


Abbildung B.3:  
Modellierung der  
Nutzungsbeziehungen  
mit verschiedenen  
Kardinalitäten  
in COM

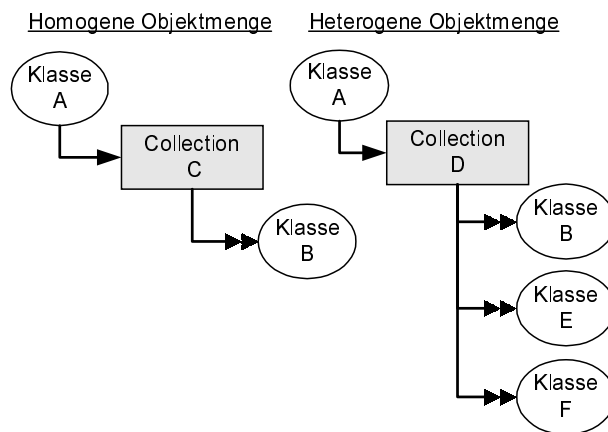


Abbildung B.4:  
Unterscheidung  
zwischen homogener  
und heterogener  
Collection

### Nutzungsbeziehungen auf Basis von Methoden

- Gestrichelte Linien** 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.
- 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 beschritten 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

- Gabelung** Eine sich gabelnde Linie bedeutet, dass das Attribut bzw. die Methode Instanzen verschiedener Klassen zurückliefern kann.
- 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 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.

## B.7 Komponenten-Schnellinfo

In diesem Buch finden Sie zu jeder besprochenen Komponente ein sogenanntes *Schnellinfo*. Dies gibt Ihnen die wichtigsten Informationen zu der jeweiligen Komponente auf einen Blick.

Tabelle B.2:  
Aufbau eines  
Komponenten-  
Schnellinfos

|                                  |                                                                                                                                                       |
|----------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name und Abkürzung</b>        | Hier stehen der Name und die Abkürzung, mit dem bzw. der die Komponente üblicherweise in der Dokumentation und in Fachkreisen bezeichnet wird.        |
| <b>Name der Komponentendatei</b> | Name der DLL oder EXE, die die Komponente implementiert. Sofern die Typbibliothek nicht Teil der Komponente ist, steht ihr Name in Klammern dahinter. |

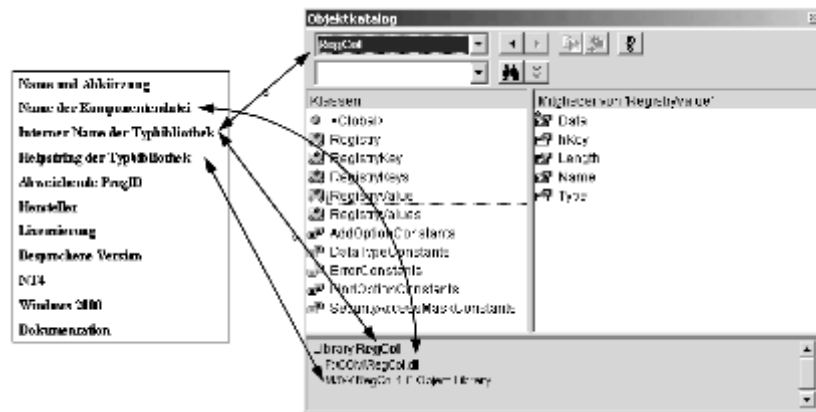
### Komponenten-Schnellinfo

|                                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Interner Name der Typlibibliothek</b> | Hier steht der Name der Typlibibliothek, wie Sie ihn im Microsoft-Objektkatalog finden und wie Sie ihn auch zur Instanziierung in der Visual Basic-IDE verwenden.                                                                                                                                                                                                                                                                                       |
| <b>Helpstring der Typlibibliothek</b>    | Hier steht der ausführliche Name der Typlibibliothek (der in der IDL als »Helpstring« angegeben wird). Dies ist der Name, der in den »Verweise«-Dialogen der Entwicklungsumgebungen erscheint.                                                                                                                                                                                                                                                          |
| <b>Abweichende ProgID</b>                | (Optionale Angaben) Normalerweise setzt sich die ProgID aus dem Namen, der als interner Name der Typlibibliothek verwendet wird, und dem Klassennamen der jeweiligen instanzierbaren Klasse zusammen. Nur wenn die ProgIDs der (Stamm-)Klassen nicht dem in der Typlibibliothek verwendeten Komponenten- bzw. Klassennamen entsprechen, nennt Ihnen dieser Eintrag die abweichenden ProgIDs.                                                            |
| <b>Hersteller</b>                        | Name des Herstellers                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Lizenzierung</b>                      | z.B. Freeware, Shareware, kostenloses Add-on, kommerzielle Komponente                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>Besprochene Version</b>               | Diese Information sagt Ihnen, welche Version der Komponente in diesem Buch besprochen wird.                                                                                                                                                                                                                                                                                                                                                             |
| <b>Windows xx</b>                        | Installationshinweise für die entsprechende Windows-Version                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Dokumentation</b>                     | Hier finden Sie den wichtigen Hinweis auf die Dokumentation der Komponente. Bei Microsoft-Komponenten finden Sie in der Regel einen Verweis auf die Microsoft Developer Network (MSDN) Library. Da sich diese in letzter Zeit häufig geändert hat, wurde in dieser Überarbeitung des Buchs auf die Angabe von URLs in die MSDN Library verzichtet. Die Eingabe des Komponentennamens in den Suchdialog wird Sie aber schnell zum gesuchten Ziel führen. |

*Tabelle B.2:  
Aufbau eines  
Komponenten-  
Schnellinfos  
(Forts.)*

Sofern die Komponente auf der Buch-CD enthalten ist, finden Sie einen entsprechenden Verweis bei den Installationshinweisen.

Abbildung B.5:  
Entsprechung zwischen dem Komponenten-Schnellinfo  
in diesem Buch und  
der Anzeige im  
Microsoft-Objekt-  
katalog (VB6)



# CD-ROM und Website

## C.1 Der Inhalt der CD-ROM

Die diesem Buch beiliegende CD enthält folgende Verzeichnisse:

|                        |                                                                                                                                                                                                                                                                                                                                           |
|------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| \CODE                  | Das Verzeichnis enthält alle Programmbeispiele aus dem Buch, geordnet nach Kapiteln und Unterkapiteln.                                                                                                                                                                                                                                    |
| \INSTALL               | COM-Komponenten, Skriptsprachen und Werkzeuge -zum Teil als Vollversionen, zum Teil als Demo-Versionen.                                                                                                                                                                                                                                   |
| \KONSTANTENLISTEN      | Bei der Besprechung einiger Komponenten wird Bezug auf Konstantenlisten genommen, die sich aufgrund ihrer Größe nicht für den Abdruck im Buch geeignet haben. Diese finden Sie als Textdateien in diesem Verzeichnis.                                                                                                                     |
| \SDK                   | frei verfügbare Software Development Kits zu COM-Komponenten von Microsoft mit Dokumentation, Beispielen und Werkzeugen                                                                                                                                                                                                                   |
| \LINKS                 | Eine Sammlung von Websites zu den Themen COM und Scripting                                                                                                                                                                                                                                                                                |
| \UMGEBUNG              | Dieses Verzeichnis enthält Dateien, die von einigen Code-Beispielen als Arbeitsdateien verwendet werden. In den Code-Beispielen haben diese Dateien immer den Pfad D:\BUCH. Kopieren Sie die Dateien dieses Verzeichnisses vorzugsweise auch nach D:\BUCH auf Ihre Festplatte. Andernfalls müssen Sie die Pfade in den Skripten anpassen. |
| \WEBSITE_ZUM_BUCH      | 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.                                                                                                                                                                                            |
| \WEITERE INFORMATIONEN | Dieses Verzeichnis enthält einige Original-Dokumentationen sowie andere frei verfügbare Informationen.                                                                                                                                                                                                                                    |

## C.2 Die Code-Beispiele auf der CD-ROM

Alle größeren 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.

**Kapitel 1-3 und 27**

**Kapitel 1-3 und 27**

Die Listings aus Kapitel 3 finden Sie in verschiedenen VB6-Projektdateien. Der Pfad zur Projektdatei ist jeweils unter dem Listing angegeben. Kapitel 1, 2 und 27 enthalten keine Listings.

**Kapitel 4 bis 26**

**Kapitel 4 bis 26**

Die Beispiele für Kapitel 4 bis 26 finden Sie auf der Buch-CD-ROM unter /CODE/KOMPONENTEN in zwei Formen:

**WSH-Dateien**

► in Form von XML-strukturierten Skriptdateien für den Windows Script Host ab Version 2.0 (Dateiextension .wsf). Die Skriptdateien sind in zahlreiche Unterzeichnisse einsortiert, wobei der Verzeichnisname die Abkürzung der Komponente ist, zu der die Beispiele gehören (z.B. enthält das Verzeichnis /CODE/KOMPONENTEN/ADO Beispiele zu den ActiveX Data Objects). Innerhalb eines jeden Unterkapitels gibt es noch einen Ordner /HILFSROUTINEN, der die zu diesem Thema vorgestellten Hilfsroutinen in einer Datei mit dem Namen [KAPITELKURZNAME]\_HELPER.VBS enthält. Jedes Skript bindet diese Datei ein, unabhängig davon, welche Hilfsroutinen wirklich für das Skript verwendet werden. Außerdem bindet jedes Skript die Bibliotheken /CODE /\_BIBLIOTHEKEN/WS\_SCRIPTLIB.VBS und /CODE/\_BIBLIOTHEKEN/WS\_VBWSHLIB.VBS ein, die einige häufig verwendete Routinen enthalten.

**VB6-Dateien**

► in Form von Visual Basic 6-Modulen (Dateiextension .bas). Diese Module sind zusammengefasst zu dem Projekt /CODE/KOMPONENTEN/\_ALLE/BUCH.VBP.

**Ausnahme**

Einige Beispiele, die besondere Features des WSH oder des Internet Explorers nutzen (z.B. die Ereignisbehandlung), sind nicht in diesem VB-Projekt enthalten. Diese Beispiele liegen nur als Skriptdatei vor. Die Namen dieser Beispiele beginnen mit einem Unterstrich »\_«.

**Arbeit mit der VB6-Projektdatei**

Die Projektdatei enthält auch Verweise zu allen verwendeten Komponenten. Sie können die Beispiele mit Hilfe des Direktfensters einzeln starten. Wenn dabei der Fehler »Projekt oder Bibliothek nicht gefunden« auftritt, dann liegt dies daran, dass nicht alle in das Projekt eingebundenen Komponenten auf Ihrem System registriert sind. Diese Fehlermeldung heißt allerdings nicht, dass eine für das konkrete Beispiel notwendige Komponente fehlt: Es kann auch sein, dass irgendeine der eingebundenen Komponenten fehlt. Die Lösung: Deaktivieren Sie in diesem Fall unter PROJEKT/VERWEISE den Ver-

weis auf die nicht vorhandenen Komponenten. Diese sind mit »NICHT VORHANDEN:« eindeutig gekennzeichnet.

### Keine kompilierten Projekte

Die VB6-Projektdateien aus Kapitel 3 bis 26 sind in der Regel nicht in kompilierter Form auf der CD-ROM, da die Ausgaben der Beispiele in das VB6-Direktfenster gehen und beim Start in kompilierter Form nichts zu sehen wäre.

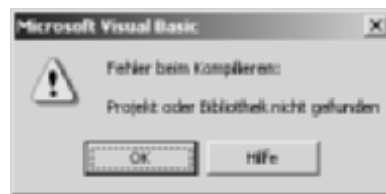


Abbildung C.1:  
Fehlermeldung in  
der VB6-Entwick-  
lungsumgebung,  
wenn irgendeine  
eingebundene  
COM-Komponente  
fehlt

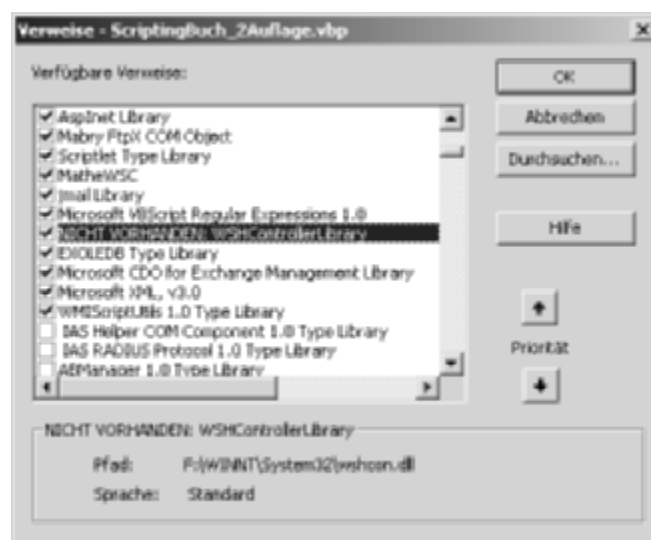


Abbildung C.2:  
Anzeige der fehlen-  
den COM-Kompo-  
nenten im Verweise-  
Dialog

## C.3 Die Website zu diesem Buch

Zu diesem Buch gibt es eine eigene Website:

[HTTP://WWW.COM-KOMPONENTEN.DE](http://www.com-komponenten.de)

Sie als Leser haben neben den öffentlichen Bereichen auch die Möglichkeit, auf einen geschützten Bereich zuzugreifen, der besondere Informationen enthält:

Weitere Informa-  
tionen und Unter-  
stützung im WWW

#### CD-ROM und Website

- ▶ **Skript-Archiv:** Die aktuellsten Versionen der in diesem Buch abgedruckten Skripte sowie weitere Skripte
- ▶ **Komponenten-Verzeichnis:** Überblick über COM-Komponenten, auch über solche, die nicht in diesem Buch beschrieben wurden
- ▶ **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.



# D Abkürzungs- verzeichnis

|         |                                                    |
|---------|----------------------------------------------------|
| 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 Tablegram                            |
| AKM     | Active Knowledge Module                            |
| ANSI    | American National Standards Institute              |
| API     | Application Programming Interface                  |
| APPID   | Application Identifier                             |
| ASCII   | American Standard Code for Information Interchange |
| ASP     | Active Server Pages                                |
| ASP.NET | Active Server Pages.NET                            |
| ATL     | Active Template Library                            |
| AUO     | Active User Objects                                |
| BCL     | Base Class Library                                 |
| BIOS    | Basic Input/Output System                          |
| BLOB    | Binary Large Object                                |
| BOF     | Begin Of File                                      |
| 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                         |

### *Abkürzungsverzeichnis*

|         |                                             |
|---------|---------------------------------------------|
| DCO     | Domino Collaboration Objects                |
| CDOEX   | CDO 3.0 for Exchange 2000                   |
| CDOEXM  | CDO for Exchange Management                 |
| CDONTS  | CDO for NT Server                           |
| CDO SYS | 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                    |
| CLR     | Common Language Runtime                     |
| CLS     | Common Language Specification               |
| CLSID   | Class Identifier                            |
| CMIP    | Common Management Information Protocol      |
| CN      | Common Name                                 |
| COM     | Component Object Model                      |
| 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           |
| 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              |

### *Abkürzungsverzeichnis*

|       |                                                                  |
|-------|------------------------------------------------------------------|
| 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 oder<br>Distributed Transaction Coordinator |
| DTD   | Document Type Definition                                         |
| DTS   | Data Transformation Service                                      |
| ECMA  | European Computer Manufactures Association                       |
| EJB   | Enterprise Java Beans                                            |
| EOF   | End Of File                                                      |
| EOS   | End of Stream                                                    |
| ESATE | Exchange Script Agent Test Environment                           |
| EXE   | Executable (ausführbare Datei)                                   |
| 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                                              |
| GC    | Global Catalogue                                                 |
| GC    | Garbage Collector                                                |
| 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                                |

### Abkürzungsverzeichnis

|        |                                                |
|--------|------------------------------------------------|
| 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                       |
| IS     | Information Store                              |
| ISO    | International Organization for Standardization |
| JUMP   | Java User Migration Path to Microsoft .NET     |
| 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        |
| MIDL   | Microsoft Interface Definition Language        |
| MIME   | Multipurpose Internet Mail Extensions          |
| MINFU  | Microsoft Nomenclature Foul-Up                 |
| MMC    | Microsoft Management Console                   |
| 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                   |

### *Abkürzungsverzeichnis*

|         |                                             |
|---------|---------------------------------------------|
| NDR     | Network Data Representation                 |
| NDS     | Novell Directory Service                    |
| NetBIOS | NETwork 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                                |
| 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                           |
| 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               |

### *Abkürzungsverzeichnis*

|       |                                         |
|-------|-----------------------------------------|
| 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                |
| SID   | Security Identifier                     |
| SMS   | Systems Management Server               |
| SMTP  | Simple Mail Transfer Protocol           |
| 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               |
| TOM   | Text Object Model                       |
| T-SQL | Transaction SQL                         |
| UDA   | Universal Data Access                   |
| UDL   | Universal Data Link                     |
| UDP   | User Datagram Protocol                  |
| UMI   | Universal Management Interface          |
| UML   | Unified Markup Language                 |

### *Abkürzungsverzeichnis*

|          |                                                       |
|----------|-------------------------------------------------------|
| UNC      | Universal Naming Convention                           |
| UPN      | Umgekehrt polnische Notation oder User Principal Name |
| URI      | Uniform Resource Identifier                           |
| URL      | Uniform Resource Locator                              |
| URN      | Uniform Resource Name                                 |
| UCS      | Universal Character Set (»Unicode«)                   |
| 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 Application                          |
| VBS      | Visual Basic Script                                   |
| VBScript | Visual Basic Script                                   |
| VES      | Virtual Execution System                              |
| VOS      | Virtual Object System                                 |
| VTBL     | Virtual Table                                         |
| W3C      | World Wide Web Consortium                             |
| W3SVC    | Webservice                                            |
| WBEM     | Web Based Enterprise Management                       |
| WDM      | Win32 Driver Model                                    |
| WINS     | Windows Internet Naming Service                       |
| WKGUID   | Well Known Global Unique Identifier                   |
| WMI      | Windows Management Instrumentarium                    |
| 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      |

### *Abkürzungsverzeichnis*

|      |                                |
|------|--------------------------------|
| XMP  | Extended Management Packs      |
| XSD  | XML Schema Definition          |
| XSL  | Extensible Stylesheet Language |
| XSLT | XSL Transformation             |



# E Literaturverzeichnis

## E.1 Gedruckte Literatur

- [AHO96] Aho, A.; Ullman, J.D.: Informatik – Datenstrukturen und Konzepte der Abstraktion. Bonn: International Thomson Publishing, 1996.
- [APP99] Appleman, D.: COM/ActiveX-Komponenten mit VB6 entwickeln. München: Markt & Technik, 1999.
- [BEH98] Behme, H.; Mintert, S.: XML in der Praxis. Bonn: Addison Wesley Longman, 1998.
- [BLA97] Blaschek, G.: Objektorientierte Programmierung. In: Rechenberg, G.; Pomberger, P.: Informatikhandbuch. Wien: Hanser, 1997.
- [BÖH96] Böhm, R., et al.: Systementwicklung in der Wirtschaftsinformatik. Zürich: vdf-Verlag, 1996.
- [BOX98] Box, D.: COM: Microsofts Technologie für komponentenbasierte Softwareentwicklung. Bonn: Addison Wesley Longman, 1998.
- [BÜC97] Büchi, M.; Weck, W.: A Plea for Grey-Box Components. TUCS Technical Report No. 122, 1997.
- [CHA96] Chadwick, D.: Understanding X.500 – The Directory. Chapman & Hal, 1996.
- [CWO99] o.V.: »Visual Basic ist beliebt«. In: Computerwoche, Ausgabe vom 27.10.99.
- [EDD00] Eddon, G.; Eddon, H.: Inside COM+. Unterschleißheim: MS Press, 2000.
- [EIC99] Eicker, S.; Nietsch, M.: Standards zum objektorientierten Paradigma, Wirtschaftsinformatik, Heft 4/99.
- [FIS99] Fischbach, R.: Schwierige Abgrenzungen. Von den Job Control Languages bis Perl und Python. In: iX 12/99. S. 60.
- [GOO97] Goos, G.; Zimmermann, W.: Programmiersprachen. In: Rechenberg, G.; Pomberger, P.: Informatikhandbuch. Wien: Hanser, 1997.
- [GOO98] Goodman, D.: Dynamic HTML. The Definite Reference. Sebastopol: O'Reilly, 1998.
- [GRI98] Griffel, F.: Componentware. Heidelberg: dpunkt-Verlag, 1998.
- [GRM97] Grimmes, R.: Professional DCOM Programming, Birmingham: Wrox Press, 1997

### *Literaturverzeichnis*

- [GRU00] Gruhn, V.; Thiel, A: Komponentenmodelle. München: Addison Wesley, 2000.
- [HAH98] Hahn, S.: ADSI ASP Programmer's Reference. Birmingham: Wrox Press, 1998.
- [HOM99] Homer, A. et al.: Professional Active Server Pages 3.0. Birmingham: Wrox Press, 1999.
- [KIR98] Kirtland M.: COM+: Eine neue Umgebung für Komponenten. In: Microsoft System Journal 2/1998. S. 24 ff.
- [KOF98] Koffler, M.: Visual Basic 6.0. Bonn: Addison Wesley Longman, 1998.
- [KRA98] Krause, J.: Active Server Pages. Bonn: Addison Wesley Longman, 1998.
- [KRA99] Krause, J.: Microsoft Site Server 3.0. München: Addison Wesley Longman, 1999.
- [MIC98] o.V. (Hrsg: Microsoft Corporation): Microsoft Visual Basic 6.0 Komponentenhandbuch. Unterschleißheim: MSPress, 1998.
- [MIN97] Mintert, S.: Java Script 1.2. Bonn: Addison Wesley Longman, 1997.
- [NNI00] Nicoloudis, N. and Mingins, C.: Implementing Trading Object Services in COM. Proceedings of the 36th International Conference on Technology of Object-Oriented Languages and Systems (TOOLS-Asia'00)
- [OES97] Oesterreich, B.: Objektorientierte Softwareentwicklung. München, Wien: R. Oldenburg Verlag, 1997.
- [ÖST96] Österle, H.: Integration: Schlüssel zur Informationsgesellschaft, in: Österle, H. et al.: Middleware. Braunschweig: Vieweg, 1996.
- [PAR00] Pardi, J.W.: XML in Action. Redmond: MSPress, 2000.
- [PET99] Petovic, D.: Microsoft SQL Server 7.0. Bonn: Addison-Wesley-Longman, 1999.
- [PLA99] Platt, D.S.: COM+ verstehen. München: MS Press, 1999.
- [QUI94] Quibeldey-Cirkel, K.: Das Objekt-Paradigma in der Informatik. Stuttgart: Teubner, 1994.
- [RAS97] Rauh, O.; Stickel, E.: Konzeptuale Datenmodellierung. Stuttgart: Teubner, 1997.
- [RED97] Redmond, F.E.: DCOM. Foster City: IDG Books, 1997.
- [RIK98] Riehn, L.; Kattner, H.: Microsoft Exchange Server 5.5. Bonn: Addison Wesley Longman, 1998.
- [RUM93] Rumbaugh, J.: Objektorientiertes Modellieren und Entwerfen. München, Wien: Carl Hanser, 1993.

### *Literaturverzeichnis*

- [SAP99] Oleg Ovanesyan, O.: Professional Visual Basic SAP R/3 Programming. Birmingham: Wrox Press, 1999
- [SCH00a] Schwichtenberg, H.: Mit MTS zu COM+: Komponentendienste in NT4 und Windows2000. In: iX 02/00. S. 124.
- [SCH00b] Schwichtenberg, H.: Verzeichnisdienste verwalten mit ADSI. In: basicpro 1/00. S. 10.
- [SCH00c] Schwichtenberg, H.: Gefährliche Liebesgrüße: Windows Scripting-Viren: Inside LoveLetter. In: iX 6/00. S. 16ff.
- [SCH00d] Schwichtenberg, H.: Gezielter Zugriff: Skriptsteuerung des Windows 2000-Verzeichnisdienstes. In: iX 9/00. S. 110 ff.
- [SCH00e] Schwichtenberg, H.: Neuester Stand: Neuerungen in VBScript und JScript 5.x. In: iX 10/00, S. 118 ff.
- [SCH00f] Schwichtenberg, H.: Microsofts WBEM-Umsetzung: WMI. In: iX 11/00. S. 214 ff.
- [SCH00g] Schwichtenberg, H.: Radikale Emanzipation: Visual Basic .NET Preview. In: iX 12/00. S. 120
- [SCH01a] Schwichtenberg, H.: Fingerabdruck: Signierte Skripte im WSH 5.6. In: iX 2/01. S. 108ff.
- [SCH01b] Schwichtenberg, H.: Nachrichtenkontrolle: Exchange-2000-Webstore-Programmierung. In: iX 6/01. S. 124ff.
- [SCH99a] Schwichtenberg, H.: Objekte im Zugriff: ADSI-Programmierung. In: iX 2/99. S. 136.
- [SCH99b] Schwichtenberg, H.: Postfächer einrichten: Stapelverarbeitung für Exchange. In: iX 3/99. S. 138.
- [SCH99c] Schwichtenberg, H.: Druckkontrolle: Webbasierter Druckmanager. In: iX 7/99. S. 138.
- [SCH99d] Schwichtenberg, H.: Daten-Modellierung: Microsofts Data Transformation Service. In: iX 10/99. S. 137.
- [SCH99e] Schwichtenberg, H.: Bausteine: COM als Basic für NT-Skripts. In: iX 12/99. S. 66.
- [SHW97] Schwarze, J.: Einführung in die Wirtschaftsinformatik. Berlin, Herne: Verlag Neue Wirtschafts-Briefe, 1997.
- [SIR99] Sirotin, V.: ActiveX/DCOM-Programmierung mit Visual C++ 6. Bonn: Addison Wesley Longman, 1999
- [STA93] Stallings, W.: SNMP, SNMPv2 und CMIP. The Practical Guide to Network-Management Standards. Bonn: Addison Wesley Longman, 1993.
- [UNL95] Unland, R.: Objektorientierte Datenbanken – Konzepte und Modelle. Bonn: International Thomson Publishing, 1995.

- [WEG89] Wegner, P.: Learning the Language, 1989.
- [WEG90] Wegner, P.: Concepts and Paradigms of object-oriented programming. In: ACM OOPS Messenger, Band 1, Nr. 1, S. 7-87, 1990.
- [WES99a] Westphal, R.: Typelib-Konstanten auf der Spur. In: basicpro 5/99. S. 48.
- [WES99b] Westphal, R.: Set...=Nothing oder nicht? In: basicpro 6/99. S. 21.
- [WES99c] Westphal, R.: COM-Interface Casting. In: basicpro 6/99. S. 60.
- [WIS01] Schwichtenberg, H.: Windows-Scripting. München: Addison Wesley, 2001.

## E.2 Quellen im Internet



Diese Verweise ins Internet finden Sie auf der Buch-CD in Form einer HTML-Datei, in der Sie die Links nur anklicken müssen [CD:/weitere Informationen/links/quellen.htm].

### Links

|          |                                                                                                                           |                                                                           |
|----------|---------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------|
| [ACE00]  | <a href="http://www.digt.com/aces">http://www.digt.com/aces</a>                                                           | Die Firma ACES liefert das im Buch besprochene ACES WinNT Automation Kit. |
| [ACT00]  | <a href="http://www.activestate.com">http://www.activestate.com</a>                                                       | ActiveState ist der Hersteller von PerlScript.                            |
| [ADM01]  | <a href="http://www.admascripts.net">http://www.admascripts.net</a>                                                       | Skript-Archiv, zum Teil kommerziell                                       |
| [ADS00]  | <a href="http://www.microsoft.com/adsi">http://www.microsoft.com/adsi</a>                                                 | Shortcut zur offiziellen Microsoft-ADSI-Site                              |
| [AGR00]  | <a href="http://www.activex.org">http://www.activex.org</a>                                                               | Homepage der Active Group                                                 |
| [ASP00]  | <a href="http://www.asptoday.com">http://www.asptoday.com</a>                                                             | Alles rund um ASP                                                         |
| [ATX00]  | <a href="http://www.opengroup.org/comsource/techref2/TITLE.HTM">http://www.opengroup.org/comsource/techref2/TITLE.HTM</a> | ActiveX Core Technology Reference der Open Group                          |
| [BAK01a] | <a href="http://www.ddj.com/articles/2001/0165/0165h/0165h.htm">http://www.ddj.com/articles/2001/0165/0165h/0165h.htm</a> | Teil 1 des Interviews von Mark Baker mit Andrew Clinick                   |
| [BAK01b] | <a href="http://www.ddj.com/articles/2001/0165/0165j/0165j.htm">http://www.ddj.com/articles/2001/0165/0165j/0165j.htm</a> | Teil 2 des Interviews von Mark Baker mit Andrew Clinick                   |
| [BOR00]  | <a href="http://www.borncity.de">http://www.borncity.de</a>                                                               | WSH-Bazar von Günther Born                                                |

# Literaturverzeichnis

|          |                                                                                                                                                                   |                                                                                                                                 |
|----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------|
| [BOX01]  | <a href="http://www.develop.com/dbox/dotnet/clrscrip/">http://www.develop.com/dbox/dotnet/clrscrip/</a>                                                           | CLRHost von Don Box                                                                                                             |
| [CAW01]  | <a href="http://www2.hursley.ibm.com/rexx">http://www2.hursley.ibm.com/rexx</a>                                                                                   | Homepage des Erfinders von Rexx, Mike F. Cawlishaw, mit Links zu allem, was mit Rexx zu tun hat, auch Object Rexx               |
| [CDO00]  | <a href="http://www.cdolive.com">http://www.cdolive.com</a>                                                                                                       | Site zur Exchange- und Outlook-Programmierung                                                                                   |
| [CHI00]  | <a href="http://www.chilisoft.com">http://www.chilisoft.com</a>                                                                                                   | Anbieter eines Active Server Pages-kompatiblen Scripting Hosts »ChiliASP« für andere Webserver (auch Nicht-Windows-Plattformen) |
| [CLI00]  | <a href="http://msdn.microsoft.com/voices/scripting.asp">http://msdn.microsoft.com/voices/scripting.asp</a>                                                       | Kolumne von Andrew Clinick, Programmmanager in der Arbeitsgruppe Skripttechnologien bei Microsoft                               |
| [CLI01a] | <a href="http://msdn.microsoft.com/voices/scripting11132000.asp">http://msdn.microsoft.com/voices/scripting11132000.asp</a>                                       | Artikel zum WSH 5.6 von Andrew Clinick in seiner »Scripting Clinic«                                                             |
| [CLI01b] | <a href="http://msdn.microsoft.com/workshop/languages/clinic/scripting07142000.asp">http://msdn.microsoft.com/workshop/languages/clinic/scripting07142000.asp</a> | Andrew Clinick über die Zukunft der Skriptsprachen                                                                              |
| [CLI01c] | <a href="http://msdn.microsoft.com/workshop/languages/clinic/scripting01162001.asp">http://msdn.microsoft.com/workshop/languages/clinic/scripting01162001.asp</a> | Beitrag von Andrew Clinick in der Scripting Clinic über VSA                                                                     |
| [CMS00]  | <a href="http://www.componentsource.com">http://www.componentsource.com</a>                                                                                       | Sehr großer kommerzieller Anbieter von Komponenten                                                                              |
| [COM00]  | <a href="http://www.com-objekte.de">http://www.com-objekte.de</a>                                                                                                 | Deutsches Komponentenverzeichnis                                                                                                |
| [COS00]  | <a href="http://www.opengroup.org/comsource/">http://www.opengroup.org/comsource/</a>                                                                             | COMSource: COM-Referenzportierung der Open Group für Solaris und True64UNIX                                                     |
| [CRY01]  | <a href="http://support.microsoft.com/support/kb/articles/Q247/2/57.asp">http://support.microsoft.com/support/kb/articles/Q247/2/57.asp</a>                       | Informationen über die CryptoAPI-Tools                                                                                          |

*Literaturverzeichnis*

|          |                                                                                               |                                                                                           |
|----------|-----------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------|
| [CWA00]  | <a href="http://cwashtington.netreach.net">http://cwashtington.netreach.net</a>               | Umfangreiche Scripting Site von Clarence Washington, viele Komponenten                    |
| [DAR00]  | <a href="http://www.winscripiter.com">http://www.winscripiter.com</a>                         | Scripting Site von Daren Thiel                                                            |
| [DEJ00]  | <a href="http://www.deja.com">http://www.deja.com</a>                                         | Suchservice für (weit) zurückliegende Beiträge in Newsgroups                              |
| [DEV00]  | <a href="http://www.4developers.com">http://www.4developers.com</a>                           | Hersteller von Tools für die COM-Programmierung (u. a. Registry Crawler und COM-Explorer) |
| [DOB01]  | <a href="http://www.ddj.com/topics/altlang/">http://www.ddj.com/topics/altlang/</a>           | Dr. Dobb's Scripting Site                                                                 |
| [DOM00]  | <a href="http://www.w3.org/DOM/">http://www.w3.org/DOM/</a>                                   | Informationen zum Document Object Model                                                   |
| [DOT01a] | <a href="http://www.microsoft.com/net">http://www.microsoft.com/net</a>                       | Offizielle .NET-Site von Microsoft                                                        |
| [DOT01b] | <a href="http://www.gotdotnet.com/">http://www.gotdotnet.com/</a>                             | Weitere .NET-Site von Microsoft                                                           |
| [DUN00]  | <a href="http://www.dundas.com">http://www.dundas.com</a>                                     | Hersteller des Hyper-Hosts                                                                |
| [DVM00]  | <a href="http://www.developmen tor.com/soap">http://www.developmen tor.com/soap</a>           | SOAP-Site der Firma Developmentor                                                         |
| [DVX01a] | <a href="http://www.devx.com/dotnet/">http://www.devx.com/dotnet/</a>                         | DevX .NET Center                                                                          |
| [DYN01]  | <a href="http://www.innermedia.com">http://www.innermedia.com</a>                             | DynaZIP-AX                                                                                |
| [EYE00]  | <a href="http://www.microeye.com">http://www.microeye.com</a>                                 | Micro Eye Inc. ist Hersteller des Script Directors für Exchange.                          |
| [GRE00]  | <a href="http://www.swynk.com/friends/green/">http://www.swynk.com/friends/green/</a>         | DTS-Beispiele von Darren Green                                                            |
| [HAL00]  | <a href="http://www.halcyonsoft.com">http://www.halcyonsoft.com</a>                           | Anbieter des Active Server Pages-kompatiblen Scripting Hosts »InstantASP«                 |
| [HAS00]  | <a href="http://haskell.cs.yale.edu/haskellsript">http://haskell.cs.yale.edu/haskellsript</a> | Site zur Skriptsprache HaskellScript                                                      |
| [IAN00]  | <a href="http://wsh.glazier.co.nz">http://wsh.glazier.co.nz</a>                               | Bekannte Scripting-Site von Ian Morrish                                                   |

# *Literaturverzeichnis*

|          |                                                                                                                                                                                           |                                                                              |
|----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------|
| [IBM00]  | <a href="http://www-1.ibm.com/servers/eserver/series/clientaccess/oledb/">http://www-1.ibm.com/servers/eserver/series/clientaccess/oledb/</a>                                             | OLE DB-Provider für DB2/400                                                  |
| [IBM01]  | <a href="http://www-4.ibm.com/software/ad/obj-rexx">http://www-4.ibm.com/software/ad/obj-rexx</a>                                                                                         | Object REXX Homepage                                                         |
| [ISG00]  | <a href="http://www.isgsoft.com">http://www.isgsoft.com</a>                                                                                                                               | Hersteller von OLE DB-Providern                                              |
| [ITS01]  | <a href="http://www.intouchsoftware.com">http://www.intouchsoftware.com</a>                                                                                                               | Kleine amerikanische Firma, die verschiedene Werkzeuge als Freeware anbietet |
| [KNU01]  | <a href="http://home.knuut.de/heiko.rost/">http://home.knuut.de/heiko.rost/</a>                                                                                                           | MiniOLE                                                                      |
| [LDA00a] | <a href="http://www.openldap.org/">http://www.openldap.org/</a>                                                                                                                           | kostenlose LDAP-Implementierungen                                            |
| [LDA00b] | <a href="http://www.innosoft.com/ldapworld">http://www.innosoft.com/ldapworld</a>                                                                                                         | LDAP-Site der Firma Innosoft                                                 |
| [LOT01a] | <a href="http://www.lotus.com">http://www.lotus.com</a>                                                                                                                                   | Lotus Notes OLE Library                                                      |
| [LOT01b] | <a href="http://www.lotus.com/developers/devbase.nsf/homedata/homedco">http://www.lotus.com/developers/devbase.nsf/homedata/homedco</a>                                                   | Informationen zu den Domino Collaboration Objects                            |
| [LOT01c] | <a href="http://archives.groupcomputing.com/index.cfm?fuseaction=ListArticlesByTopic&amp;ID=33">http://archives.groupcomputing.com/index.cfm?fuseaction=ListArticlesByTopic&amp;ID=33</a> | Lotus Notes und COM                                                          |
| [LUA01a] | <a href="http://www.lua.org">http://www.lua.org</a>                                                                                                                                       | Offizielle Homepage der Sprache LUA                                          |
| [LUA01b] | <a href="http://www.tecgraf.puc-rio.br/lu">http://www.tecgraf.puc-rio.br/lu</a>                                                                                                           | Informationen zu LUAScript                                                   |
| [MAR00]  | <a href="http://www.mabry.com">http://www.mabry.com</a>                                                                                                                                   | Anbieter von COM-Komponenten                                                 |
| [MCC00]  | <a href="http://msdn.microsoft.com/code">http://msdn.microsoft.com/code</a>                                                                                                               | MSDN Code Center, Verzeichnis von Beispiel-Code                              |
| [MCO00]  | <a href="http://www.microsoft.com/com/">http://www.microsoft.com/com/</a>                                                                                                                 | Microsoft COM-Site                                                           |
| [MER00]  | <a href="http://www.merant.com">http://www.merant.com</a>                                                                                                                                 | Die Firma Merant ist ein bekannter Anbieter für OLE DB-Provider.             |
| [MER01]  | <a href="http://www.cs.mu.oz.au/research/mercury">http://www.cs.mu.oz.au/research/mercury</a>                                                                                             | Informationen über die Sprache Mercury                                       |

*Literaturverzeichnis*

|          |                                                                                                                                                                                                                 |                                                                                       |
|----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------|
| [MKS00]  | <a href="http://www.datafocus.com">http://www.datafocus.com</a>                                                                                                                                                 | Perl-Implementierung von MKS                                                          |
| [MMS01]  | <a href="http://www.microsoft.com/windows2000/management/">http://www.microsoft.com/windows2000/management/</a>                                                                                                 | Microsoft Management Strategy Portal zum Thema Systemmanagement                       |
| [MOM01]  | <a href="http://www.microsoft.com/mom">http://www.microsoft.com/mom</a>                                                                                                                                         | Microsoft-Site zum Microsoft Operations Manager                                       |
| [MSD01a] | <a href="http://msdn.microsoft.com/library/en-us/com/hh/com/security_47fx.asp">http://msdn.microsoft.com/library/en-us/com/hh/com/security_47fx.asp</a>                                                         | MSDN Library: COM-Sicherheit                                                          |
| [MSD01b] | <a href="http://msdn.microsoft.com/library/en-us/com/hh/com/monikers_1xpv.asp">http://msdn.microsoft.com/library/en-us/com/hh/com/monikers_1xpv.asp</a>                                                         | MSDN Library: COM-Moniker                                                             |
| [MSD01c] | <a href="http://msdn.microsoft.com/library/en-us/com/hh/com/comp_cat_7hri.asp">http://msdn.microsoft.com/library/en-us/com/hh/com/comp_cat_7hri.asp</a>                                                         | MSDN Library: COM - Komponentenkategorien                                             |
| [MSD01d] | <a href="http://msdn.microsoft.com/library/en-us/com/hh/com/comportal_3qn9.asp">http://msdn.microsoft.com/library/en-us/com/hh/com/comportal_3qn9.asp</a>                                                       | MSDN Library: COM                                                                     |
| [MSD01e] | <a href="http://msdn.microsoft.com/library/en-us/com/hh/com/reg_9ann.asp">http://msdn.microsoft.com/library/en-us/com/hh/com/reg_9ann.asp</a>                                                                   | MSDN Library: Registry-Einträge für COM                                               |
| [MSD01f] | <a href="http://msdn.microsoft.com/library/en-us/com/hh/com/registry_32er.asp">http://msdn.microsoft.com/library/en-us/com/hh/com/registry_32er.asp</a>                                                         | MSDN Library: COM und die Registry                                                    |
| [MSD01g] | <a href="http://msdn.microsoft.com/library/en-us/cdo/html/_olemsg_item_property_infostores_collection.asp">http://msdn.microsoft.com/library/en-us/cdo/html/_olemsg_item_property_infostores_collection.asp</a> | CDO-Dokumentation in der MSDN Library: Beispiel für Item() als Attribut               |
| [MSD01h] | <a href="http://msdn.microsoft.com/library/en-us/script56/html/jsproitem.asp">http://msdn.microsoft.com/library/en-us/script56/html/jsproitem.asp</a>                                                           | Scripting Runtime-Dokumentation in der MSDN Library: Beispiel für Item() als Attribut |
| [MSD01i] | <a href="http://msdn.microsoft.com/library/en-us/vbenlr98/html/vamthItem.asp">http://msdn.microsoft.com/library/en-us/vbenlr98/html/vamthItem.asp</a>                                                           | VB6.0-Dokumentation in der MSDN Library: Beispiel für Item() als Methode              |
| [MSD01j] | <a href="http://msdn.microsoft.com/library/en-us/vbcmn98/html/vbmthItemMethodActiveXControls.asp">http://msdn.microsoft.com/library/en-us/vbcmn98/html/vbmthItemMethodActiveXControls.asp</a>                   | VB6.0-Dokumentation in der MSDN Library: Beispiel für Item() als Methode              |



# *Literaturverzeichnis*

|          |                                                                                                                                                               |                                                                                                |
|----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------|
| [MSD01k] | <a href="http://msdn.microsoft.com/library/en-us/vbext98/html/vbmthItemAddIn.asp">http://msdn.microsoft.com/library/en-us/vbext98/html/vbmthItemAddIn.asp</a> | VB6.0-Dokumentation in der MSDN Library: Beispiel für Item() als Attribut                      |
| [MSL00]  | <a href="http://msdn.microsoft.com/library">http://msdn.microsoft.com/library</a>                                                                             | Kostenlose Online-Fassung der MSDN Library                                                     |
| [MSR01]  | <a href="http://research.microsoft.com">http://research.microsoft.com</a>                                                                                     | Homepage von Microsoft Research (MSR)                                                          |
| [MSS00]  | <a href="http://msdn.microsoft.com/scripting">http://msdn.microsoft.com/scripting</a>                                                                         | Microsoft Scripting Website                                                                    |
| [MSU00]  | <a href="http://support.microsoft.com/support/activescript">http://support.microsoft.com/support/activescript</a>                                             | Microsoft Support-Site für Active Scripting                                                    |
| [NCL00]  | <a href="http://www.ncompasslabs.com">http://www.ncompasslabs.com</a>                                                                                         | Hersteller des VBS-Plugins für den Netscape Navigator                                          |
| [NCO00]  | <a href="http://ncompasslabs.com">http://ncompasslabs.com</a>                                                                                                 | NCompassLab stellt ein Plug-in her, das Netscape COM-fähig macht.                              |
| [NET00]  | <a href="http://www.netal.com">http://www.netal.com</a>                                                                                                       | Hersteller des System Script Hosts                                                             |
| [NIQ00]  | <a href="http://www.netiq.com/Manageability_Partnership">http://www.netiq.com/Manageability_Partnership</a>                                                   | Informationen zur Partnerschaft zwischen Microsoft und NetIQ                                   |
| [NIQ01]  | <a href="http://www.netiq.com">http://www.netiq.com</a>                                                                                                       | Homepage der Firma NetIQ, dem ursprünglichen Hersteller des Microsoft Operations Manager (MOM) |
| [NTF00]  | <a href="http://www.vbadmincode.btinternet.co.uk">http://www.vbadmincode.btinternet.co.uk</a>                                                                 | Der Däne Morten Brun sammelt Automatisierungslösungen.                                         |
| [OAS00]  | <a href="http://www.activescripting.com">http://www.activescripting.com</a>                                                                                   | OpenASP (ASP für Win32-Apache und Win32-WebServer von Netscape)                                |
| [ODB00a] | <a href="http://www.microsoft.com/data">http://www.microsoft.com/data</a>                                                                                     | Microsoft Site zum Thema Universal Data Access                                                 |
| [ODB00b] | <a href="http://www.oledb.com">http://www.oledb.com</a>                                                                                                       | OLE DB-Site der Firma Merant                                                                   |

# *Literaturverzeichnis*

|          |                                                                                                                                       |                                                                                                                         |
|----------|---------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------|
| [ODI00]  | <a href="http://www.odi.com">http://www.odi.com</a>                                                                                   | Site der Firma Object Design (Hersteller der Datenbank Object Store)                                                    |
| [PGP01]  | <a href="http://community.wow.net/grt/nsdpgp.html">http://community.wow.net/grt/nsdpgp.html</a>                                       | PGP (NSDPGP)                                                                                                            |
| [PYT00a] | <a href="http://www.python.org/windows">http://www.python.org/windows</a>                                                             | PythonScript                                                                                                            |
| [PYT00b] | <a href="http://starship.python.net/crew/mhammond">http://starship.python.net/crew/mhammond</a>                                       | Site von Mark Hammond, dem Schöpfer der ActiveX Scripting Version von Python-Script                                     |
| [ROG]    | <a href="http://www.mypeecee.org/rogsoft">http://www.mypeecee.org/rogsoft</a>                                                         | Homepage zu Notepad+ (Firma Rogsoft)                                                                                    |
| [ROL00]  | <a href="http://www.rollthunder.com">http://www.rollthunder.com</a>                                                                   | COM-Homepage des Buchautors David S. Platt (kostenloser Newsletter)                                                     |
| [RUB01a] | <a href="http://activeruby.freervers.com/main.html">http://activeruby.freervers.com/main.html</a>                                     | Homepage zu ActiveScriptRuby, der Active Scripting-fähigen Implementierung der objektorientierten Skriptsprache »Ruby«. |
| [RUB01b] | <a href="http://www.ruby-lang.org/en/">http://www.ruby-lang.org/en/</a>                                                               | Offizielle Homepage der objektorientierten Skriptsprache »Ruby«                                                         |
| [SEC00]  | <a href="http://www.15seconds.com">http://www.15seconds.com</a>                                                                       | Developer-Community, insbesondere zu ASP; viel Beispiel-Code und zahlreiche Komponenten und Tools                       |
| [SOA01a] | <a href="http://msdn.microsoft.com/xml/general/soapspec.asp">http://msdn.microsoft.com/xml/general/soapspec.asp</a>                   | SOAP-Spezifikation auf dem Microsoft-Webserver                                                                          |
| [SOA01b] | <a href="http://www.w3.org/TR/SOAP">http://www.w3.org/TR/SOAP</a>                                                                     | SOAP-Spezifikation auf dem W3C-Webserver                                                                                |
| [TEB00]  | <a href="http://snad.ncsl.nist.gov/snad-staff/tebbutt/x5eg/x5eg.html">http://snad.ncsl.nist.gov/snad-staff/tebbutt/x5eg/x5eg.html</a> | Tebbutt, J. et al.: Guidelines for the Evaluation of X.500 Directory Products (1995)                                    |
| [TOP00]  | <a href="http://www.toptensoftware.com/quickprompts">http://www.toptensoftware.com/quickprompts</a>                                   | TopTenSoftware ist der Hersteller der Komponente QuickPrompts.                                                          |

### Literaturverzeichnis

|          |                                                                                                                                                   |                                                                                                                  |
|----------|---------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------|
| [TXP00]  | <a href="http://www.ozemail.com.au/~launch/textpipe.html">http://www.ozemail.com.au/~launch/textpipe.html</a>                                     | Das Text-Transformationswerkzeug Text-Pipe unterstützt ActiveX Scripting auf Basis des Microsoft Script Control. |
| [VBA01]  | <a href="http://msdn.microsoft.com/vba/companies/company.asp">http://msdn.microsoft.com/vba/companies/company.asp</a>                             | Liste der VBA-Partner und VBA-fähigen Produkte                                                                   |
| [VBN00]  | <a href="http://msdn.microsoft.com/vstudio/nextgen/technology/language.asp">http://msdn.microsoft.com/vstudio/nextgen/technology/language.asp</a> | Vorabinformationen zu VB 7.0                                                                                     |
| [VSA01a] | <a href="http://msdn.microsoft.com/vstudio/vsip/vsa">http://msdn.microsoft.com/vstudio/vsip/vsa</a>                                               | VSA-Site von Microsoft                                                                                           |
| [VSA01b] | <a href="http://vsip.summsoft.com/vsa/">http://vsip.summsoft.com/vsa/</a>                                                                         | VSA-Site von Summit Software                                                                                     |
| [WAR00]  | <a href="http://home.att.net/~wshvbs/">http://home.att.net/~wshvbs/</a>                                                                           | Scripting-Site von J. Warrington                                                                                 |
| [WIN01]  | <a href="http://www.wininfo.de">http://www.wininfo.de</a>                                                                                         | Website von Dr. Tobias Weltner                                                                                   |
| [WMA01]  | <a href="http://www.win32scripting.com/">http://www.win32scripting.com/</a>                                                                       | Scripting-Website des Windows 2000 Magazine mit einer kostenpflichtigen Skript-Bibliothek                        |
| [WSS00]  | <a href="http://www.windows-scripting.de">http://www.windows-scripting.de</a>                                                                     | Deutsche Windows Scripting-Site                                                                                  |
| [XLN00]  | <a href="http://www.xlnow.ch">http://www.xlnow.ch</a>                                                                                             | Hersteller des OnScript Hosts und des OnScript Editors                                                           |
| [XML01]  | <a href="http://msdn.microsoft.com/xml/default.asp">http://msdn.microsoft.com/xml/default.asp</a>                                                 | Microsoft XML Developer Center                                                                                   |
| [XRP00]  | <a href="http://www.xml-rpc.com">http://www.xml-rpc.com</a>                                                                                       | Site zum Thema XML RPC                                                                                           |

## **E.3 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.

## **E.4 Newsgroups**

### **Scripting**

#### **Diskussions- gruppen im Internet**

news://microsoft.public.de.german.scripting.wsh  
news://microsoft.public.active.directory.interfaces  
news://microsoft.public.inetexplorer.scripting  
news://microsoft.public.scripting.debugger  
news://microsoft.public.scripting.hosting  
news://microsoft.public.scripting.jscript  
news://microsoft.public.scripting.remote  
news://microsoft.public.scripting.scriptlets  
news://microsoft.public.scripting.vbscript  
news://microsoft.public.scripting.wsh  
news://microsoft.public.scripting.virus.discussion  
news://microsoft.public.windows.inetexplorer.ie5.  
programming.dhtml.scripting

### **COM-Programmierung allgemein**

news://microsoft.public.vb.ole  
news://microsoft.public.vb.ole.servers  
news://microsoft.public.vb.ole.automation  
news://microsoft.public.windna.components  
news://microsoft.public.platformsdk.com\_ole  
news://microsoft.public.platformsdk.complus\_mts  
news://microsoft.public.platformsdk.component\_svcs

### **.NET (Auswahl)**

news://microsoft.public.dotnet.general  
news://microsoft.public.dotnet.framework  
news://microsoft.public.dotnet.languages.vb  
news://microsoft.public.de.german.entwickler.dotnet.framework  
news://microsoft.public.de.german.entwickler.dotnet.vb

### **Spezielle Komponenten**

news://microsoft.public.adsi.general  
news://microsoft.public.wbem  
news://microsoft.public.vb.database.ado  
news://microsoft.public.data.ado

## **E.5    Andere Quellenangaben**

### **Verweise auf Knowledge Base-Artikel**

[Qxxxxx]

**Knowledge Base**

Diese Quellenangabe nimmt Bezug auf einen Beitrag aus der Microsoft Knowledge Base mit der Nummer xxxx. Sie finden die Knowledge Base im Internet unter <http://search.support.microsoft.com/kb>.

### **Verweise auf die Buch-CD**

[CD:/pfad/] oder  
[CD:/pfad/datei.extension]

**Buch-CD**  
**[Datei.extension]**  
**oder**

Auch Verweise auf Dateien auf der Buch-CD sind im Stil eines COM-Monitors angegeben. Unter vielen Listings in Kapitel 4 bis 26 sehen Sie allerdings nur die Kurzform ohne Pfadangaben, z.B. [adsint\_neuerBenutzer.wsf]. In diesem Fall finden Sie den Programmcode anhand des Kapitelnamens, da das Verzeichnis /CODE/KOMPONENTEN auf der CD genauso strukturiert ist wie das Buch.



# Stichwortverzeichnis

## !

.eml 621  
.NET 25, 44, 100, 143, 159, 781–782  
    Anwendung 156  
    Framework 88  
    Garbage Collection 158, 183  
    Laufzeitumgebung 156  
    Metadaten 156  
    Newsgroup 821  
    Sicherheit 159  
    vs. COM 156

## A

Abmelden 531  
Access 751  
Access Control Entry 391, 453,  
    471, 480  
Access Control List 35, 108, 408,  
    453, 470–471, 532  
AccessMask 471  
Account Control 35  
AcctCrt 35, 453  
AceFlags 471  
AceType 471  
ACL-Komponente 35  
Active Directory 25, 35, 122, 379,  
    383, 393, 395, 415, 435, 485–486  
    Exchange 2000 626  
    Klasse 438  
    MOM Hilfsobjekte 735, 742  
    Namespaces Container 416  
    Objektmodell 436  
    Rechte 470  
    Replikation 485  
Active Directory Service 32  
Active Directory Service  
    Interface 31–32, 35–36, 38, 128,  
    308, 375, 383, 386, 400, 464  
    Active Directory-  
        Administration 435  
    ADO-Query 412, 441, 452, 456  
    ADSI-Browser 380, 393  
    ADSI-Edit 395  
    Architektur 382  
    Bindung 397

    Client 383  
    Container 403  
    Exchange 2000 626  
    Exchange Server  
        Administration 444  
    IIS-Provider 458  
    Impersonifizierung 407  
    Installation 383  
    Instanzenverwaltung 405  
    Integration mit  
        CDOEXM 671  
    Internet Information  
        Server 458  
    Klasse ADSI 722  
    LDAP-Provider 403, 435, 444  
    Meta-Objektmodell 130, 385,  
        388  
    Namespace Extension  
        383, 468, 530  
    Pfad 397  
    Programmierung 397  
    Provider 383, 387  
    Query 393  
    Resource Kit 464  
    SchemaBrowser 397  
    SDK 385, 464  
    Sicherheit 407  
    Standardschnittstelle 388  
    Werkzeug 393  
    WinNT-Provider 418, 436  
    WMI-Integration 529  
    Zusatzkomponente 464  
Active Group 152  
Active Messaging 582  
Active Script 109  
Active Scripting 26–28, 67, 109  
Active Scripting Host 173, 787  
Active Server Page 114  
    DNA 29  
    Intrinsic Object 164  
Active Template Library 41, 156  
ActiveScriptEventConsumer  
    501, 527  
ActiveScriptRuby 112  
ActiveX 26–28, 152, 186

    Container 28  
    Control Host 28  
    Definition 27  
    DLL 185, 187  
    Document 186  
    Document Host 28  
    Dokument 28, 47  
    EXE 185  
    Steuerelement 47  
    Steuerelement erzeugen 185  
ActiveX Data Objects 34, 38, 128,  
    165, 305–306, 309, 318, 374,  
    412, 798  
    Exchange 2000 633  
    Extension 34  
    Meta-Objektmodell 131  
    Multi Dimensional 34  
    via CDO 3.0 657  
ActiveX Scripting siehe Active  
    Scripting  
ActiveXObject() 90  
ActiveX-Steuerelement  
    28, 141, 725  
    Internet Transfer 691  
    WinSock 691  
Add 180  
Add-In 186  
AddRef() 79, 158  
Address Book Provider 580  
AddressEntry 585  
AddressList 585  
admin.exe 449  
Administrator 25–26  
ADO siehe ActiveX Data  
    Objects 34  
ADOMD 312  
ADOX 312  
Adressbuch 580, 585, 612  
ADsError 466  
ADsFactory 467  
ADSI Explorer 394  
ADSI siehe Active Directory  
    Service Interface  
ADsRas 468

## Stichwortverzeichnis

- ADsSecurity 32–33, 35, 362, 409, 453, 455, 470, 532
  - Klasse 475
- ADsSID 476
- ADsVersion 465
- Advanced Data Tablegram 331
- Aggregation 196
  - Definition 770
- Aktivierungsfehler 94
- Aktivierungsort 89
- Anhang siehe Attachment
- Anwendung 25, 30
  - herunterfahren 556
- Anwendungskennung siehe AppID
- Anwendungsprotokoll 741
- API siehe Application Programming Interface
- AppID 95, 137, 144, 162
  - Definition 95
- Application Programming Interface 21, 25, 30
  - Registryzugriff 362
- Application Server 782
- Arbeitsgruppe 376
- Arbeitsplatz 271
- Arbeitsverzeichnis 229
- AS/400 309
- ASCII 260
- ASPInet 33, 691
- Assembly 44, 100, 152
  - Private 100
  - Public 100
  - Shared 100
- Assoc-NT-Account 455
- Assoziation 196
  - Definition 770
  - WMI 491, 495
- ATL siehe Active Template Library
- Attachment
  - CDO 1.21 615
  - CDOSYS 687
  - Exchange 5.5 585
- Attribut 124
  - Active Directory 435
  - Definition 763
  - erstellen 187, 191
  - Notation 792
  - Verzeichnisdienst 376–377, 391
  - vs. Property 197
- Attribut-Wert-Paar 381
- Attributzugriff 170
- Aufgabe
  - Exchange 2000 629
  - Exchange 5.5 605
- Aufruf, Methode 517
- Ausgabe 787
- Automation 784
- Automation Client 76
- Automation Server 76
- Automation siehe COM Automation
- Automatisierung 22, 38
  - Active Directory 35, 435, 470
  - Benutzeroberfläche 31
  - Benutzerverwaltung 31, 424, 720
  - COM+ 545
  - Computerverwaltung 511
  - Dateisystem 32, 470, 516
  - Datenbank 34, 305
  - Definition 784
  - Dienste 32
  - Domänenverwaltung 421
  - Drucker 226, 434
  - DTS 577
  - Ereignisprotokoll 31, 233, 369, 522
  - Exchange Server 444, 470
  - Exchange Server 2000 36, 619
  - Exchange Server 5.5 35, 580
  - Explorer 268
  - Freigabe 532
  - FTP 699, 702
  - Groupware 33
  - IIS 458
  - Internet Information Server 35
  - Komponenten 32
  - Komponentenkonfiguration 32
  - Lotus Notes 37
  - Messaging 33
  - Microsoft Office 37, 751
  - MOM 735
  - MTS 556
  - Netzwerk 225
  - Netzwerkcommunication 33
  - NT-Dienst 431, 518
  - Oracle 34
  - PGP 34
  - Rechteverwaltung 470
  - Registry 32, 230, 361, 470
  - SAP R/3 37
  - Scheduler 32
  - SMTP 681
  - SQL Server 34, 559
  - Systemmanagement 31
  - Taskscheduler 349
  - TCP/IP 33, 691
  - Textdatei 34
  - Verschlüsselung 34
  - Verzeichnisdienst 35
  - XML 288
  - Zeitplandienst 32
- B**
- BackOffice 22, 38, 111
- Backup
  - erstellen 574
  - starten 575
- Bag 124, 774
- Banyan Vines 376
- Benutzer 225, 375, 390, 421, 424, 485
  - anlegen 426, 442
  - anlegen (ADSI) 721
  - anlegen (CDO 3.0) 673
  - auflisten 425, 440
  - deaktivieren 428, 721
  - im Active Directory 438, 440
  - löschen 428, 721
  - Passwort ändern 721
  - umbenennen 427
  - Verwaltung 31
- Benutzergruppe siehe Gruppe
- Benutzerkontext 359
- Benutzeroberfläche 31
- Benutzerrechte 428
- Benutzerverwaltung 31, 187
- Betriebssystem 25, 30, 739
  - Herunterfahren 739
  - Neustarten 739
  - Sperrung 739
- Beziehung
  - 0/n-zu-0/m 771
  - 1-zu-0/1 771
  - 1-zu-0/n 771
  - 1-zu-1 771, 793
  - 1-zu-m 771
  - 1-zu-n 771–772, 793
  - Is-part-of 770
  - Kann-Beziehung 771, 773, 793
  - Muss-Beziehung 771, 773, 793



## Stichwortverzeichnis

- n-zu-m 124
- n-zu-n 772
- zwischen Klassen 770
- zwischen Objekten 770
- Big Endian 413
- Bildschirmmaske 31
- Binärstandard 29, 39, 63
- Binden
  - Definition 777
  - dynamisch 77, 80, 777
  - früh 77, 163, 167
  - serverlos 398
  - spät 77, 80, 163, 167
  - statisch 77–78
  - über eine GUID 398, 439
- Bindung
  - Active Directory 438
  - ADSI 397
  - ADsSecurity 477
  - COM+ Administration
    - Objects 550
  - Exchange Server 448
  - IIS 460
  - in Meta-Objektmodellen 127
  - Managed Object 502
- BIOS 484
- BizTalk Server 111
- Blatt siehe Leaf
- BodyPart 658
- Boot-Konfiguration 484
- Borland 782
- Botschaftenkonzept siehe Nachricht
- Browserfenster 269
- Built-In Object siehe Intrinsic Object
- Bullet-and-Stick 67
- Byte 197
- C**
- C 25
- C++ 25–26, 39, 63–64, 70, 92, 162, 172, 309, 771, 792
- Callout 790
- CAS siehe Code Access Security
- CATID 101
- catsrv 545
- CDO for Exchange
  - Management 36, 626, 666
- CDO siehe Collaboration Data Objects
- CDO Workflow Objects 37, 627
- CDOEXM siehe CDO for Exchange Management
- CDONTS 33, 582, 681
- CDOSYS 33, 681–682, 684
- CD-ROM zu diesem Buch 797
- CheckCreate() 722
- CHKDSK 485
- CIM siehe Common Information Model
  - 483
  - Klasse 494
- CIM Studio 537
- Class Factory 53, 91
- Class Store 122
- Class\_Initialize 161, 206
- Class\_Terminate 161, 206
- ClassFactory-Objekt 91
- ClassObject siehe ClassFactory
- Client 765
  - Definition 765, 781
- CliReg 134
- CLSID 89, 95, 135, 139
  - Definition 58
- Clustering 486
- CoClass 58, 66, 189
- CoCreateInstance() 89
- Code Access Security 159
- CoGetInstanceFromFile() 93
- Collaboration Data Objects 33, 35, 65, 172
  - Anmeldung 593
  - CDONTS 682
  - Exchange 2000 625
  - Session 584, 593
  - Version 1.21 35, 579, 625, 682–683
  - Version 2.0 625, 684
  - Version 3.0 36, 625, 656
  - Versionsüberblick 582
- Collection 68, 124, 178–179, 248, 388, 405, 774
- CDO 589
- COM+ Administration
  - Objects 548
- DHTML 286
- erstellen 203
- Iteration 179
- verändern 180
- WSH 221
- COM 25, 27, 84, 280, 772, 781–782
  - Add-In 186
  - Anwendung 43, 95, 137
  - API siehe COM-Bibliothek
- Automation 28, 66, 76, 80, 161, 383
- Bibliothek 27
- Binärstandard 29
- Client 44, 65, 189
- Definition 26
- Dienst 28, 47, 78
- DLL 52, 95, 118, 133, 162, 383
- EXE 95, 133, 162, 165, 189
- Identität 147
- Implementierung 27
- Integration in Windows 29
- Internet Service 86
- Kategorie 100
- Klasse 42, 57, 135, 140, 164, 169, 185
- Komponente 22, 38, 109, 184–185, 485
- Komponentendatei 60
- Komponenteninstallation 758
- Komponentenüberblick 29–30
- Laufzeitumgebung 122
- Moniker 397
- Namensdienst 92
- Newsgroup 821
- Objekt 169
- Schnittstelle 61, 140
- Scripting 109
- Server 44
- Sicherheit 104, 137, 147–148, 499
- Spezifikation 27
- Standardschnittstelle 78
- Statischer Aufruf 76
- Typbibliothek 32
- UNIX 152
- Werkzeug 131
- COM+ 28, 32, 119, 165
  - Administration Objects 545
  - Anwendung 149, 553
  - Automatisierung 545
  - Catalog 120
  - Definition 28
  - Event Service 121
  - Katalog 545
  - versus .NET 156
- comdlg32 siehe Common Dialog Control
- COM-DLL 149
- COM-Explorer 139–140, 754

## Stichwortverzeichnis

- COM-Kategorie, auflisten 100, 522
  - COM-Komponente
    - Erforschung 543
    - erzeugen 184–185
  - Comma Separated Value 722
  - Command Line Event
    - Consumer 501
  - Command, ADO 319, 333
  - Commerce Server 111
  - Common Dialog Control 31, 725
  - Common Information Model 483
  - Common Language Runtime 156
  - Common Management
    - Information Protocol 483
  - Common Name 381, 447
  - Common Object Request Broker
    - Architecture 27, 47, 781–782
  - Compensating Resource
    - Manager 121
  - Compiler 156, 777
  - Component Load Balancing 121
  - Componentware 781
    - Definition 778
  - Compound Document 27, 93
  - Compound File 102
  - Computer 225, 375, 390, 420, 423, 446
    - Computerverwaltung 423
    - im Active Directory 438
  - Computer-Klasse, WinNT-Provider 387
  - COMSource 152
  - comTLBrowser 143
  - COM-Viewer 55, 59, 74, 96, 102, 132, 135, 754
  - Configuration Utility siehe DCOMCNFG
  - Connection 319
    - ADO 319, 337
    - Exchange 2000 639
  - Connection Point 69, 121
    - siehe Verbindungspunkt
  - Connection Pooling 118
  - Connection String 315
    - Ereignisprotokoll 373
  - Consumer 306
  - Container 248, 376, 386, 403
    - im Active Directory 438
  - Containment siehe Nutzungsbeziehung
  - Containment-Hierarchie 535
    - ADSI 378
    - Definition 770
    - WMI 491
  - Content Class 628
  - CORBA siehe Common Object Request Broker Architecture
  - Corel 782
  - Count 179
  - Country 381
  - CreateObject 163–164, 166, 189
  - CreateObject()
    - ADSI 416
    - CoCreateInstance() 90
    - Kapselung für
      - GetObject() 467
    - Syntax 166
    - versus GetObject() 94
  - CSVadd() 722
  - CSV-Datei 34, 722
  - CurrentDirectory 229
  - Cursor-Typ 322
  - Custom Marshaller 51
  - Cutter Consortium 781
- D**
- Dana Scully 788
  - Das 763
  - Data Access Components 310
    - Component Checker 311
  - Data Access Objects 309
  - Data Link 314
  - Data Provider 306
  - Data Source Name 34, 313
  - Data Transformation Service
    - 34, 114
    - Automatisierung 577
    - Paket 577
  - Datei 251, 266, 271, 484
    - CSV 34
    - Eigenschaft 256, 266, 273
    - INI 34
    - Rechte 470, 485
    - Verknüpfung 266
    - Version 258
  - Dateigruppe anlegen 568
  - Dateisystem 32, 168, 251, 254, 390
    - Sicherheit 252
  - Dateisystemoperation 258–259
  - Datenbank 34
    - anlegen 567
    - löschen 571
    - relational 306, 580
    - semi-strukturiert 306, 412, 580, 619
    - strukturiert 306
  - Datenbankdatei 306
    - anlegen 568
  - Datenbankmanagementsystem 305–306
  - Datenlink siehe Data Link
  - Datenquelle 34
  - Datensatz 324
  - Datenstrom siehe Stream
  - Datentyp 175, 786
    - ADSI 403
    - elementar 163
    - WMI 496
  - Datenverbindungen 312
  - DB2/400 309
  - DCO siehe Domino
    - Collaboration Objects
  - DCOM 26, 84, 165, 280
    - Definition 27
    - im Internet 86
    - Installation 85
    - Konfiguration 86, 485
    - Nutzung in VB 161
    - Protokoll 84, 146
    - Rechte 166
  - DCOM Connector, SAP 37
  - DCOMCNFG 27, 87, 96, 104, 108, 132, 137, 144, 148, 162, 166
    - Vorstellung 144
  - DCOM-Konfiguration 149
  - Debug, VB6.0 164
  - Decision Support Objects 34
  - Deep Traversal 636
  - Default Property 176, 202
  - DefaultNamingContext 626
  - Deklaration 163, 786
  - Delphi 112
  - Demarshaling 51
  - Dependency Walker 57, 140
  - Design Time Control 28
  - Desktop 485
  - DFS siehe Distributed File System
  - DFÜ 33
  - DHTML siehe Dynamic HTML
  - Dialogbox 229
  - Dialogfenster 726
  - Dictionary 203, 248

## Stichwortverzeichnis

- Dienst 421, 484, 765, 778
    - ADSI 375, 431
    - auflisten 521
    - IADsService 390
    - IADsServiceOperations 390
    - starten/stoppen 432, 518
    - Überwachung 524
    - Windows 32
  - Dim 191
  - Directory Access Protocol 379
  - Directory Application
    - Programming Interface 444
  - Directory Management
    - Domain 445
  - Directory Service siehe Verzeichnisdienst
  - Directory System Agent 375
  - Directory Tree siehe Verzeichnisbaum
  - Directory User Agent 375
  - Disk Quota 33, 486
  - DispID 81
    - Definition 80
  - Distinguished Name 381, 397
    - Exchange Server 448
  - Distributed Authoring and Versioning 628
  - Distributed COM, versus .NET 157
  - Distributed COM siehe DCOM
  - Distributed Computing
    - Environment 45, 84
  - Distributed File System 33, 485
  - Distributed InterNet
    - Applications Architecture 29, 116
  - Distributed Management
    - Object 34, 165, 559
  - Distributed Management Objects, Meta-Objektmodell 131
  - Distributed Transaction
    - Coordinator 804
  - DLL, Versionsinformation 252
  - DllCanUnloadNow 53
  - DLL-Datei 52
  - DllGetClassObject 53
  - DLL-Hölle 97, 152
  - DLLHost 92
    - COM+ 120
  - DllRegisterServer 53
  - DllUnregisterServer 53
  - DMO siehe Distributed Management Objects 34
  - DN siehe Distinguished Name
  - DNA siehe Distributed InterNet Application Architecture
  - DNS siehe Domain Name Service
  - Document Object Model 277, 279–280, 288
  - Document Type Definition 288, 297
  - Dokument 28, 279
    - Exchange 2000 629
    - Exchange 5.5 605
    - HTML 281
    - XML 293
  - DOM siehe Document Object Model
  - Domäne 376, 390, 420–421, 425
    - im Active Directory 436
  - Domänen-Controller 435
  - Domain Name Service 35, 482
  - Domino Collaboration Objects 33, 37
  - DOS 705
  - DOS-Anwendung 239
  - DOS-Batch 25
  - DOTNET siehe .NET
  - DropDirectory
    - CDO 2.0 684
    - CDO 3.0 658
    - CDOSYS 687
    - SMTP-E-Mail 681
  - Druckauftrag 391, 485
  - Drucker 226, 375, 390, 421, 434, 485
    - im Active Directory 438
  - Druckerauswahl-Dialog 728
  - Druckerverbindung 225
  - Druckerwarteschlange 485
  - DSN siehe Data Source Name
  - DSO siehe Decision Support Objects
  - DTD siehe Document Type Definition
  - DTS siehe Data Transformation Service
  - DTSPackage Object Library 34
  - Duke 37
  - Dynamic Data Exchange 77
  - Dynamic Host Configuration Protocol 802
  - Dynamic HTML
    - DNA 29
    - Object Model 280
  - Dynamic Method Invocation 76
  - Dynamic Object Interface 389
  - DynaZIP 33, 814
- ## E
- Early Binding siehe Binden, früh
  - ECMA 112
  - ECMAScript 112
  - Eingabehilfe 81
  - EJB siehe Enterprise Java Beans
  - E-Mail 33
    - EmailEvent 524
    - senden (CDO 1.21) 613
    - senden (CDO 3.0) 664
    - senden (CDONTS) 683
    - senden (CDOSYS) 685
    - senden (JMAIL) 689
  - E-Mail-Empfänger siehe Empfänger
  - E-Mail-Verteiler siehe Verteiler
  - EML-Datei 682, 684
    - lesen (CDOSYS) 687
  - Empfänger 445–446, 452, 626
    - Exchange 5.5 585
  - EntireX 152
  - Entity-Relationship-Diagramm 42
  - Entwickler 25–26, 30
  - Entwicklungsumgebung 141
  - Erde 789
  - Ereignis 67–68, 124, 161, 764
    - Aggregation 526
    - Definition 764
    - erstellen 187, 191, 205
    - Internet Explorer 278
    - von COM-Objekt empfangen 173
    - WMI 500
  - Ereignisanzeige 369
  - Ereignisbehandlungsroutine 69
  - Ereigniskonsument 173, 205, 501
  - Ereignisprotokoll 31, 309, 369, 484, 488, 501, 522, 738, 741
    - Überwachung 524–525
  - Error siehe Fehler 319
  - ESConfig-Komponente 35
  - Event Agent 114
  - Event Consumer 540
    - permanent 500, 527
    - temporär 500, 525
  - Event Correlation 485, 526
  - Event Filter 540

## Stichwortverzeichnis

- Event Publisher 68, 161, 206
- Event Registration Tool 540
- Event Scripting Agent 111
- Event siehe Ereignis 161
- Event Sink 69
- Event Subscriber 68, 161, 206
- Event Viewer 541
- Eventlog siehe Ereignisprotokoll
- EventLogOLEDB 373
- EventViewerConsumer 500
- Excel 751
- Exchange Installable File
  - System 620
- Exchange Server 25, 35, 111, 375, 379, 415, 444
  - 2000 36, 619
  - Automatisierung (Version 2000) 619
  - Automatisierung (Version 5.5) 579
  - Empfänger 452, 626
  - Klasse 446
  - Konfiguration 457
  - MOM Hilfsobjekte 735, 745
  - Rechte 470
  - Resource Kit 35
  - System-Manager 622, 745
  - Version 2000 36, 307, 393, 435, 488, 619
  - Version 5.0 444
  - Version 5.5 35, 444
- EXE-Datei 52
- Exemplar siehe Instanz
- ExIFS siehe Exchange Installable File System
- ExistsObject() 399, 722
- Explorer siehe Internet Explorer
- Explorer-Fenster 268
- Extended WQL 502
- Extensible Markup Language 34, 279, 331
  - Programmierung 288
- F**
- Farbauswahl-Dialog 728
- Fehler
  - ADO 345
  - ADODB 319
- Feld
  - ADODB 319
  - CDO 1.21 583, 585, 590
  - Datenbank 323
  - Exchange 2000 627
  - MAPI 590
- Fenster 265
- Fernaktivierung 165
- Fernausführung 157
- Fernsteuerung 239
- File System Objects 32, 247, 251, 259, 266
  - Dateisystem 251
- File Transfer Protocol 33, 309, 695
- Fileservice 390, 431
- Filter 327
- Firewall 88, 157
- Flat() 722
- Flexible Single Master Operations 436
- Folder, CDO 1.21 585
- For 179
- For...Each 179
- For...Next 179
- Forest 436
- Fox Mulder 788
- Freeware 31, 33
- Freigabe 390, 421, 432, 485
  - anlegen 532
  - Erstellung eines Security Descriptor 532
- Friendly Class Name 58–60, 135
- FrontPage 751
- FSO siehe File System Objects
- FTP Service 703
- FTP siehe File Transfer Protocol 695, 700
- FTPX 33, 691, 702
- Fully Qualified Distinguished Name siehe Distinguished Name
- Function 191
- Funktionsoffset 63
- G**
- GAC siehe Global Assembly Cache
- Garbage Collection 85, 158
  - VB 183
- Get\_From\_Array() 722
- GetCol() 722
- GetIDsofNames 174
- GetIDsOfNames() 80
- GetInterface(), CDO 3.0 657
- GetObject() 163, 168
  - ADsFactory 467
  - ADSI 416
  - Aktivierung aus Datei 93
  - Syntax 168
  - versus CreateObject() 94
- GetTypeInfo() 81
- GetTypeInfoCount() 81
- Global Unique Identifier 45, 97
- Globale Adressliste 612
- GlobalMultiUse 189
- GlobalSingleUse 189
- Glue Code 779
- Grafikkarte 484
- Graph 37
- Graphical User Interface 31, 483
- Größe ermitteln 743
- Group Policy 482
- Groupware 33
- Gruppe 390, 421, 429
  - anlegen 430
  - im Active Directory 438
- GUID siehe Global Unique Identifier
- GUIDGEN 46
- Gullivers Reisen 381
- H**
- Hardware 484
- HaskellScript 112
- Hersteller 30, 159
- Herunterfahren 531
- HKEY\_CLASSES\_ROOT 48
- Hotfix 484
- HS 788
- HTTP siehe Hypertext Transfer Protocol
- HTTPS 695
- Hypertext Markup Language 279–280
- Hypertext Transfer Protocol 33, 695
- I**
- IADs 387, 392, 420
- IADsContainer 387, 392, 420
- IADsTools 481
- IBM 112
- IClientSecurity 108
- Identifikation
  - Exchange 2000 627
  - Registry 365
- Identität 105

## Stichwortverzeichnis

- IDispatch 51, 66, 77, 83, 89, 156, 161–162, 168, 172, 186, 290
    - CDO 3.0 657
    - Definition 80
  - IDispatchEx 84
  - IDL siehe Interface Definition
    - Language 765
  - IID 89
    - Definition 63
  - IIS Resource Kit 34–35
  - IIS siehe Internet Information
    - Server 35
  - IISAdmin-Objekte 458
  - Impersonation siehe
    - Impersonifizierung
  - Impersonifizierung 107
    - ADSI 391, 407
    - WMI 499
  - Implementierung einer
    - Klasse 766
  - Implementierungsvererbung 211
  - Implements 211–212
  - Index 179
  - Information Hiding siehe
    - Kapselung
  - Information Store 36, 580, 584
    - CDO 1.21 584
    - Zugriff per CDO 597
  - Informix 309
  - Infostore siehe Information Store
  - Inheritance siehe Vererbung
  - Inherits 211
  - INI-Datei 34, 720
  - In-Memory-Database 122
  - In-process-Komponente 49, 186, 546
  - InprocServer32 116
  - Instancing 187–188
  - Instanz 766
  - Instanzierbarkeit 189
  - Instanziierung entfernt siehe
    - Fernaktivierung
  - Interactive Window-Session 106
  - Interception 117, 120
  - Interface Casting 66
  - Interface Casting siehe
    - Schnittstellenwechsel
  - Interface Definition
    - Language 71, 138, 498, 765
  - Intermediation 158
  - Intermediation Language 158
  - Internal Object siehe Intrinsic
    - Object
  - Internet 33, 159
    - Recherche 30
  - Internet Controls 31, 265, 275
  - Internet Engineering Task
    - Force 379
  - Internet Explorer 31, 350, 379, 467
    - Automatisierung 265, 275
    - Sicherheit 159
  - Internet Information Server
    - 35, 111, 384, 458
    - Administration 458
    - Resource Kit 458
  - Internet Information
    - Services 485
  - Internet Transfer-
    - Komponente 33, 695, 700
  - Internet-Protokoll 740
  - Interpersonal Message Folder
    - Hierarchy siehe Information
      - Store 580
  - Interpreter 109
  - Interprocess Communication 579
  - Intrinsic Object 114, 164
  - Invoke 174
  - Invoke() 81
  - IP
    - Adresse 522, 693
    - Konfiguration 483, 522
  - IPersist 103
  - IP-Routing 485
  - IRQ 484
  - ISAM 307
  - IServerSecurity 108
  - ISO Management
    - Framework 483
  - ISO/OSI-Referenzmodell 85, 691, 695
  - ISPSignup-Komponente 35
  - IStream 103, 332
  - Item 68, 126, 178
  - IUnknown 53, 61, 81, 83, 156, 161
    - Definition 78
- ## J
- J++ 52
  - Java 52, 64, 771, 782
    - Beans 781
    - Enterprise Beans 781, 803
  - Java Application Server 782
  - JavaScript 112
  - JET 307
  - JMAIL 33, 681, 688
  - JScript 112
  - Just-in-Time-Activation 118, 556
  - Just-in-Time-Compiler 158
- ## K
- Kalender, Exchange 5.5 605
  - Kante, Objektdiagramm 792
  - Kapselung 191, 764
  - Kardinalität 124, 771, 793
  - Kennwort ändern 426
  - Kerberos 104, 499
  - Klasse 64, 124, 768
    - abstrakt 212, 766
    - Active Directory 435, 438
    - COM 42, 164, 169, 188
    - definieren 188
    - Definition 766
    - erstellen 187, 192
    - Exchange Server 446
    - Klassenereignis 206
    - Meta-Klasse 127
    - Mitglieder 67
    - Notation 791
    - Oberklasse 768, 777
    - Objektmengenverwaltungs-  
klasse 772
    - Unterklasse 768, 777
    - VB-Klasse 188
    - Verzeichnisdienst 377–378, 391
    - WinNT 420
    - WMI 494, 502
  - Klassenhierarchie 769–770
    - Definition 770
  - Klassenmodul 187, 196
  - Knoten 376
    - Objektdiagramm 791
    - XML 290
  - Knowledge Base 27, 821
  - Kollektion siehe Collection
  - Komponente 29, 42
    - Definition 778
    - entfernt 49
    - Erforschung 30
    - erzeugen mit VB 184, 187
    - Grey-Box 780
    - Hersteller 30
    - In-Process 186
    - Installation 157, 778
    - Konfiguration 778
    - lokal 49
    - Make-or-Buy 30
    - Out-Process 186
    - Parametrisierung 779

## Stichwortverzeichnis

- prozesseextern 49
- prozessintern 49
- Quellcode 115
- Suche 30
- Überblick 30
- verfügbare Komponenten 29
- Version 157
- verwalten 32
- White-Box 780
- Komponentenarchitektur 156, 763
- Vergleich 781–782
- Komponentendienst 145–146, 148–149
- Komponentenerforschung 753
  - Analyse 758
  - Dokumentation 759, 761
  - Implementierung 761
  - Installation 758
  - Newsgroup 757
  - Suche 753
  - Test 760
  - Typbibliothek 758
- Komponentenkategorie siehe COM-Kategorie
- Komponentenprojekt 188
- Komprimieren 33
- Konfiguration
  - CDO 3.0 658
  - CDOSYS 685
  - Exchange 2000 670
- Konstante 192, 787
  - Definition 144
  - Gebrauchshinweis 787
  - Konstantenliste 75
  - symbolisch 75
- Kontakt
  - Exchange 2000 629, 669
  - Exchange 5.5 605
- Kontakt im Active Directory 438
- Kontext 117
- Kontextmenü 268, 273
- Kontorichtlinie 422
- Konvention
  - Formatierung 784
  - Icon 785
  - Komponenten-Schnellinfo 794
  - Listing 786
  - Nutzungsbeziehung 792
  - Objektdiagramm 789
  - Sprache 783
- Kosten 30
- L**
- LAN 691
- Late Binding siehe Binden, spät
- Laufwerk 251, 254, 516
  - Laufwerksbezeichnung 517
- Laufzeitumgebung 41, 156, 183
  - Visual Basic 161
- LDAP siehe Lightweight Directory Access Protocol
- Leaf 376, 386, 403
- Lesezeichen 327
- Lightweight Directory Access Protocol 35, 376, 444
  - Namen 380
  - Query-Syntax 413
- Lightweight RPC 50
- Lisp 112
- Liste, verkettete 771
- Little Endian 381, 413
- Lizenz 57
- Local Procedure Call 50
- Log File Event Consumer 501
- Lokalisierung 493, 503
- Lotus Notes 33, 37, 309, 379, 580, 815
- LUA 112
- LUAScript 112
- M**
- Mabry 33, 702
- Mailbox siehe Postfach
- Mailkomponente 681
- Mailserver, CDOSYS 685
- Mailstore 579
- Make-or-Buy 30
- Managed Object 483, 489
- Managed Object Format 498, 539, 543
  - Compiler 543
- Manifest 44
- MAPI siehe Messaging Application Programming Interface
- Mars 789
- Marshalling 50–51
- Master Property 464
- MDAC siehe Data Access Components 310
- Me 197
- Medium 254
- Mehrfachschnittstelle 123, 163, 172, 210–211, 214, 765
  - CDO 3.0 657
  - CDO.Message 660
  - erstellen 212
- Member siehe Mitglied
- Merant 634
- Message Queue Service 485
- Message Store 35, 306
- Message Store Provider 580
- Message Transport Provider 581
- Message, CDO 1.21 585
- MessageFilter 585
- Messaging 33
- Messaging Application Programming Interface 25, 33, 444, 579
  - Feld 590
  - ID 587
  - Klasse 581, 588
  - MAPI-Explorer 581
  - MAPI-Profil 581
  - Nachrichtenklasse 605
  - Property Tag 591
  - Provider 581
- Metabase 458
- Meta-Directory 378
- Meta-Klasse 127, 398
  - WMI 505
- Meta-Objektmodell
  - ADO 318, 320
  - ADSI 375, 382, 385
  - CDO 581
  - COM+ Administration Objects 545, 548
  - Definition 126
  - WMI 489
- Meta-Schnittstelle 386
- Methode 124
  - Aufruf in VB 170
  - Definition 763
  - erstellen 187, 191
  - Notation 794
- MFC siehe Microsoft Foundation Classes
- Microsoft 67
- Microsoft Access 37
- Microsoft Agent 37
- Microsoft BackOffice 22, 26, 38
- Microsoft BizTalk Server 111
- Microsoft Commerce Server 111
- Microsoft Common Dialog Control siehe Common Dialog Control
- Microsoft Excel 37

## Stichwortverzeichnis

- Microsoft Exchange siehe
  - Exchange Server
- Microsoft Foundation Classes 98
- Microsoft FrontPage 37
- Microsoft Graph 37
- Microsoft Intermediation
  - Language 158
- Microsoft Internet Control
  - 31, 175
- Microsoft Mail 580
- Microsoft Message Queue
  - Service 485
- Microsoft Office 26, 37, 162, 273, 751
- Microsoft Operations
  - Manager 111
  - Hilfsobjekte 735
- Microsoft Outlook 33, 37
- Microsoft PowerPoint 37
- Microsoft Routing Objects 35
- Microsoft Transaction Server
  - 92, 147, 167
  - Administration Objects 556
  - PAK-Datei 553
- Microsoft Transaction Server
  - siehe MTS
- Microsoft Word 37
- Middleware 27, 780
- Minbar 789
- MINFU 82, 100, 489, 547
- MiniOLE 31, 34
- Mitglied, Definition 764
- Mitglieder 190–191
- MMC 369, 487, 547
- Modellierung 124, 126–127, 155, 771–773, 775–776
- MOF siehe Managed Object
  - Format
- MOM siehe Microsoft Operations
  - Manager
- Moniker 46, 93, 277, 397, 416, 821
- MSDB 559
- MSDN Library 560, 756
- MsgBox() 787
- MSHTML 31, 267, 275, 279
  - Programmierung 280
- MSI 120
- MSIL 158
- MSMQ siehe Microsoft Message
  - Queue Service
- MSQL siehe SQL Server
- MSXML 34, 279–280, 288
- MTS 32, 116, 120, 165
  - Katalog 118
  - Package 118
- MTS Explorer 147
- Multi-Document-Interface 92
- Multinationale
  - Unternehmen 239
- MultiUse 189
- Musterersetzung 716
- Mustervergleich 705, 708
- N**
- Nachricht 579, 764
  - anlegen (CDO 1.21) 610
  - anlegen (CDO 3.0) 663
  - CDO 1.21 604
  - CDO 3.0 658
  - Exchange 2000 629–630, 648
  - Exchange 5.5 585
  - löschen 611
  - senden 33
- Nachrichtenkommunikation 33
- Namensraum 492
- Namespace 267
  - ADSI 407, 417
  - Verzeichnisdienst 376
  - WMI 487, 492
- NDS 393
- Netscape
  - Communicator 379
  - Directory Server 379
  - Navigator 280
- Netware Directory Service 376
- Network Data Representation 51
- Network News Transport
  - Protocol 664, 684
- Network OLE siehe Distributed
  - Component Object Model
- Netzlaufwerk 227, 484
- Netzwerkkarte 484, 522
- Netzwerkcommunication 33
- Netzwerkmanagement 483
- Netzwerknachricht 482
- Netzwerkverbindung 225, 485
- Neustart 531
- New 163–164, 166, 189
  - Instanziierung COM-
    - Klasse 166
  - Klasse RegExp 706
  - Vergleich zu
    - CreateObject() 166
- Newsgroup 757
  - CDOSYS 684–685
- Notation
  - Objekmodell 773
  - umgekehrt polnische 413
- Notes siehe Lotus Notes
- Nothing 176, 184
- Notiz
  - Exchange 2000 629
  - Exchange 5.5 605
- Novell 375–376, 384
- Novell Directory Service 393
- NSDPGP 34
- NT 35
  - Dienst 32
- NT Event Log Event
  - Consumer 501
- NT-Dienst 53, 737
- NT-Domäne siehe Domäne
- NTFS 33, 489
- ntlanmanserver 432
- NTRights.EXE 428
- Nutzer 765
- Nutzungsbeziehung 770
  - Notation 792
- O**
- Object (Datentyp) 168, 176
- Object Browser siehe
  - Objektkatalog
- Object Construction String 121
- Object Linking and
  - Embedding 76
  - Definition 27
- Object Management Group
  - (OMG) 782
- Object Modelling Technique 774
- Object Pooling 121
- Object REXX 112
- Object RPC 84
- Object Trader 89
- Objekt 763, 768, 778
  - ADSI 398
  - Definition 763
  - eingebaut 164
  - Notation 791
  - Operationen 176
  - Typüberprüfung 163
  - Vergleich 163
  - Verzeichnisdienst 376
  - Wertzuweisung 163
  - WMI 502

## Stichwortverzeichnis

- Objektcontainer 102
  - Objektdiagramm 775
    - Definition 774
    - Notation 789
  - Objektfabrik 766
  - Objekthierarchie 770, 780
    - Definition 770
    - Verzeichnisdienst 378
  - Objektidentifikation
    - ADSI 397
    - Exchange 2000 629
    - LDAP 381
    - Verzeichnisdienste 378
  - Objektkatalog 62, 75, 126, 140, 143, 535
  - Objektkontext 117, 119–120, 167
  - Objektmenge 68, 124, 178, 306, 774, 793
    - heterogen 774
    - homogen 774
  - Objektmenge siehe Collection
  - Objektmodell 30, 38, 122, 153, 770
    - Active Directory 436
    - ADO 318
    - ADSI 385
    - auf Anwendungsebene 776
    - auf Modellierungsebene 127, 776
    - CDO 1.21 583
    - CDO 2.0 684
    - CDO 3.0 658
    - CDOEXM 668
    - CDONTS 683
    - COM+ Administration
      - Objects 548
    - COM+-Katalog 546
    - Definition 770, 774
    - DMO 560
    - DOM 279
    - DTS 577
    - Erforschung 760
    - Exchange Server 445
    - FSO 252
    - IIS 458
    - Internet Controls 276
    - Meta-Objektmodell 127
    - Microsoft Office 751
    - MSHTML 281
    - MSXML 290
    - Notation 773
    - RegCol 361
    - RegExp 706
    - Shell Objects 266
    - STMAAdmin 370
    - Taskscheduler 352
    - WindowsScripting 719
    - WinNT 419
    - WMI 505
    - WSH Runtime 218
  - Objektorientierung 29, 151, 483, 763
  - Objektpersistenz 102
  - Objektsammlung siehe Collection
  - Objektvariable 163, 171, 173, 765
    - Definition 764
  - Objektvergleich 176
  - Objektvernichtung 183
  - Objektzeiger 64
  - OCX 52
  - ODBC siehe Open Database Connectivity
  - ODBCTools 34
  - Öffentlicher Ordner 35, 580
    - Deep Traversal 636
    - Exchange 2000 630–631, 670
    - Exchange 5.5 605
    - Größe ermitteln 748
  - Öffnen-Dialog 729
  - Office 37
  - Office 2000 185, 499
  - Office siehe Microsoft Office
  - OLAP siehe Online Analytical Processing 312
  - OLE 305
  - OLE DB 305–306, 318, 412
    - Architektur 306
  - OLE DB, Provider 306, 323
  - OLE DB-Provider
    - EventLog 369
    - Exchange 5.5 634
    - für FTP-Verzeichnisse 702
    - Internet Publishing 307, 340, 633
  - OLE Messaging 582
  - OLE siehe Object Linking and Embedding 27
  - OLE/COM Object Viewer siehe COM-Viewer
  - OLEDB, Provider für Oracle 34
  - OnePoint Utility Objects 31–32, 35, 735
  - OnePoint, Hilfsobjekte 735
  - Online Analytical Processing 34, 312
  - OO4O siehe Oracle Objects for OLE
  - Open Database
    - Connectivity 305–307, 527
    - Einstellung 485
  - Open Group 152, 813
  - Open Software Foundation 45, 84
  - Operations Manager siehe Microsoft Operations Manager
  - Option Pack 681
    - NT4.0 116
  - Oracle 34, 307
    - Objects for OLE 34
  - Ordner
    - anlegen (CDO 3.0) 663
    - CDO 599
    - Dateisystem 251, 254, 484
    - Eigenschaft 256
    - Exchange 2000 629, 631, 642–643
    - Exchange 5.5 585
    - FSO 274
    - Größe 36
    - Rechte 470
    - Shell Objects 266, 271, 274
  - Ordnerhierarchie 580
    - Exchange 2000 670
    - Exchange 5.5 579
  - Organisation
    - ADSI-Schnittstelle 390
    - im Exchange Server 446
    - LDAP 381
  - Organisationseinheit 390
    - im Active Directory 438, 443
    - im Exchange Server 446
  - Organizational Unit 381
  - Outlook 111, 751
  - Outlook 2000 142
  - Outlook Express 33, 380
  - Out-process-Komponente 49, 186
- P**
- Passwort siehe Kennwort
  - Performance Counter 489
  - Performance Counter
    - Provider 532
  - Performance Monitor 484, 489
  - Perl 248
  - PerlScript 112



## Stichwortverzeichnis

- Persistenz 102
  - Persönlicher Ordner 580–581, 612
  - Persönliches Adressbuch 581, 612
  - Pfad
    - ADSI 397
    - Exchange 2000 627
    - LDAP 381
    - Registry 365
    - WMI 502
  - PHP4 112, 782
  - Physical Packaging 52
  - Ping 485, 741–742
  - Plattformunabhängigkeit 158
  - Plug&Play 109, 778
  - Polymorph 211
  - Polymorphie siehe Polymorphismus
  - Polymorphismus 77, 122, 163, 777
    - Definition 776
  - Popup 229
  - Portscanner 694
  - Posteingang, Exchange 2000 629
  - Postfach 446, 579–580
    - anlegen 452
    - anlegen (CDO 3.0) 673
    - anlegen (CDOEXM) 671
    - bewegen (CDOEXM) 671
    - Eigenschaften (CDOEXM) 675
    - Exchange 2000 626, 670
    - Größe 36
    - Größe ermitteln 745–746
    - löschen 456
    - löschen (CDOEXM) 671
  - Postfix-Notation 413
  - Power Management 489
  - PowerPoint 751
  - Pretty Good Privacy 34
  - Principal 104
  - Private 189, 191, 212
  - Privilege, WMI 499
  - Profile Provider 581
  - ProgID 58–60, 135, 162, 169, 189, 554
    - Moniker 46
  - Programmiersprache
    - Active Scripting-fähig 112
    - COM-fähig 112
  - Projekt 188, 195
    - VB 185
  - Projektdatei 798
  - Projekttypen 185
  - Property
    - ADODB 319
    - Default 202
  - Property Cache 402
  - Property Tag 591
  - Property-Routine 197–198, 212
  - Protokolldatei 501
  - Provider
    - ADSI 383–384
    - Messaging Application Programming Interface 580–581
    - OLE DB 306
    - WMI 488
  - Proxy 51
  - Prozess 484
    - auflisten 526
    - beenden 517
    - starten 517, 741–742
  - Prozessortyp 158
  - PScript 112
  - Public 191
  - PublicNotCreatable 189
  - PythonScript 112
- Q**
- Qualifier, WMI 495
  - Quellcodekomponente 115
  - QueryInterface 167, 174
  - QueryInterface() 61, 65, 79, 91, 101, 136, 171
  - Queued Component 121
  - Quota-Management 266
- R**
- Rapid Application Development 115, 792
  - RAS/X 33
  - Raw-Modus 449
  - Reboot, Neustart 531
  - Recipient 585
  - Recipient siehe Empfänger
  - Record 636
    - ADODB 319
  - Recordset 319–320, 332
    - dynamisch 330
    - Exchange 2000 635
    - Persistenz 330
    - Typen 322
  - Referenz, zirkulär 158
  - Referenzzählung 53, 158
  - RegCol 32, 361
  - RegEdit 131, 133, 755
  - RegEdt32 108, 131
  - RegExp 705
  - Registrierung 133, 187
  - Registry 29, 32, 53, 56, 59–60, 66, 75, 157, 167, 230, 361, 385, 468, 485, 489, 783
    - Änderung 367
    - COM 96
    - Editor 131, 136, 162
    - Klasse 363
    - Lesen 365
    - Rechte 362, 470
    - Registry Crawler 132
    - Suche 132–133
  - RegSvr32 56, 133
  - Regulärer Ausdruck 705
  - Rekursion 366, 404, 600
  - Relation 306
  - Relative Distinguished Name 378, 381
  - Release() 79, 158
  - Remote Access Service 33, 468
  - Remote Data Objects 309
  - Remote Data Service 87, 311
  - Remote Procedure Call 45, 84
  - RemoteServerName 89
  - Remoting 157
  - Remove() 180
  - Replikation 436, 451, 482, 485
  - Repository, WMI 492
  - Request for Comment 820
  - Resource Kit
    - ADSI 385, 433, 453, 464
    - IIS 458
    - NT4 428
    - Windows 2000 135, 428
  - Response 164
  - Ressource 375, 391, 433
  - REXX 112
  - Rich-Text 34
  - Root 376
  - ROT siehe Running Objects Table
  - Routing 485
  - Routing and Remote Access Service 806
  - Routing Objects 35
  - RPC siehe Remote Procedure Call 45, 84
  - Ruby 112
  - Rückgabewert 170, 192

## Stichwortverzeichnis

- Rückruf 470
- Rule-Komponente 35
- Running Objects Table 92, 168
  - GetObject() 164
  - Nutzung in VB 164
  - ROT Viewer 149
- S**
- SAP R/3 37, 309
- Saturn 789
- say() 787
- Scheduler 32
- Scheduler siehe Taskscheduler
- Scheduling Agent siehe
  - Taskscheduler
- Schema 779
  - Active Directory 436, 438
  - Exchange 2000 628
  - Exchange Server 445, 449
  - IIS 459
  - Modifikation 378
  - Verzeichnisdienst 377, 409
  - WinNT 421
  - WMI 491, 512
  - XML 297
- Schlüsselattribut 381
  - WMI 495
- Schlüsselwert 178
- Schnittstelle 64–65, 190–191
  - Definition 765
  - dual 83, 162
  - in VB 171
  - Standardschnittstelle 190
  - Test 172
  - Wahl 163, 171
  - Wechsel 66, 171, 211
- Schnittstellenvererbung 211
- Schnittstellenwechsel 64
  - Wechsel 171
- Schriftarten-Dialog 732
- Scope 184
- Screen Scraping 239
- Scriptlet 185
- Script Component Runtime 116
- Script Control 111
- Scripting Engine 109, 111
- Scripting Host 75, 109–110
- Scripting Newsgroup 820
- Scripting Runtime Library
  - 38, 155, 247
- Scripting-Architektur 110
- Scriptlet 52, 115, 162, 185
- Scriptor Component 111
- ScrObj 133
- SCT 52, 56, 115
- Secure Socket Layer 695
- Security Descriptor 391, 453, 471, 477
- Security Identifier 427, 471
- Security Service Provider
  - 104, 499
- Selbstbeschreibung 156
- Selbstreferenzierung 197
- SELECT (SQL) 501
- SendKeys 239
- Serveranwendung 37
- Serverdefinition 781
- Service Control Manager 89
- Service Provider 306
- Set 124, 774
- Shallow Traversal 636
- Shared Property Manager 118
- Shareware 33
- Shell Objects 32, 125, 265, 275
- Sicherheit
  - Active Directory 470
  - ADSI 389, 407
  - ADsSecurity 470
  - CDO/MAPI 595
  - COM 104, 499
  - Dateisystem 252, 470, 485, 489
  - Datenbank 312
  - Exchange 470
  - Exchange-Postfach 452, 455
  - Freigabe 485, 532
  - Registry 470
  - Sicherheitsprotokoll 371
  - SQL Server 566
  - Taskscheduler 349, 359
  - Überblick 471
  - WMI 487, 499, 504, 506
  - WMI-Pfad 502
- Sicherungsmedium anlegen 574
- Sichtbarkeit 189
- Side-by-Side-Execution 97
- Signatur 100
- Simple Mail Transport
  - Protocol 33, 664, 681
  - E-Mail-Adresse 456
- Simple Network
  - Management 483–484, 489
- Simple Object Access
  - Protocol 88, 157
- SimpleIISAdmin 463
- SingleUse 189
- Site Server 32, 379, 681
- Sitzung 337, 390–391, 433
- Skript, Verschlüsselung 247
- Skriptsprache 66, 89
- Skript-Steuerelement 28
- SNA Server 489
- Software 484
- Softwareentwicklung,
  - komponentenbasiert 778
- Softwareentwicklungsprozess 30
- Solaris 152, 813
- Sonderordner
  - FSO 252
  - Shell Objects 271
  - WSH 243
- Sonderzeichen 261
- Sonne 788
- Sonnensystem 788
- Speichergruppe, Exchange
  - 2000 670
- Speichern-Dialog 731
- Speicherplatz 183
- Spitzname 46
- Sprache 239
  - objektorientiert 764
- Sprache siehe
  - Programmiersprache
- Sprechblase 790
- SQL 321, 333, 338, 373, 413, 501
  - Exchange 2000 636
- SQL Server 34, 111, 307, 310
  - Automatisierung 559
- SQL-DMO siehe Distributed
  - Management Objects 559
- Stammklasse 125, 775
- Standard Marshalling 51
- Standardattribut 202
- Standardattribut siehe Default
  - Property
- Standardausgabe 247, 260
- Standarddialog 31, 725
- Standardeingabe 247, 260
- Standardschnittstelle 62, 190, 212
- Standort
  - Active Directory 482
  - Exchange 444
- Startmenü 265, 484
- Static Method Invocation 76
- Steuerelement siehe ActiveX-
  - Steuerelement
- STMAdmin 31, 369

## Stichwortverzeichnis

- Storage 102
  - Stream 263
    - ADO 319, 331
    - Structured Storage 102
  - String 175, 198
  - Structured Query Language
    - siehe SQL
  - Structured Storage 47, 102
  - Stub 51, 117
  - Sub 191
  - Submatch 715
  - Sun 781
  - Support Tools 465, 481
  - Surrogat-Prozess 55, 91, 149, 165
  - SWbemObject 508
  - SWBRegSvr 134
  - Syntaxbeschreibung 391
  - System Management Server 489
  - System Scripting Host 69
  - Systemattribut, WMI 497
  - Systemklassen, WMI 495
  - Systemmanagement 31, 483
  - Systems Management
    - Server 483, 502
  - Systemzugriff 25
- T**
- Tabelle 320
    - anlegen 571
    - löschen 574
  - Tabellenkalkulation 306
  - TAPI siehe Telefon Application
    - Programming Interface
  - Task 350
  - Taskleiste 265
  - Task-Manager 89
  - Taskscheduler 32, 349, 485
    - Impersonifizierung 359
  - TCL 112
  - TCP/IP 33, 379, 691, 740
  - Teilstring 715
  - Telefon Application
    - Programming Interface 33
  - Termin, Exchange 2000 629
  - Terminal Service 486
  - Terminaldienstprofil 442
  - Terminalserver 429, 442
  - Text Object Model 34
  - Textdatei 34, 247, 260, 262, 306
  - Textstream 260
  - TLB-Datei 74
  - Tooltip 273
  - Trader 101
  - Trading 89
  - Transaktion 116, 344
    - COM+ 119
    - MTS 116
    - Programmierung 119
  - Transaktionsprotokolldatei,
    - anlegen 568
  - Transmission Control Protocol
    - siehe TCP/IP
  - Transparenz 780
  - Tree 436
  - Treiber 305
  - Trigger 350
  - True64UNIX 152
  - Trustee 471
  - Typ 766
  - Typbibliothek 43, 74, 138, 155, 162,
    - 507, 758, 787
    - DMO 561
  - Type Library siehe Typbibliothek
  - TypeInfo-Viewer 138
  - TypeName 168, 176
  - TypeOf 172, 177
  - Typinformation 66, 69, 176
  - Typisierung 786
  - TypLib Marshalling 51
  - Typüberprüfung 163, 176
- U**
- UDL 315, 325
  - UDP 691
  - Uhrzeit 486
  - Umgebungsvariable 484, 741
  - UMI siehe Universal
    - Management Interface
  - Unicode 260, 314
  - Uniform Resource Identifier
    - Definition 627
    - Exchange 2000 627, 664
    - XML 297
  - Uniform Resource Locator 46,
    - 159, 276, 381
    - Definition 627
    - Exchange 2000 627, 629
  - Uniform Resource Name
    - Definition 627
    - Exchange 2000 627
  - Universal Coordinated Time 496
  - Universal Data Access 305, 383
  - Universal Management
    - Interface 486, 806
  - Universal Marshaler 51
  - Universalklasse siehe Meta-
    - Klasse
  - Universum 789
  - Unix 152
  - Unterobjekt 170, 196, 398, 793
  - Unterroutine 191
  - UNZIP 33
  - User Datagram Protocol 691
- V**
- Variable
    - Deklaration 786
    - global 191
  - Variablenname 786
  - Variant 168, 175, 197, 403
    - ADSI 403
  - VarType 176
  - VB Runtime 41
  - VB6/7/A, Definition 783
  - VBCCE 185
  - VBInstance 542
  - VBScript 112, 161, 169, 185
    - eingebaute Klasse 705
    - Komponente erzeugen 185
  - VB-Vollversion 783
  - Vendor Log-In 782
  - Verbindung 337
  - Verbindungspunkt 69
  - Verbindungszeichenfolge siehe
    - Connection-String
  - Vererbung 123, 210, 766
    - Definition 768
    - Implementierungsvererbung 769
    - in .NET 157
    - Schnittstellenvererbung 769
  - Vererbungsbeziehung 770
  - Vererbungshierarchie 537, 769
    - Definition 770
    - Verzeichnisdienst 378
    - WMI 491
  - Vergleich 176
  - Verkettung,
    - Objektverkettung 771
  - Verknüpfung 272
    - Dateisystem 243
    - URL 246
  - Verschlüsselung 34
  - Versionierung 123, 157
  - Versionsermittlung 311
    - ADSI 465
    - CDO 582

## Stichwortverzeichnis

- Verteiler 456
    - Exchange 2000 626
  - Verteiltes System 27
  - Verteilungsinfrastruktur 780
  - Verteilungsplattform siehe Middleware
  - Vertrauensstellung 482
  - Verwaltungs-klasse 772
  - Verwaltungsobjekt 772–773
  - Verweise 798
  - Verweise-Dialog 141, 753
  - Verzeichnisbaum 255–256, 375
  - Verzeichnisdienst 35, 306–307, 375, 418, 435, 488
    - Exchange 579
  - Verzeichnisfreigabe siehe Freigabe
  - Verzeichnisobjekt 398
  - Vielgestaltigkeit siehe Polymorphismus
  - Visio 67
  - Visual Basic 26, 64, 161, 172
    - .NET 211
    - Definition 783
    - Dialekte 161
    - Intrinsic Object 164
    - Klasse erzeugen 185
    - Komponente erzeugen 185
    - Laufzeitumgebung 161
    - Objektmenge 772
  - Visual Basic for Applications 111, 161, 719
    - Komponente erzeugen 185
  - Visual InterDev 141, 261
  - Visual J++ 52
  - Visual Studio 69, 135, 725
  - Visual Studio 6.0 34
  - Visual Studio.NET 143
  - Vorgehensmodell 23, 30, 38, 753
  - vTable 63, 80–81, 83
    - Binding 77, 81, 290
  - VTBL siehe vTable 63
- W**
- W3Svc 458
  - WAN 691
  - Warteschlange 434
  - Warteschleife 741
  - WBEM siehe Web Based Enterprise Management
  - Web Based Enterprise Management 483
  - Webbrowser 267, 281
  - Web-Scripting, DNA 29
  - Webseite 239
  - Webserver 35, 460
    - anlegen 462
    - löschen 463
  - Website zu diesem Buch 797
  - Webstore 308, 642
  - Well Known Security Principal 474
  - Werkzeug
    - ADSI 393
    - COM 131
  - Wertzuweisung 163, 176
  - Whistler 44, 145, 149, 152, 486, 493, 501, 507, 526, 789
  - Win32-API 26
  - Windows
    - Bildschirmmaske 31
    - Dienst 32
    - Fenster 28
    - Sicherheit 29
    - Standarddialog 31
  - Windows .NET 44, 152
  - Windows 2000 97, 493
    - Professional 419
    - Server 419
  - Windows 2002 145, 149, 486, 493, 526, 789
  - Windows 95 149
  - Windows 98 149
  - Windows 9x 85, 87, 109, 145, 233, 350, 424, 486, 783
  - Windows Driver Model 489
  - Windows Installer 111, 120, 488
    - Automatisierung 31
    - MSI-Datei 553
    - Objektmodell 31
  - Windows Management Instrumentation 31–32, 38, 105, 165, 483, 535, 740
  - ADSI-Integration 529
  - Asynchroner Aufruf 525
  - CIM Studio 537
  - Class Explorer 537
  - COM-Komponente 489
  - Data Query 521
  - Dienste starten/stoppen 358
  - entfernter Zugriff 516
  - Ereignis 491, 500, 540
  - Erforschen 535
  - Event Query 501, 523
  - Event Viewer 541
  - Exchange 2000 37, 627
  - Instanz 537–538
  - Kernel 486
  - Klasse 494, 536–537
  - Konfiguration 487
  - lokaler Zugriff 510
  - Meta-Objektmodell 130
  - Namespace 492, 495, 536
  - Object Browser 535, 537, 543
  - Object Explorer 537
  - Objektmodell 505
  - ODBC-Treiber 527
  - Provider 488, 493
  - Query 521
  - Query Language 491, 501, 521, 539
  - Repository 500
  - Schema 491, 512, 522
  - Schema-Query 522
  - Scripting API 489
  - SDK 491
  - semi-synchroner Aufruf 524
  - Testprogramm 543
  - Werkzeug 487, 535
  - Werteänderung 516
- Windows ME 85, 87
  - Windows NT 375
  - Windows Script Component 52, 115–116, 133, 185
  - Windows Script Host 69, 94, 114, 217
    - Ausgabe 229, 277
    - Impersonifizierung 359
    - Intrinsic Object 164
  - Windows Scripting 26
  - Windows Terminal Services 429
  - Windows Whistler siehe Whistler
  - Windows XP 44, 87, 98, 145, 149, 152, 486, 493, 501, 507, 526
  - Windows.NET 87, 145, 149
  - Windows-Explorer,
    - Automatisierung 265
  - Windows-Hilfe 728
  - WindowsScripting-Komponente 34, 719
  - Windows-Sicherheit 471
  - WinMgmt.exe 486
  - WinSock-Komponente 33, 691
  - WinSock-Steuerelement 28
  - With 174
  - WMI Query Language, Extended WQL 502

## *Stichwortverzeichnis*

- WMI siehe Windows Management Instrumentation
- Word 751
- Workflow 35
- World Wide Web Consortium 279
- WQL siehe Windows Management Instrumentation Query Language
- WriteTo() 722
- WSC siehe Windows Script Component
- WSH Objects siehe WSH Runtime Library
- WSH Runtime Library 31–32, 38, 153, 217
- WSHNetwork 225
- WSHShell 229
- Wurzel siehe Root
- Wurzelklasse siehe Stammklasse
- WWW-Dienst 458
- X**
- X.500 376, 379
- XCOPY-Deployment 100, 157
- XDR siehe XML-Data Reduced
- XML Schema Definition 288, 808
- XML siehe Extensible Markup Language 288
- XML-Data Reduced 288, 803
- XMLHTTP 288
- XSD siehe XML Schema Definition
- Z**
- Zeiger 163
- Zeitplandienst 32
- Zeitzone 485
- ZIP 33
- Zirkuläre Referenz 124