

net.com
networking & communications

HOLGER SCHWICHTENBERG

2., aktualisierte Auflage

Windows PowerShell 2.0
Das Praxisbuch

Einführung und Lösungen für Windows-Administratoren

- Für Windows XP/Server 2003/Vista/Server 2008 & Windows 7
- Kommandozeile und Scripting
- Zahlreiche Praxisbeispiele

ADDISON-WESLEY

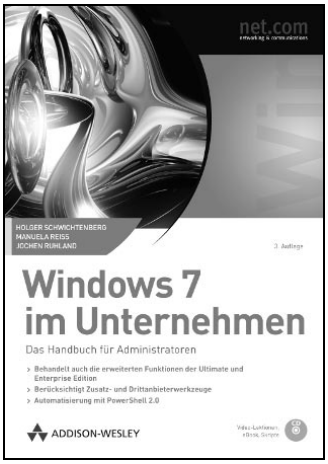
www.IT-Visions.de
Dr. Holger Schwichtenberg

CD

Windows PowerShell 2.0
Das Praxisbuch

net.com
networking & communications

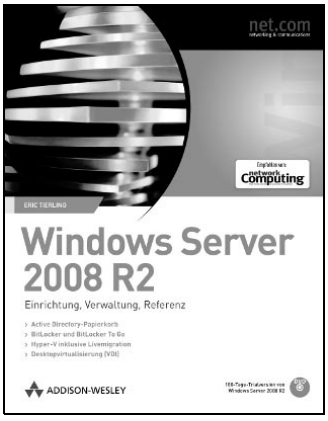
Netzwerke, Betriebssysteme, Sicherheit ... hierzu bietet Ihnen die Reihe net.com umfassende, praxisnahe Information. Neben Fragen der Systemverwaltung greift sie auch Themen wie Protokolle, Technologien und Tools auf. Profitieren Sie bei Ihrer täglichen Arbeit vom Praxiswissen unserer erfahrenen Autoren.



Windows 7 im Unternehmen

Holger Schwichtenberg/Manuela Reiss/Jochen Ruhland
1088 Seiten, € 59,80 [D]
ISBN 978-3-8273-2886-1

Fortgeschrittene Anwender und Administratoren erhalten hier fundierte Informationen für den Einsatz von Windows 7 im Unternehmen. Die Autoren beschreiben die Benutzeroberfläche und deren Konfigurationsmöglichkeiten ebenso wie Konzeption und Einrichtung von Benutzer- und Systemverwaltung im Netzwerk sowie Sicherheitsaspekte. Weitere neue und wichtige Themen sind u.a. der Remote-Zugriff aufs Netzwerk, die Zweigstellenverwaltung und die Automatisierung mit PowerShell 2.0.



Windows Server 2008 R2

Eric Tierling
1680 Seiten, € 69,80 [D]
ISBN 978-3-8273-2907-3

Dieses Buch zu Windows Server 2008 R2 knüpft an die Bestseller zu Windows Server 2003 und 2008 an und wurde komplett auf die Neuerungen von R2 aktualisiert. Installation auf virtueller Festplatte, Remoteverwaltung mit dem Server-Manager, Verwaltung über Windows PowerShell-Cmdlets, IPv6, Active Directory-Papierkorb, Einrichtung verwalteter Dienstkonten, Offline-Domänenbeitritt, AppLocker-Anwendungssteuerungsrichtlinien, DirectAccess, BranchCache, die Remotedesktopdienste sowie Hyper-V sind einige der im Buch beschriebenen Highlights.

Holger Schwichtenberg

Windows PowerShell 2.0

Das Praxisbuch



An imprint of Pearson Education

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

Bibliografische Information der Deutschen Nationalbibliothek

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über *http://dnb.d-nb.de* abrufbar.

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

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

Fast alle Hard- und Softwarebezeichnungen und weitere Stichworte und sonstige Angaben, die in diesem Buch verwendet werden, sind als eingetragene Marken geschützt.
Da es nicht möglich ist, in allen Fällen zeitnah zu ermitteln, ob ein Markenschutz besteht, wird das ®-Symbol in diesem Buch nicht verwendet.

10 9 8 7 6 5 4 3 2 1

12 11 10

ISBN 978-3-8273-2926-4

© 2010 by Addison-Wesley Verlag,
ein Imprint der Pearson Education Deutschland GmbH,
Martin-Kollar-Straße 10–12, D-81829 München/Germany
Alle Rechte vorbehalten
Einbandgestaltung: Marco Lindenbeck, mlindenbeck@webwo.de
Fachlektorat: Rolf Masuch, rolf@powershell-ag.de
Lektorat: Sylvia Hasselbach, shasselbach@pearson.de
Korrektorat: Sandra Gottmann, sandra.gottmann@t-online.de
Herstellung: Philipp Burkart, pburkart@pearson.de
Satz: mediaService, Siegen, www.media-service.tv
Druck und Verarbeitung: Kösel, Krugzell (www.KoeselBuch.de)
Printed in Germany

Inhaltsverzeichnis

Vorwort zur zweiten Auflage 25

Danksagungen 27

Über den Autor Dr. Holger Schwichtenberg 27

Teil 1 PowerShell-Basiswissen 29

1 Erste Schritte mit der Windows PowerShell 31

1.1 Was ist die Windows PowerShell? 31

1.2 Geschichte 32

1.3 Betriebssysteme mit PowerShell 36

1.4 PowerShell herunterladen und installieren 37

1.5 Die Windows PowerShell testen 39

PowerShell im interaktiven Modus 39

PowerShell im Skriptmodus 41

1.6 PowerShell Community Extensions herunterladen und installieren 44

1.7 Die ISE verwenden 46

2 Architektur der Windows PowerShell 49

3 Einzelbefehle der PowerShell 53

3.1 Commandlets 53

Commandlet-Parameter 54

3.2 Aliase 56

Aliase auflisten 56

Neue Aliase anlegen 61

3.3 Ausdrücke 63

3.4 Externe Befehle 64
3.5 Dateinamen 66

4 Hilfefunktionen 67
4.1 TechNet-Dokumentation 67
4.2 Verfügbare Befehle 68
4.3 Erläuterungen zu den Befehlen 70
4.4 Hilfe zu Parametern 72
4.5 Dokumentation der .NET-Klassen 72

5 Objektorientiertes Pipelining 75
5.1 Grundlagen 75
5.2 Pipelining mit Objekten 75
Zugriff auf einzelne Objekte 77
Zugriff auf einzelne Werte 77
Methoden ausführen 79
Pipelining von Parametern 80
Pipelining von klassischen Befehlen 81
5.3 Pipeline Processor 81
5.4 Komplexe Pipelines 84
5.5 Analyse des Pipeline-Inhalts 85
Methode GetType() 86
Get-PipelineInfo 87
Get-Member 88
Methoden (Mitgliedsart Method) 90
Eigenschaften (Mitgliedsart Property) 90
Eigenschaftssätze (PropertySet) 92
Notizeigenschaften (NoteProperty) 93
Skripteigenschaften (ScriptProperty) 93
Codeeigenschaften (Code Property) 94
Aliaseigenschaft (AliasProperty) 95
Hintergrundwissen: Extended Type System (ETS) 95
5.6 Filtern 96
5.7 Zusammenfassung von Pipeline-Inhalten 98

5.8 „Kastrierung“ von Objekten in der Pipeline 99
5.9 Sortieren 100
5.10 Gruppierung 100
5.11 Berechnungen 102
5.12 Zwischenschritte in der Pipeline mit Variablen 102
5.13 Verzweigungen in der Pipeline 103
5.14 Vergleiche zwischen Objekten 104
5.15 Praxisbeispiele 104

6 PowerShell-Skripte 107
6.1 Skriptdateien 107
6.2 Start eines Skripts 108
6.3 Aliase für Skripte verwenden 110
6.4 Parameter für Skripte 110
6.5 Skripte dauerhaft einbinden (Dot Sourcing) 112
6.6 Sicherheitsfunktionen für PowerShell-Skripte 112
Sicherheitsrichtlinien 113
Skripte signieren 114
6.7 Skripte anhalten 116

7 PowerShell-Skriptsprache 117
7.1 Hilfe zur PowerShell-Skriptsprache 117
7.2 Befehlstrennung 118
7.3 Kommentare 118
7.4 Variablen 119
Typisierung 120
Typisierungszwang 122
Gültigkeitsbereiche 123
Vordefinierte Variablen 123
7.5 Zahlen 126
Zufallszahlen 127
7.6 Zeichenketten 128
Bearbeitungsmöglichkeiten 128
Zeichenketten trennen und verbinden 130

7.7 Reguläre Ausdrücke 131
Beispiel: Prüfung einer E-Mail-Adresse 131
Allgemeiner Aufbau von regulären Ausdrücken 133
Die Elemente der Sprache für reguläre Ausdrücke 133
Beispiele 138
7.8 Datum und Uhrzeit 138
7.9 Arrays und assoziative Arrays (Hash-Tabelle) 140
7.10 Operatoren 143
7.11 Überblick über die Kontrollkonstrukte 145
7.12 Schleifen 145
Die For-Schleife 146
Die Do/While/Until-Schleifenfamilie 147
Die ForEach-Schleife 149
7.13 Bedingungen 150
If…Else-Bedingung 151
Switch-Bedingung 152
7.14 Unterroutinen (Prozedur/Funktionen) 153
Prozedur versus Funktion 153
Prozeduren 153
Der Umgang mit Rückgabewerten 154
Art der Rückgabewerte 156
Parameterübergabe 157
7.15 Eingebaute Funktionen 159
7.16 Fehlerbehandlung 159
Fehlerbehandlung mit Trap 159
Konfiguration des Fehlerverhaltens 162
Fehlerbehandlung mit Try-Catch-Finally 164

8 Ausgaben 167

8.1 Ausgabe-Commandlets 167
8.2 Out-GridView 169
8.3 Standardausgabe 170
8.4 Einschränkung der Ausgabe 173
8.5 Seitenweise Ausgabe 173

8.6 Ausgabe einzelner Werte 174
8.7 Details zum Ausgabeoperator 175
8.8 Benutzerdefinierte Tabellenformatierung 179
8.9 Ausgabe von Methodenergebnissen und Unterobjekten in Pipelines 179
8.10 Ausgabe von Methodenergebnissen und Unterobjekten in Zeichenketten 180
8.11 Unterdrückung der Ausgabe 181
8.12 Weitere Ausgabefunktionen 181

9 Eingabefunktionen 183

10 Das PowerShell-Navigationsmodell 185
10.1 Navigation in der Registrierungsdatenbank 185
10.2 Provider und Laufwerke 186
10.3 Navigationsbefehle 188
10.4 Pfadangaben 188
10.5 Beispiel 190
10.6 Eigene Laufwerke definieren 191

11 PowerShell-Werkzeuge 193
11.1 PowerShell-Konsole 193
Funktionsumfang der Konsole 193
Tabulatorvervollständigung 195
Kommandomodus versus Interpretermodus 195
Benutzerkontensteuerung in Windows Vista/Windows 7/ Windows Server 2008 (inkl. R2) 196
11.2 PowerShell Integrated Scripting Environment (ISE) 197
11.3 PowerShellPlus 200
11.4 PoshConsole 204
11.5 PowerGUI 205
11.6 PowerShell Analyzer 205
11.7 PrimalScript 206

11.8 PowerShell Help 209
11.9 PowerShell Help Reader 209
11.10 PowerTab 210
11.11 VS Command Shell 210
11.12 PowerShell Remoting 210
11.13 Vergleich der Skripteditoren 211

Teil 2 PowerShell-Aufbauwissen 213

12 Verwendung von .NET-Klassen 215
12.1 Erzeugen von Instanzen 215
12.2 Parameterbehaftete Konstruktoren 216
12.3 Nutzung von Attributen und Methoden 216
12.4 Statische Mitglieder in .NET-Objekten/ statische .NET-Klassen 217
12.5 Laden von Assemblies 218
12.6 Objektanalyse 219
12.7 Auflistungen (Enumerationen) 219
12.8 Verknüpfen von Aufzählungswerten 221

13 Verwendung von COM-Klassen 223
13.1 Erzeugen von Instanzen 223
13.2 Nutzung von Attributen und Methoden 223
13.3 Holen bestehender Instanzen 224

14 Zugriff auf die Windows Management Instrumentation (WMI) 225
14.1 Einführung in WMI 225
Was ist WMI? 225
WMI-Funktionsumfang 226
WMI-Klassen und WMI-Objekte 228
Schlüsselattribute 229
Systemattribute 230

Datentypen 230
Objektassoziationen 232
WMI-Namensräume (Namespaces) 232
Lokalisierung 233
WMI-Pfade 233
WMI-Schema 235
WMI-Repository 235
WMI-Systemdienst 236
WMI-Provider 236
Managed Object Format (MOF) 246
WMI-Sicherheit 248
WMI Query Language (WQL) 250
Typen von WQL-Anfragen 250
Datenabfragen (Data Queries) 250
Schemaabfragen (Schema Queries) 251
Ereignisabfragen (Event Queries) 251
WMI-Ereignissystem 253
Konsumenten 253
WMI-Werkzeuge 255
WMI-Programmierschnittstellen 258
System.Management 258

14.2 WMI in der Windows PowerShell 259

14.3 Zugriff auf einzelne WMI-Objekte 260

14.4 Der PowerShell-WMI-Objektadapter 261

14.5 Umgang mit Datumsangaben 264

14.6 Zugriff auf WMI-Objektmengen 264

14.7 Filtern und Abfragen 265
Filtern mit Get-WmiObject 265
WQL-Abfragen 265
Beispiel 266

14.8 Zugriff auf Mitglieder von WMI-Klassen 267

14.9 Werte setzen mit Set-WmiInstance 269

14.10 Methodenaufrufe mit Invoke-WmiMethod 269

14.11 Instanzen entfernen 269

14.12 Ermittlung der Mitglieder des WMI-Objekts 270

14.13 Liste aller WMI-Klassen 271
14.14 Instanziieren von WMI-Klassen 271

15 Dynamische Objekte 273
15.1 Erweitern bestehender Objekte 273
15.2 Komplett dynamische Objekte 275

16 Einbinden von C# und VB.NET 277

17 Win32-API-Aufrufe 279

18 Fernausführung (Remoting) 283
18.1 Fernabfrage ohne WS-Management 284
18.2 Anforderungen 285
18.3 Rechte für Fernaufrufe 286
18.4 Einrichten von WinRM 286
18.5 Überblick über die Commandlets 288
18.6 Interaktive Fernverbindungen im Telnet-Stil 288
18.7 Fernausführung von Befehlen 289
18.8 Fernausführung von Skripten 292
18.9 Ausführung auf mehreren Computern 293
18.10 Sitzungen 294
Commandlets zur Sitzungsverwaltung 295
Sitzungen erstellen 296
Schließen von Sitzungen 296
Sitzungskonfigurationen 297
Zugriffsrechte für Fernaufrufe 297
18.11 Zugriff auf entfernte Computer außerhalb der eigenen Domäne 299
Herleitung des Problems 299
Eintrag in die Liste vertrauter Systeme 302
18.12 Verwaltung des WS-Management-Dienstes 303

19 Fehlersuche 305
19.1 Detailinformationen 306
19.2 Einzelschrittmodus 307
19.3 Zeitmessung 308
19.4 Ablaufverfolgung 308

20 Transaktionen 309
20.1 Commandlets für Transaktionen 309
20.2 Start und Ende einer Transaktion 309
20.3 Zurücksetzen der Transaktion 311
20.4 Mehrere Transaktionen 312

21 Hintergrundaufträge („Jobs“) 313
21.1 Voraussetzungen 313
21.2 Architektur 313
21.3 Starten eines Hintergrundauftrags 314
21.4 Hintergrundaufträge abfragen 314
21.5 Warten auf einen Hintergrundauftrag 315
21.6 Abbrechen und Löschen von Aufträgen 315
21.7 Analyse von Fehlermeldungen 316
21.8 Fernausführung von Hintergrundaufträgen 316

22 Ereignissystem 317
22.1 WMI-Ereignisse 317
22.2 WMI-Ereignisabfragen 317
22.3 WMI-Ereignisse mit PowerShell 1.0 319
22.4 Registrieren von WMI-Ereignisquellen 320
22.5 Auslesen der Ereignisliste 321
22.6 Reagieren auf Ereignisse 323
22.7 Registrieren von .NET-Ereignissen 325
22.8 Erzeugen von Ereignissen 326

23 Datenbereiche und Datendateien 327

23.1 Datenbereiche 327

23.2 Datendateien 328

23.3 Mehrsprachigkeit/Lokalisierung 328

24 PowerShell-Snap-Ins 333

24.1 Einbinden von Snap-Ins 333

Registrieren der DLL 333

Hinzufügen des Snap-In zur PowerShell-Konsole 334

Laden des Snap-In 335

Liste der Snap-Ins 336

24.2 Liste der Commandlets 337

24.3 Doppeldeutige Namen 338

25 PowerShell-Module 339

25.1 Überblick über die Commandlets 339

25.2 Modularchitektur 340

25.3 Module installieren 341

25.4 Importieren von Modulen 341

25.5 Entfernen von Modulen 344

26 Ausgewählte PowerShell-Erweiterungen 345

26.1 PowerShell-Module in Windows 7 und Windows Server 2008 R2 345

26.2 BITS-Transfer 347

26.3 Windows Server Backup 348

26.4 PowerShell Diagnostics 348

26.5 Windows PowerShell Community Extensions 349

26.6 PowerShellPack 352

26.7 www.IT-Visions.de PowerShell Extensions 355

26.8 Quest Management Shell for Active Directory 356

26.9 Microsoft Exchange Server 2007 357

26.10 System Center Virtual Machine Manager 2007 358
26.11 PowerShell Management Library for Hyper-V (pshyperv) 359
26.12 Powershell Outlook Account Manager 360
26.13 PowerShell Configurator (PSConfig) 360
26.14 Weitere Erweiterungen 362

27 Tipps und Tricks zur PowerShell 363
27.1 Befehlsgeschichte 363
27.2 System- und Hostinformationen 364
27.3 Alle Anzeigen löschen 365
27.4 Anpassen der Eingabeaufforderung (Prompt) 365
27.5 Profileinstellungen für die PowerShell-Konsole 366
27.6 Einblicke in die Interna der Pipeline-Verarbeitung 372

Teil 3 PowerShell im Praxiseinsatz 375

28 Dateisystem 377
28.1 Laufwerke 378
Auflisten der Laufwerke 378
Laufwerk anlegen 379
Füllstand 380
Laufwerksbezeichnungen 382
Netzlaufwerke 382
28.2 Ordnerinhalte 382
28.3 Dateisystemoperationen 383
28.4 Dateieigenschaften lesen 384
28.5 Datei-Hash 384
28.6 Finden von Duplikaten 385
28.7 Dateieigenschaften verändern 387
Eigenschaften ausführbarer Dateien 388

28.8 Verknüpfungen im Dateisystem 390
Explorer-Verknüpfungen 390
URL-Verknüpfungen 390
Hardlinks 391
Junction Points 392
Symbolische Verknüpfungen ab Windows Vista 393
28.9 Komprimierung 393
28.10 Dateisystemfreigaben 395
Freigaben auflisten 396
Freigaben anlegen 396
28.11 Überwachung des Dateisystems 401

29 Dokumente 403
29.1 Textdateien 403
29.2 Binärdateien 404
29.3 CSV-Dateien 404
29.4 INI-Dateien 406
29.5 XML-Dateien 406
Prüfung von XML-Dokumenten 407
Formatierte Ausgabe 408
XPath-Anweisungen 409
XML-Dateien aus Pipeline exportieren 411
XML-Dateien transformieren 413
29.6 HTML-Dateien 415

30 Datenbanken 417
30.1 ADO.NET-Grundlagen 417
Providerarchitektur 417
Datenprovider von Microsoft 418
Datenprovider von anderen Herstellern 418
Ermittlung der installierten Datenprovider 419
Liste der verfügbaren SQL Server 419
Datenwege 420
Datareader versus Dataset 421
30.2 Beispieldatenbank 423

30.3 Datenzugriff mit den Bordmitteln der PowerShell 424
Datenbankverbindungen (Connection) 424
Providerunabhängiger Zugriff 425
Befehle ausführen 426
Datenzugriff mit dem Datareader 428
Datenzugriff mit dem Dataset 429
Objektmodell 430
Datenadapter 431
Providerspezifisches Beispiel 432
30.4 Datenzugriff mit den PowerShell-Erweiterungen 435

31 Registrierungsdatenbank (Registry) 439
31.1 Schlüssel auslesen 439
31.2 Schlüssel anlegen und löschen 439
31.3 Laufwerke definieren 440
31.4 Werte anlegen und löschen 440
31.5 Werte auslesen 442
31.6 Praxisbeispiel 442

32 Computerverwaltung 445
32.1 Computerinformationen 445
32.2 Computername und Domäne 447
32.3 Herunterfahren und Neustarten 448
32.4 Wiederherstellungspunkte verwalten 448

33 Hardwareverwaltung 449
33.1 Hardwarebausteine 449
Prozessor 450
Anzeige 450
Batterien und USV 451
33.2 Druckerverwaltung 451

34 Softwareverwaltung 453
34.1 Softwareinventarisierung 453
34.2 Installation von Anwendungen 456

34.3 Deinstallation von Anwendungen 457

34.4 Praxisbeispiel: Installationstest 457

34.5 Versionsnummer ermitteln 458

34.6 Servermanager 459

Get-WindowsFeature 460

Ausschnitt aus Feature.format.ps1xml 463

Add-WindowsFeature 463

Remove-WindowsFeature 465

Umgang mit Neustarts 465

Ausgabe des Befehls Get-WindowsFeature auf einem Beispielsystem 465

34.7 Softwareeinschränkungen mit dem PowerShell-Modul „AppLocker“ 470

Auflisten von Regeln 472

Prüfen der Regelwirkung 473

Erstellen neuer Regeln 473

35 Prozessverwaltung 477

35.1 Prozesse auflisten 477

35.2 Prozesse starten 478

35.3 Prozesse beenden 479

35.4 Warten auf das Beenden einer Anwendung 480

36 Systemdienste 481

36.1 Dienste auflisten 481

36.2 Dienstzustand ändern 483

36.3 Diensteigenschaften ändern 484

37 Netzwerk 485

37.1 Netzwerkkonfiguration 485

37.2 Erreichbarkeit prüfen (Ping) 486

Ping über WMI 486

Ping über Test-Connection 487

37.3 Namensauflösung 488

37.4 E-Mails senden **488**
E-Mail senden mit System.Net.Mail 488
E-Mail senden mit Send-SmtpMail 489
E-Mail senden mit Send-MailMessage 489
37.5 Abruf von Daten von einem HTTP-Server **490**
37.6 Aufrufe von Webdiensten **491**

38 Ereignisprotokolle **493**
38.1 Protokolleinträge auslesen **493**
38.2 Ereignisprotokolle erzeugen **494**
38.3 Protokolleinträge erzeugen **495**
38.4 Protokollgröße festlegen **495**
38.5 Protokolleinträge löschen **495**

39 Leistungsdaten **497**
39.1 Zugriff auf Leistungsindikatoren über WMI **497**
39.2 Get-Counter **497**
Sicherheitseinstellungen 499
39.3 Grundlagen **499**
Programmierschnittstellen 503
Kontenname und SID 504
39.4 Zugriffsrechtelisten auslesen **504**
39.5 Einzelne Rechteeinträge auslesen **506**
39.6 Besitzer auslesen **507**
39.7 Benutzer und SID **508**
Umwandeln zwischen Benutzername und SID 508
Well-Known Security Identifier verwenden 508
SDDL verwenden 510
39.8 Hinzufügen eines Rechteeintrags zu einer Zugriffsrechteliste **511**
39.9 Entfernen eines Rechteeintrags aus einer Zugriffsrechteliste **513**
39.10 Zugriffsrechteliste übertragen **515**
39.11 Zugriffsrechteliste über SDDL setzen **516**

40 Active Directory 517

40.1 Benutzer- und Gruppenverwaltung mit WMI 518

40.2 Einführung in System.DirectoryServices 519

Architektur 519
Weiterreichen an ADSI 521
Objektmodell 521
Allgemeine Klassen 521
Klasse „DirectoryEntry“ 521
Klasse „DirectoryEntries“ 523
Klassen für die Ausführung von Suchanfragen 523
Vergleich zwischen System.DirectoryServices und ADSI 524
Unzulänglichkeiten der Implementierung 524
Objektidentifikation in Verzeichnisdiensten (Verzeichnisdienstpfade) 526
Objektidentifikation im Active Directory 527
Überblick über die Programmiermechanismen 528
Bindung an einen Verzeichniseintrag 528
Impersonifizierung 529
Prüfung auf Existenz eines Verzeichniseintrags 530
Verzeichnisattribute lesen 530
ADSI Property Cache 531
Verzeichnisattribute schreiben 531

40.3 Basiseigenschaften 532

Zugriff auf Containerobjekte 532
Verzeichnisobjekt anlegen 533
Verzeichnisobjekt löschen 533

40.4 Benutzer- und Gruppenverwaltung im Active Directory 534

Die Active-Directory-Verzeichnisklasse „user“ 534
Benutzerkonto anlegen 537
Kennwort des Benutzers setzen 538
Benutzerauthentifizierung 538
Benutzerkonto löschen 539
Benutzerkonto umbenennen 539
Benutzerkonto verschieben 540
Gruppenverwaltung 540
Anlegen und Befüllen einer Gruppe 541
Testen der Gruppenmitgliedschaft 542

40.5 Verwaltung der Organisationseinheiten 542
40.6 Suche im Active Directory 543
LDAP-Suchanfragen 543
Programmierschnittstellen für die Suche 544
Ausführung einer Abfrage in der PowerShell 544
Beispiel 545
Suche nach einem Benutzer mit seinem Anmeldenamen 546
Tipps und Tricks zur Suche 547
40.7 Navigation im Active Directory mit den PowerShell Extensions 551
40.8 Verwendung der Active-Directory-Erweiterungen von www.IT-Visions.de 551
40.9 PowerShell-Modul „Active Directory“ (ADPowerShell) 554
Architektur und Installation 554
Aktivieren des Active-Directory-Moduls 556
Active-Directory-Navigationsprovider 557
Objektmodell 559
Überblick über die Commandlets 561
Allgemeine Verwaltungs-Commandlets 562
Filtern und Suchen 564
Verwaltung von Organisationseinheiten 565
Verwaltung von Benutzerkonten 566
Verwaltung von Benutzergruppen 568
40.10 Informationen über die Active-Directory-Struktur 569
Informationen über die Domäne durch die .NET-Klassenbibliothek 569
Informationen über die Domäne durch das Modul ADPowerShell 571

41 Gruppenrichtlinien 573
41.1 Verwaltung der Gruppenrichtlinien 573
41.2 Verknüpfung der Gruppenrichtlinien 575
41.3 Berichte 576
41.4 Gruppenrichtlinienvererbung 577
41.5 Weitere Möglichkeiten 578

42 Internet Information Server (IIS) 579

42.1 Überblick 579
42.2 Navigationsprovider 581
42.3 Anlegen von Websites 583
42.4 Massenanlegen von Websites 584
42.5 Ändern von Eigenschaften von Websites 586
42.6 Anwendungspool anlegen 586
42.7 Virtuelle Verzeichnisse und IIS-Anwendungen 587
42.8 Website-Zustand ändern 588
42.9 Anwendungspools starten und stoppen 588
42.10 Löschen von Websites 589

43 Microsoft Exchange Server 591

43.1 Daten abrufen 591
43.2 Postfächer verwalten 591
43.3 Öffentliche Ordner verwalten 592

44 Optimierungen und Problemlösungen 593

44.1 PowerShell-Modul „TroubleShootingPack“ 593
44.2 PowerShell-Modul „Best Practices“ 596

45 Grafische Benutzeroberflächen 599

45.1 Eingabemasken 599
45.2 Universelle Objektdarstellung 601
45.3 Zwischenablage 602
45.4 WPF PowerShell Kit (WPK) 602
Hello World mit WPF 603
Unzureichende Dokumentation 603
Schaltflächen und Ereignisse 606
Panel-Elemente zum Anordnen von Steuerelementen 608
Weitere Möglichkeiten 610

Teil 4 **Profiwissen – Erweitern der PowerShell** 613

46 Entwicklung von Commandlets in der PowerShell-Sprache 615

46.1 Aufbau eines skriptbasierten Commandlets 615
46.2 Parameterfestlegung 618
46.3 Auszeichnung der Parameterdefinitionen 624
46.4 Dokumentation 627

47 Entwicklung eigener Commandlets mit C# 631

47.1 Technische Voraussetzungen 631
47.2 Grundkonzept der .NET-basierten Commandlets 633
Commandlet-Klassen 633
Methoden 633
Ausgabe des Commandlets 634
Snap-In-Klasse 634
47.3 Schrittweise Erstellung eines minimalen Commandlets 635
Anlegen eines Visual-Studio-Projekts 635
Anlegen des Commandlets 636
Implementierung des Commandlets 637
Kompilieren des Commandlets 639
Installation des Commandlets 639
Verwendung des Commandlets 642
Weiterentwicklung der Commandlet-Erweiterungen 642
47.4 Erstellung eines Commandlets mit einem Rückgabeobjekt 643
47.5 Erstellung eines Commandlets mit mehreren Rückgabeobjekten 645
47.6 Erstellen eines Commandlets mit Parametern 649
47.7 Verarbeiten von Pipeline-Eingaben 651
47.8 Verkettung von Commandlets 653
Kopplung auf Basis elementarer Datentypen 654
Kopplung auf Basis von typisierten Objekten 655
Generische Kopplung 658
47.9 Fehlersuche in Commandlets 658
47.10 Statusinformationen 661

47.11 Unterstützung für Sicherheitsabfragen 665
47.12 Festlegung der Hilfeinformationen 668
47.13 Erstellung von Commandlets für den Zugriff auf eine Geschäftsanwendung 672
47.14 Konventionen für Commandlets 673
47.15 Weitere Möglichkeiten 675

48 Hosting der Windows PowerShell 677
48.1 Voraussetzungen für das Hosting 678
48.2 Hosting mit PSHost 678
48.3 Vereinfachtes Hosting ab PowerShell 2.0 682

49 PowerShell-Module erstellen 685
49.1 Erstellen eines Skriptmoduls 685
49.2 Erstellen eines Moduls mit Binärdateien 686
49.3 Erstellen eines Moduls mit Manifest 687

A Anhang 697
A.1 Crashkurs „Objektorientierung“ 697
Was ist ein Objekt? 697
Was ist eine Klasse? 698
Kapselung 700
Objektbeziehungen 701
Vererbung 702
Weitere Konzepte 704

B Crashkurs „.NET Framework“ 705
B.1 Was ist das .NET Framework? 706
Eigenschaften des .NET Frameworks 708
.NET-Klassen 709

C Commandlet-Referenz 715

Stichwortverzeichnis 799

Vorwort zur zweiten Auflage

Liebe Leserin, lieber Leser,

willkommen zur zweiten Auflage meines PowerShell-Buches! Dieses Buch behandelt die Windows PowerShell Version 2.0 von Microsoft sowie ergänzende Werkzeuge von Microsoft und Drittanbietern (z.B. PowerShell Community Extensions, PowerShellPack, PowerShellPlus). Gegenüber der ersten Auflage zur PowerShell 1.0 wurde das Buch erheblich erweitert. Die Erweiterung des Buchs betrifft dabei nicht nur die neuen Funktionen der PowerShell 2.0. Vielmehr wurden in dem Buch auch Grundlagen für Einsteiger ergänzt, die in der vorherigen Auflage vorausgesetzt wurden. Damit eignet sich dieses Buch sowohl für PowerShell-Einsteiger als auch fortgeschrittene Benutzer.

Dieses Vorwort dient primär dazu, Ihnen eine Entscheidungsgrundlage zu bieten, ob dieses Buch für Sie geeignet ist.

Wer bin ich?

Mein Name ist Holger Schwichtenberg, ich bin derzeit 37 Jahre alt und habe im Fachgebiet Wirtschaftsinformatik promoviert. Ich lebe (in Essen, der europäischen Kulturhauptstadt 2010) davon, dass ich mit meiner Firma *www.IT-Visions.de* Unternehmen bei der Entwicklung von .NET-Anwendungen unterstütze. Es ist mein Hobby und „Nebenberuf", IT-Fachbücher zu schreiben. Dieses Buch ist, unter Mitzählung aller nennenswerten Neuauflagen, das 46. Buch, das ich allein oder mit Co-Autoren geschrieben habe. Meine weiteren Hobbys sind Radfahren, Fotografie und Reisen.

Natürlich verstehe ich das Bücherschreiben auch ein wenig als Werbung für die Arbeit meines Unternehmens und hoffe, dass der ein oder andere von Ihnen uns beauftragen wird, Sie durch Beratung, Schulung und Auftragsentwicklung zu unterstützen.

Wer sind Sie?

Damit Sie den optimalen Nutzen aus diesem Buch ziehen können, möchte ich – so genau es mir möglich ist – beschreiben, an wen sich dieses Buch richtet. Hierzu habe ich einen Fragebogen ausgearbeitet, mit dem Sie schnell erkennen können, ob das Buch für Sie geeignet ist.

Sind Sie Systemadministrator in einem **homogenen** Windows-Netzwerk?	O Ja O Nein
Laufen alle für Sie relevanten Computer mit Windows XP oder einem neueren Betriebssystem (Server 2003, Vista, Server 2008, Windows 7)?	O Ja O Nein
Sie besitzen zumindest rudimentäre Grundkenntnisse im Bereich des objektorientierten Programmierens?	O Ja O Nein
Sie haben bereits den Windows Script Host (WSH) verwendet?	O Ja O Nein
Wünschen Sie einen kompakten Überblick über die Architektur, Konzepte und Anwendungsfälle der PowerShell?	O Ja O Nein
Sie können auf Schritt-für-Schritt-Anleitungen verzichten?	O Ja O Nein
Sie können auf formale Syntaxbeschreibungen verzichten und lernen lieber am aussagekräftigen Beispiel?	O Ja O Nein
Sie erwarten nicht, dass in diesem Buch alle Möglichkeiten der PowerShell detailliert beschrieben werden?	O Ja O Nein
Sind Sie, nachdem Sie ein Grundverständnis durch dieses Buch gewonnen haben, bereit, Detailfragen in der Dokumentation der PowerShell, von .NET und WMI nachzuschlagen?	O Ja O Nein
Sie haben nicht als primäres Ziel, Ihre alten WSH-Skripte in die PowerShell zu konvertieren?	O Ja O Nein

Wenn Sie alle obigen Fragen mit „Ja“ beantwortet haben, ist das Buch richtig für Sie. In anderen Fällen sollten Sie sich erst mit einführender Literatur beschäftigen.

Wie können Sie mit mir in Kontakt treten?

Unter *http://www.IT-Visions.de/leser* biete ich ein Webportal für Leser meiner Bücher an. In diesem Portal können Sie Feedback zu diesem Buch geben (Bewertung abgeben und Fehler melden), technische Fragen in einem Webforum stellen und ergänzende Beispiele herunterladen. Alle registrierten Leser erhalten auch Einladungen zu kostenlosen Community-Veranstaltungen sowie Vergünstigungen bei unseren öffentlichen Seminaren zu .NET und zur PowerShell. Bei der Registrierung müssen Sie das Kennwort *VOYAGER* angeben.

Unter meiner E-Mail-Adresse *hs@IT-Visions.de* stehe ich für Anfragen bezüglich Schulung, Beratung und Entwicklungstätigkeiten zur Verfügung.

Viel Spaß mit der PowerShell wünscht Ihnen

Dr. Holger Schwichtenberg

Danksagungen

Folgenden Personen möchte ich meinen Dank für Ihre Mitwirkung an diesem Buch aussprechen:

- Frau Sylvia Hasselbach, die dieses Buchprojekt bei Addison-Wesley koordiniert und vermarktet,
- Herrn Rolf Masuch, der den Inhalt und alle Listings genau geprüft hat und das Buch mit seinen Vorschlägen verbessert hat
- Frau Petra Kienle, die meine Tippfehler gefunden und sprachliche Ungenauigkeiten eliminiert hat,
- meinem Kollegen und Freund Tobias Weltner für sein sehr hilfreiches Werkzeug „PowerShellPlus“,
- allen Lesern, die mit ihrem Feedback zu der ersten Auflage dazu beigetragen haben, dass dieses Buch noch besser wurde.
- meiner Frau und meinem Sohn dafür, dass Sie mir das Umfeld geben, um neben meinem Hauptberuf an Büchern wie diesem zu arbeiten.

Über den Autor Dr. Holger Schwichtenberg



Ausbildung:

- Studium Diplom-Wirtschaftsinformatik an der Universität Essen
- Promotion an der Universität Essen im Gebiet komponentenbasierter Softwareentwicklung
- Tätig in der Softwareentwicklung seit 1996

Aktuelle Tätigkeit:

- Leitung der Firma *www.IT-Visions.de*
- Softwareentwicklung und Softwarearchitektur im Kundenauftrag
- Beratung und Schulung von Softwareentwicklern
- Individueller Support für Softwareentwicklungsprojekte
- Gutachter in den Wettbewerbsverfahren der EU gegen Microsoft

Kernkompetenzen:

- Einführung von .NET Framework und Visual Studio/Migration auf .NET
- .NET-Strategie/Auswahl von .NET-Technologien
- Webanwendungsentwicklung mit IIS, ASP.NET und AJAX

- Enterprise .NET
- Verteilte Systeme mit .NET insbesondere Windows Communication Foundation
- C#, Visual Basic (VB/VB.NET/VBS/VBA)
- Component Object Model (COM), Windows Scripting (WSH), Windows PowerShell (WPS)
- Relationale Datenbanken, XML und Object Relational Mapping (ORM)
- Active Directory-Programmierung und Windows Management Instrumentation (WMI)

Veröffentlichungen und Vorträge:

- Über 40 Fachbücher bei Addison-Wesley, Microsoft Press und dem Carl Hanser-Verlag
- Mehr als 500 Beiträge in Fachzeitschriften
- Ständiger Mitarbeiter der Zeitschriften iX und dotnetpro sowie bei heise.de
- Sprecher auf nationalen und internationalen Fachkonferenzen (z.B. TechEd, OOP, Microsoft Launch Event, MSDN Summit, BASTA, Advanced Developers Conference, Microsoft IT Forum, Wirtschaftsinformatik, VS One, u.a.)
- Sprecher in MSDN Webcasts

Ehrenamtliche Community-Tätigkeiten:

- Vorstandsmitglied bei codezone.de
- Sprecher für die International .NET Association (INETA)
- Betrieb der Community-Websites *www.dotnetframework.de* und *www.windows-scripting.de*

Zertifikate und Auszeichnungen von Microsoft:

- Most Valuable Professional (MVP)
- Microsoft Certified Solution Developer (MCSD)
- .NET Code Wise Community-Experte
- Codezone Premier Site Member

Firmen-Website:

- *http://www.IT-Visions.de*

Weblog:

- *http://www.dotnet-doktor.de* (bei Heise.de)

Kontakt:

- hs@IT-Visions.de
- 0201 7490-700 (Büro *www.IT-Visions.de*)



Teil 1

PowerShell-Basiswissen

Dieser Buchteil informiert über die Basiskonzepte der PowerShell, insbesondere Commandlets, Pipelines, Navigation und Skripte. Außerdem werden am Ende dieses Teils Werkzeuge vorgestellt.

1 Erste Schritte mit der Windows PowerShell

Das DOS-ähnliche Kommandozeilenfenster hat viele Windows-Versionen in beinahe unveränderter Form überlebt. Mit der Windows PowerShell (WPS) besitzt Microsoft nun endlich einen Nachfolger, der es mit den Unix-Shells aufnehmen kann und diese in Hinblick auf Eleganz und Robustheit in einigen Punkten auch überbieten kann. Die PowerShell ist eine Adaption des Konzepts von Unix-Shells auf Windows unter Verwendung des .NET Frameworks und mit Anbindung an die Windows Management Instrumentation (WMI).

1.1 Was ist die Windows PowerShell?

In einem Satz: Die Windows PowerShell (WPS) ist eine neue, .NET-basierte Umgebung für interaktive Systemadministration und Scripting auf der Windows-Plattform.

Die Kernfunktionen der PowerShell sind:

Kernfunktionen

- Zahlreiche eingebaute Befehle, die „Commandlets“ genannt werden
- Zugang zu allen Systemobjekten, die durch COM-Bibliotheken, das .NET Framework und die Windows Management Instrumentation (WMI) bereitgestellt werden
- Robuster Datenaustausch zwischen Commandlets durch Pipelines basierend auf typisierten Objekten
- Ein einheitliches Navigationsparadigma für verschiedene Speicher (z.B. Dateisystem, Registrierungsdatenbank, Zertifikatsspeicher, Active Directory und Umgebungsvariablen)
- Eine einfach zu erlernende, aber mächtige Skriptsprache mit wahlweise schwacher oder starker Typisierung
- Ein Sicherheitsmodell, das die Ausführung unerwünschter Skripte unterbindet
- Integrierte Funktionen für Ablaufverfolgung und Debugging
- Die PowerShell kann um eigene Befehle erweitert werden.
- Die PowerShell kann in eigene Anwendungen integriert werden (Hosting).

1.2 Geschichte

Von Active Scripting zu PowerShell

Das Active Scripting ist einigen Administratoren zu komplex, weil es viel Wissen über objektorientiertes Programmieren und das Component Object Model (COM) voraussetzt. Die vielen Ausnahmen und Ungereimtheiten im Active Scripting erschweren das Erlernen von Windows Script Host (WSH) und den zugehörigen Komponentenbibliotheken.

Schon im Zuge der Entwicklung des Windows Server 2003 gab Microsoft zu, dass man Unix-Administratoren zum Interview über ihr tägliches Handwerkszeug gebeten hatte. Das kurzfristige Ergebnis war eine große Menge zusätzlicher Kommandozeilenwerkzeuge. Langfristig setzt Microsoft jedoch auf eine Ablösung des DOS-ähnlichen Konsolenfensters durch eine neue Scripting-Umgebung.

Mit dem Erscheinen des .NET Frameworks im Jahre 2002 wurde lange über einen WSH.NET spekuliert. Microsoft stellte jedoch die Neuentwicklung des WSH für das .NET Framework ein, als abzusehen war, dass die Verwendung von .NET-basierten Programmiersprachen wie C# und Visual Basic .NET dem Administrator nur noch mehr Kenntnisse über objektorientierte Softwareentwicklung abverlangen würde.

Microsoft beobachtete in der Unix-Welt eine hohe Zufriedenheit mit den dortigen Kommandozeilen-Shells und entschloss sich daher, das Konzept der Unix-Shells, insbesondere das Pipelining, mit dem .NET Framework zusammenzubringen und daraus eine .NET-basierte Windows Shell zu entwickeln. Diese ist so einfach wie eine Unix-Shell, aber kann so mächtig wie das .NET Framework sein.

In einer ersten Beta-Version wurde die neue Shell schon unter dem Codenamen „Monad“ auf der Professional Developer Conference (PDC) im Oktober 2003 in Los Angeles vorgestellt. Nach den Zwischenstufen „Microsoft Shell (MSH)“ und „Microsoft Command Shell“ trägt die neue Skriptumgebung seit Mai 2006 den Namen „Windows PowerShell“.

Erscheinungstermine

Die fertige Version der Windows PowerShell 1.0 ist am 14.11.2006 auf der TechEd Europe 2006 dem Markt übergeben worden. Die PowerShell 2.0 ist am 6.8.2009 als Bestandteil von Windows 7 und am 15.8.2009 als Bestandteil von Windows Server 2008 R2 erschienen. Zum Redaktionsschluss für dieses Buch gab es erst eine Vorabversion der PowerShell 2.0 für andere Betriebssysteme.

Softwareinventarisierung

Falls Sie eine Motivation brauchen, sich mit der PowerShell zu beschäftigen, wird dieses Kapitel Ihnen diese liefern. Es stellt die Lösung für eine typische Scripting-Aufgabe sowohl in dem „alten“ Windows Script Host (WSH) als auch in der „neuen“ Windows PowerShell vor.

Zur Motivation, sich mit der Windows PowerShell zu beschäftigen, soll folgendes Beispiel aus der Praxis dienen. Es soll ein Inventarisierungsskript für Software erstellt werden, das die installierten MSI-Pakete mit

Hilfe der Windows Management Instrumentation (WMI) von mehreren Computern ausliest und die Ergebnisse in einer CSV-Datei (*softwareinventar.csv*) zusammenfasst. Die Namen (oder IP-Adressen) der abzufragenden Computer sollen in einer Textdatei (*computernamen.txt*) stehen.

Die Lösung mit dem WSH benötigt 90 Codezeilen (inklusive Kommentare und Parametrisierungen). In der Windows PowerShell lässt sich das Gleiche in nur 13 Zeilen ausdrücken. Wenn man auf die Kommentare und die Parametrisierung verzichtet, dann reicht sogar genau eine Zeile.

Listing 1.1
Softwareinventarisierung – Lösung 1 mit dem WSH [Einsatzgebiete/Software/Software_Inverntory.vbs]

```
' ----------------------------------------
' Skriptname: Software_inventar.vbs
' Autor: Dr. Holger Schwichtenberg 2004-2007
' ----------------------------------------
' Dieses Skript erstellt eine Liste
' der installierten Software
' Version 1.02 (28.1.2007)
' ----------------------------------------
Option Explicit

' --- Vorgabewerte
Const Trennzeichen = ";" ' Trennzeichen für Spalten in der
Ausgabedatei
Const Eingabedateiname = "computernamen.txt"
Const Ausgabedateiname = "softwareinventar.csv"
Const Bedingung = "SELECT * FROM Win32_Product where not Vendor
like '%Microsoft%'"

Dim objFSO' Dateisystem-Objekt
Dim objTX    ' Textdatei-Objekt für die Liste der zu durchsuchenden
computer
Dim i  ' Zähler für Computer
Dim computer ' Name des aktuellen computers
Dim Eingabedatei' Name und Pfad der Eingabedatei
Dim Ausgabedatei' Name und Pfad der Ausgabedatei

' --- Startmeldung
WScript.Echo "Softwareinventar.vbs"
WScript.Echo "(C) Dr. Holger Schwichtenberg, http://www.Windows-
Scripting.de"

' --- Global benötigtes Objekt
Set objFSO = CreateObject("Scripting.FileSystemObject")

' --- Ermittlung der Pfade
Eingabedatei = GetCurrentPfad & "\" & Eingabedateiname
Ausgabedatei = GetCurrentPfad & "\" & Ausgabedateiname

' --- Auslesen der computerliste
```

```
Set objTX = objFSO.OpenTextFile(Eingabedatei)

' --- Meldungen
WScript.Echo "Eingabedatei: " & Eingabedatei
WScript.Echo "Ausgabedatei: " & Ausgabedatei

' --- Überschriften einfügen
Ausgabe _
"computer" & Trennzeichen & _
"Name" & Trennzeichen & _
    "Beschreibung" & Trennzeichen & _
    "Identifikationsnummer" & Trennzeichen & _
    "Installationsdatum" & Trennzeichen & _
    "Installationsverzeichnis" & Trennzeichen & _
    "Zustand der Installation" & Trennzeichen & _
    "Paketzwischenspeicher" & Trennzeichen & _
    "SKU Nummer" & Trennzeichen & _
    "Hersteller" & Trennzeichen & _
    "Version"

' --- Schleife über alle computer
Do While Not objTX.AtEndOfStream
    computer = objTX.ReadLine
    i = i + 1
    WScript.Echo "=== Computer #" & i & ": " & computer

GetInventar computer

Loop

' --- Eingabedatei schließen
objTX.Close
' --- Abschlußmeldung
WScript.echo "Softwareinventarisierung beendet!"

' === Softwareliste für einen computer erstellen
Sub GetInventar(computer)

Dim objProduktMenge
Dim objProdukt
Dim objWMIDienst

' --- Zugriff auf WMI
Set objWMIDienst = GetObject("winmgmts:" &_
    "{impersonationLevel=impersonate}!\\" & computer &_
    "\root\cimv2")
' --- Liste anfordern
Set objProduktMenge = objWMIDienst.ExecQuery _
    (Bedingung)
' --- Liste ausgeben
```

```
WScript.echo "Auf " & computer & " sind " & _
objProduktMenge.Count & " Produkte installiert."
For Each objProdukt In objProduktMenge
    Ausgabe _
    computer & Trennzeichen & _
    objProdukt.Name & Trennzeichen & _
    objProdukt.Description & Trennzeichen & _
    objProdukt.IdentifyingNumber & Trennzeichen & _
    objProdukt.InstallDate & Trennzeichen & _
    objProdukt.InstallLocation & Trennzeichen & _
    objProdukt.InstallState & Trennzeichen & _
    objProdukt.PackageCache & Trennzeichen & _
    objProdukt.SKUNumber & Trennzeichen & _
    objProdukt.Vendor & Trennzeichen & _
    objProdukt.Version
WScript.Echo     objProdukt.Name
Next
End Sub

' === Ausgabe
Sub Ausgabe(s)
Dim objTextFile
' Ausgabedatei öffnen
Set objTextFile = objFSO.OpenTextFile(Ausgabedatei, 8, True)
objTextFile.WriteLine s
objTextFile.Close
'WScript.Echo s
End Sub

' === Pfad ermitteln. in dem das Skript liegt
Function GetCurrentPfad
GetCurrentPfad = objFSO.GetFile
(WScript.ScriptFullName).ParentFolder
End Function
```

Listing 1.2 *Softwareinventarisierung – Lösung 2 als Windows PowerShell-Skript [Einsatzgebiete/Software/SoftwareInventory_WMI_PipelineAsScript.ps1]*

```
# Settings
$InputFileName = "computers.txt"
$OutputFileName = "softwareinventory.csv"
$Query = "SELECT * FROM Win32_Product where not Vendor like
'%Microsoft%'"

# Read computer list
$Computers = Get-Content $InputFileName

# Loop over all computers and read WMI information
$Software = $Computers | foreach { Get-wmiobject -query $Query -
computername $_ }
# Export to CSV
$Software | select Name, Description, IdentifyingNumber,
InstallDate, InstallLocation, InstallState, SKUNumber, Vendor,
Version | export-csv $OutputFileName -notypeinformation
```

Listing 1.3 Softwareinventarisierung – Lösung 3 als Windows PowerShell-Pipeline-Befehl [Einsatzgebiete/Software/SoftwareInventory_WMI_Pipeline.ps1]

```
Get-Content "computers.txt" | foreach {Get-wmiobject -computername
$_ -query "SELECT * FROM Win32_Product where not Vendor like
'%Microsoft%'" } | export-csv "Softwareinventory.csv" -
notypeinformation
```

1.3 Betriebssysteme mit PowerShell

Die folgende Tabelle zeigt, in welchen Betriebssystemen die PowerShell mitgeliefert wird.

Tabelle 1.1 Verfügbarkeit der Windows PowerShell auf verschiedenen Windows-Betriebssystemen

Betriebssystem	Mitgelieferte PowerShell	Nachträglich installierbare PowerShell
Windows 2000, Windows 9x, Windows NT 4.0	PowerShell nicht enthalten	Nachträgliche Installation nicht von Microsoft unterstützt
Windows XP	PowerShell nicht enthalten	PowerShell 1.0 und PowerShell 2.0 (*1)
Windows Server 2003	PowerShell nicht enthalten	PowerShell 1.0 und PowerShell 2.0 (*1)
Windows Server 2003 R2	PowerShell nicht enthalten	PowerShell 1.0 und PowerShell 2.0 (*1)
Windows Vista	PowerShell nicht enthalten	PowerShell 1.0 und PowerShell 2.0 (*2)
Windows Vista ab Service Pack 1	PowerShell 1.0 enthalten	PowerShell 2.0 (*2)
Windows Server 2008	PowerShell 1.0 enthalten als optionales Features	PowerShell 2.0 (*2)
Windows Server 2008 ab Service Pack 1	PowerShell 1.0 enthalten	PowerShell 2.0 (*2)
Windows 7	PowerShell 2.0 enthalten	
Windows Server 2008 R2	PowerShell 2.0 enthalten	
Windows Server 2008 Core	PowerShell nicht enthalten	Nicht verfügbar
Windows Server 2008 R2 Core	PowerShell 2.0 enthalten als optionales Features	

1.4 PowerShell herunterladen und installieren

Für andere Betriebssysteme muss die PowerShell separat von der Microsoft-Homepage bezogen werden. Die PowerShell ist im Paket Windows Management Framework (WMF, 19,58 MB) enthalten, siehe [MS03].

Installationsvoraussetzungen

Die nachträgliche Installation der PowerShell 2.0 ist möglich auf folgenden Betriebssystemen:

- Windows Server 2008 mit Servicepack 1
- Windows Server 2008 mit Servicepack 2
- Windows Server 2003 mit Servicepack 2
- Windows Vista mit Servicepack 2
- Windows Vista mit Servicepack 1
- Windows XP mit Servicepack 3
- Windows Embedded-POSReady 2009
- Windows Embedded for Point of Service 1.1

Notwendige Software

Die PowerShell 2.0 benötigt das Microsoft .NET Framework 2.0 oder höher.

Einige Funktionen in der PowerShell 2.0 stellen allerdings zusätzliche Anforderungen:

- Windows Remote Management (WinRM) 2.0 wird für Fernaufrufe benötigt (enthalten im Paket „Windows Management Framework“).
- `Get-WinEvent` braucht .NET Framework 3.5.
- `Out-Gridview` setzt .NET Framework 3.5 mit Service Pack 1 voraus.
- Windows PowerShell Integrated Scripting Environment (ISE) braucht .NET Framework 3.5 mit Service Pack 1.

Folgende .NET-Versionen sind schon auf den Betriebssystemen vorhanden:

- Windows 7: .NET 3.5.1
- Windows Server 2008 R2: .NET 3.5.1
- Windows Server 2008: .NET 3.5
- Windows Vista: .NET 3.0
- Windows Server 2003 R2: .NET 2.0

Unter Windows XP und Windows Server müssen Sie jedoch zuerst .NET Framework 2.0, 3.0 oder 3.5 installieren (falls diese Versionen nicht ohnehin schon durch eine andere Anwendung installiert worden sind).



Auf 64-Bit-Systemen gibt es zusätzlich die 32-Bit-Version unter C:\windows\SysWOW64\WindowsPowerShell\v1.0. Der Start dieser Version kann erforderlich sein, um Komponenten zu verwenden, für die es keine 64-Bit-Version gibt (z.B. Treiber für Microsoft Access). Es handelt sich dabei nicht um einen Tippfehler: Die 64-Bit-Version ist in einem Verzeichnis, das „32" im Namen trägt, und die 32-Bit-Version in einem Verzeichnis mit „64" im Namen!

Installationsordner

Die PowerShell 2.0 installiert sich in folgendes Verzeichnis: *%systemroot%\system32\WindowsPowerShell\V1.0* (für 32-Bit-Systeme). Auf 64-Bit-Systemen gibt es die PowerShell 2.0 zweimal, einmal als 64-Bit-Version und einmal als 32-Bit-Version. Letztere findet man unter *%systemroot%\Syswow64\WindowsPowerShell\V1.0*. Die 32-Bit-Version braucht man, wenn man eine Bibliothek nutzen will, für die es keine 64-Bit-Version gibt, z.B. den Zugriff auf Microsoft-Access-Datenbanken.



Sie sollten auch die erweiterte PowerShell-Dokumentation auf Ihrem Rechner haben, siehe */install/PowerShell/PowerShell Dokumentation* auf der Buch-CD bzw. [MS01] und [MS02].



Durch die Installation der PowerShell wird in Windows auch ein neues Ereignisprotokoll „PowerShell" angelegt, in dem die PowerShell wichtige Zustandsänderungen der PowerShell protokolliert.

Die PowerShell 1.0 verstand sich als ein Windows-Update und erfordert einen Neustart des Systems nach der Installation. Deshalb ist die PowerShell unter SOFTWARE der Systemsteuerung nicht als „Programm", sondern als „Update" namens „Hotfix for Windows (KB ...)" aufgeführt. Unter Windows 7 und Windows Server 2008 R2 lässt sich die im Standard installierte PowerShell nicht über die Systemsteuerung deinstallieren.

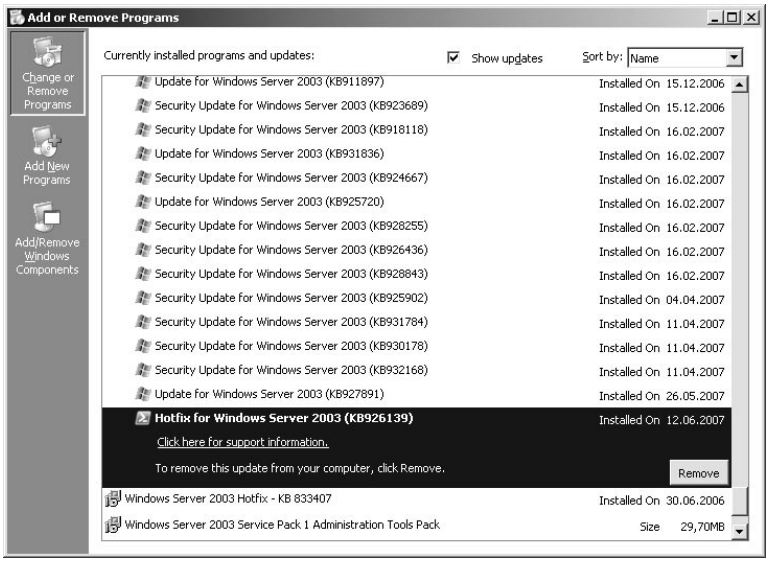


Abbildung 1.1
Die Option zur Deinstallation der Windows PowerShell ist schwer zu finden (hier: Windows Server 2003).

1.5 Die Windows PowerShell testen

Dieses Kapitel stellt einige Befehle vor, mit denen Sie die PowerShell-Funktionalität ausprobieren können. Die PowerShell verfügt über zwei Modi (interaktiver Modus und Skriptmodus), die hier getrennt behandelt werden.

PowerShell im interaktiven Modus

Der erste Test verwendet die PowerShell im interaktiven Modus.

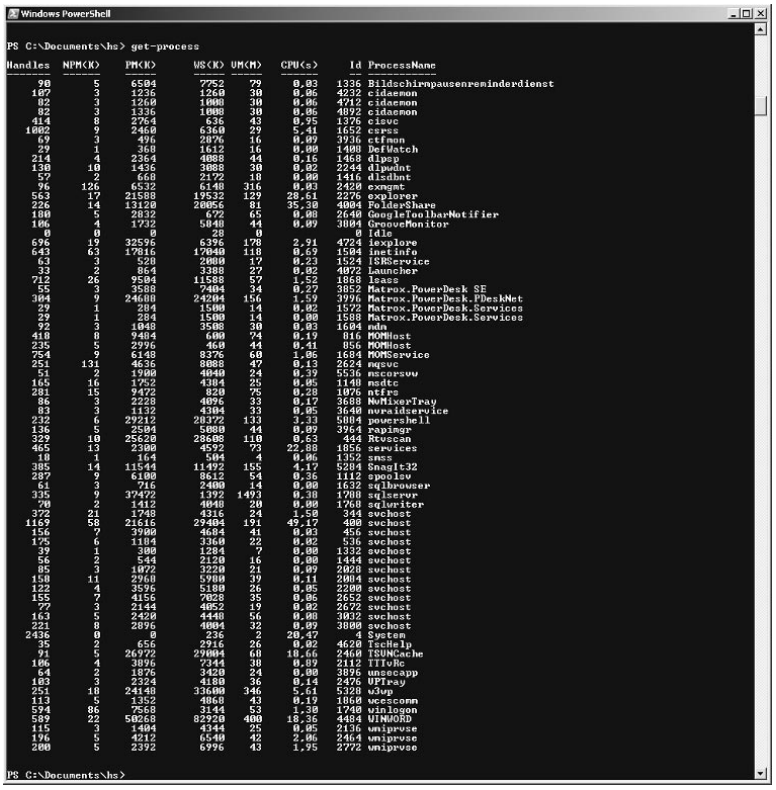
Starten Sie bitte die Windows PowerShell. Es erscheint ein leeres PowerShell-Konsolenfenster. Auf den ersten Blick ist kein großer Unterschied zur herkömmlichen Windows-Konsole zu erkennen. Allerdings steckt in der PowerShell mehr Kraft im wahrsten Sinne des Wortes.



Abbildung 1.2
Leeres PowerShell 2.0-Konsolenfenster

Geben Sie an der Eingabeaufforderung „Get-Process“ ein und drücken Sie dann die ↵-Taste. Es erscheint eine Liste aller Prozesse, die auf dem lokalen Computer laufen. Dies war Ihre erste Verwendung eines einfachen PowerShell-Commandlets.

Abbildung 1.3
Die Liste der Prozesse ist das Ergebnis nach Ausführung des Commandlets „Get-Process“.

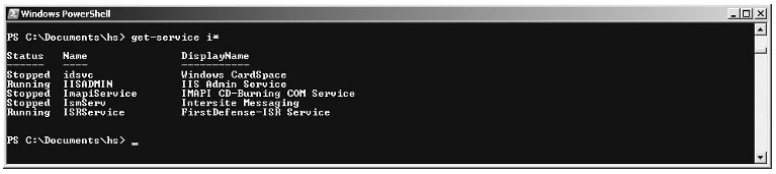




Beachten Sie bitte, dass die Groß-/Kleinschreibung keine Rolle spielt, da PowerShell keine Unterschiede zwischen groß- und kleingeschriebenen Commandlet-Namen macht.

Geben Sie an der Eingabeaufforderung „Get-service i*“ ein. Jetzt erscheint eine Liste aller installierten Dienste auf Ihrem Computer, deren Namen mit dem Buchstaben „i“ beginnen. Hier haben Sie ein Commandlet mit Parametern verwendet.

Abbildung 1.4
Eine gefilterte Liste der Windows-Dienste



TAB-Taste

Geben Sie „Get-“ ein und drücken Sie dann mehrmals die ⇆-Taste. Die PowerShell zeigt nacheinander alle Commandlets an, die mit dem Verb „get“ beginnen. Microsoft bezeichnet diese Funktionalität als „Tabulatorvervollständigung“. Halten Sie bei „Get-Eventlog“ an. Wenn Sie ↵ drücken, fordert die PowerShell einen Parameter namens „Log-

Name“ an. Bei „LogName“ handelt es sich um einen erforderlichen Parameter (Pflichtparameter). Nachdem Sie „Application“ eingetippt und die ↵-Taste gedrückt haben, erscheint eine lange Liste der aktuellen Einträge in Ihrem Anwendungsereignisprotokoll.



Abbildung 1.5
Die PowerShell fragt einen erforderlichen Parameter ab.

Pipelining

Der letzte Test bezieht sich auf die Pipeline-Funktionalität der PowerShell. Auch geht es darum, die Listeneinträge aus dem Windows-Ereignisprotokoll aufzulisten, doch dieses Mal sind nur bestimmte Einträge interessant. Die Aufgabe besteht darin, die letzten zehn Ereignisse abzurufen, die sich auf das Drucken beziehen. Geben Sie den folgenden Befehl ein, der aus drei Commandlets besteht, die über Pipes miteinander verbunden sind:

```
Get-EventLog system | Where-Object { $_.source -eq "print" } |
Select-Object -first 10
```

Die PowerShell scheint einige Sekunden zu hängen, nachdem die ersten zehn Einträge ausgegeben wurden. Dieses Verhalten ist korrekt, da das erste Commandlet (`Get-EventLog`) alle Einträge empfängt. Dieses Filtern geschieht durch aufeinanderfolgende Commandlets (`Where-Object` und `Select-Object`). Leider besitzt `Get-EventLog` keinen integrierten Filtermechanismus.

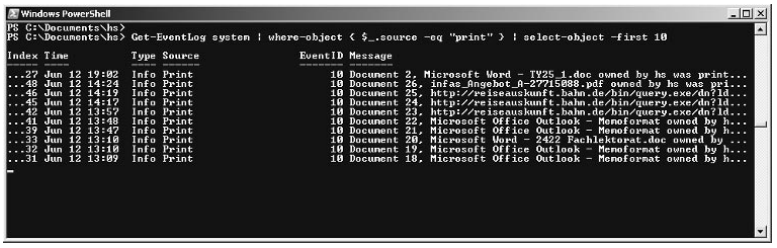


Abbildung 1.6
Die Einträge des Ereignisprotokolls filtern

PowerShell im Skriptmodus

Skripte

Die zweite Serie von PowerShell-Beispielen beinhaltet ein PowerShell-Skript. Bei einem PowerShell-Skript handelt es sich um eine Textdatei, die Commandlets und/oder Elemente der PowerShell-Skriptsprache (PSL) umfasst. Das Skript erstellt ein neues Benutzerkonto auf Ihrem lokalen Computer.

Öffnen Sie den Windows-Editor „Notepad“ (oder einen anderen Texteditor) und geben Sie die folgenden Skriptcodezeilen ein, die aus Kommentaren, Variablendeklarationen, COM-Bibliotheksaufrufen und Shell-Ausgabe bestehen:

Listing 1.4 *Ein Benutzerkonto erstellen [Einsatzgebiete/Benutzer/LocalUser_Create.ps1]*

```
### PowerShell-Script
### Create local User Acount

# Variables
$Name = "Dr. Holger Schwichtenberg"
$Accountname = "HolgerSchwichtenberg"
$Description = "Author of this book / Website:
www.powershell24.com"
$Password = "secret+123"
$Computer = "localhost"

"Creating User on Computer $Computer"

# Access to Container using the COM library "Active Directory
Service Interface"
$Container = [ADSI] "WinNT://$Computer"

# Create User
$objUser = $Container.Create("user", $Accountname)
$objUser.Put("Fullname", $Name)
$objUser.Put("Description", $Description)
# Set Password
$objUser.SetPassword($Password)
# Save Changes
$objUser.SetInfo()

"User created: $Name"
```

Speichern Sie die Textdatei unter dem Namen „createuser.ps1“ im Ordner *c:\temp*. Beachten Sie, dass die Dateierweiterung „.ps1“ lauten muss.

Starten Sie die Windows PowerShell. Versuchen Sie, das Skript zu starten. Geben Sie dazu „c:\temp\createuser.ps1“ ein. (Für die Ordner- und Dateinamen können Sie die Tabulatorvervollständigung verwenden!) Der Versuch scheitert eventuell, da die Skriptausführung – standardmäßig – in der PowerShell nicht zulässig ist. Dies ist kein Fehler, sondern eine Sicherheitsfunktionalität. (Denken Sie an den „Love Letter“-Wurm für den Windows Script Host!)

Eventuell benötigt Ihr Konto aber auch administrative Rechte, um diese Aktion durchzuführen. Starten Sie dazu die PowerShell als Administrator. Unter Windows Vista und Windows 7 wählen Sie dazu aus dem Startmenü die PowerShell mit der rechten Maustaste aus und klicken auf „Als Administrator ausführen“.



Abbildung 1.7 *Die Skriptausführung ist standardmäßig verboten.*

Für den ersten Test wird die Sicherheit ein wenig abgeschwächt. Aber wirklich nur ein wenig. Man lässt die Ausführung von Skripten zu, die sich auf dem lokalen System befinden. Allerdings braucht man für Skripte, die von Netzwerkressourcen (das Internet eingeschlossen) kommen, eine digitale Signatur von einem vertrauenswürdigen Skriptautor. Später in diesem Buch lernen Sie, wie Sie PowerShell-Skripte digital signieren. Außerdem erfahren Sie, wie Sie Ihr System auf Skripte beschränken, die Sie oder Ihre Kollegen signiert haben.

Um die Ausführung des Skripts zuzulassen, geben Sie folgenden Befehl ein:

```
Set-ExecutionPolicy remotesigned
```

Überprüfen Sie die vorgenommenen Änderungen mit dem Commandlet `Get-ExecutionPolicy`.

Starten Sie nun das Skript erneut. Jetzt sollte die Nachricht erscheinen, dass das Benutzerkonto erstellt worden ist.



Abbildung 1.8
Führen Sie Ihr erstes Skript aus, um ein Benutzerkonto zu erstellen.

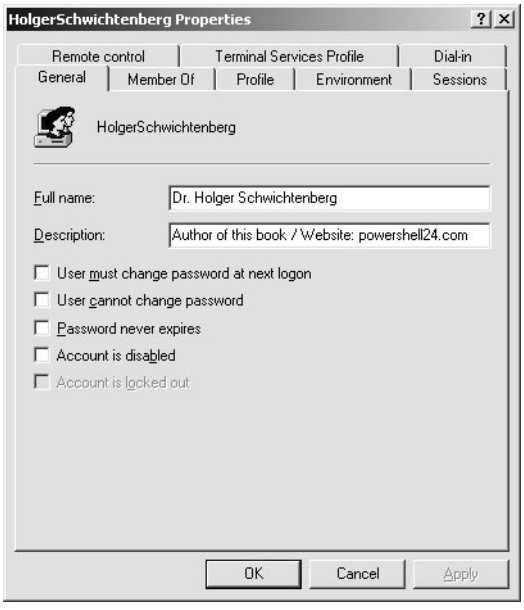


Abbildung 1.9
Das neu erstellte Benutzerkonto

1.6 PowerShell Community Extensions herunterladen und installieren

PSCX Die Windows PowerShell umfasste in der Version 1.0 nur 129 Commandlets. In PowerShell 2.0 sind es mehr, aber immer noch nur 236. Was heißt hier „nur", werden Sie fragen. Die wichtigsten Commandlets sind die, die mit den Verben „get" und „set" beginnen. Und die Anzahl dieser Commandlets ist sehr klein im Vergleich zu der riesigen Anzahl von Objekten, die das Windows-Betriebssystem zu bieten hat. Alle anderen Commandlets beziehen sich – mehr oder weniger – auf die PowerShell-Infrastruktur, z.B. Filtern, Formatieren und Exportieren.

Bei „PowerShell Community Extensions" (kurz PSCX) handelt es sich um ein OpenSource-Projekt, das zusätzliche Funktionalität mit Commandlets realisiert, wie zum Beispiel `Get-DhcpServer`, `Get-DomainController`, `Get-MountPoint`, `Get-TerminalSession`, `Ping-Host`, `Write-GZip` und viele weitere. Das Projekt steht unter Führung von Microsoft, aber jeder .NET-Softwareentwickler ist eingeladen, daran mitzuwirken. In regelmäßigen Abständen werden neue Versionen veröffentlicht. Mittlerweile ist die Version 1.1.1 als aktuelles stabiles Release erschienen.



Die PowerShell Community Extensions finden Sie auf der CD-ROM zum Buch unter */install/PowerShell\Commandlets\PowerShell Community Extensions* oder unter [CODEPLEX01].

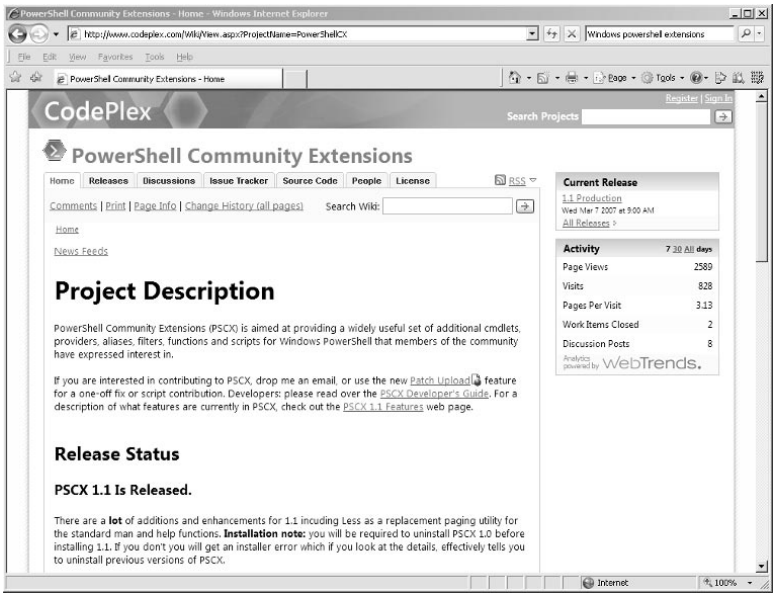


Abbildung 1.10 *Website für die PowerShell Community Extensions*

Profile.ps1

Die PowerShell Community Extensions werden als Setup-Routine bereitgestellt, die Sie installieren sollten, nachdem Sie die Windows PowerShell erfolgreich installiert haben.

Teile der zusätzlichen Funktionalität von PSCX werden in die PowerShell mit Hilfe eines Profilskripts eingebunden (*profile.ps1*). Dieses Profilskript wird in Ihrem Ordner „Eigene Dateien/Windows PowerShell" kopiert, wenn Sie dies beim PSCX-Setup so festlegen. Als Einsteiger sollten Sie diese Option wählen.

Die Installation der PowerShell Community Extensions (PSCX) ändert die PowerShell-Konsole nur ein wenig. Anstelle des aktuellen Pfads enthält die Eingabeaufforderung nun einen Zähler. Der Pfad wird jetzt im Fenstertitel angezeigt.

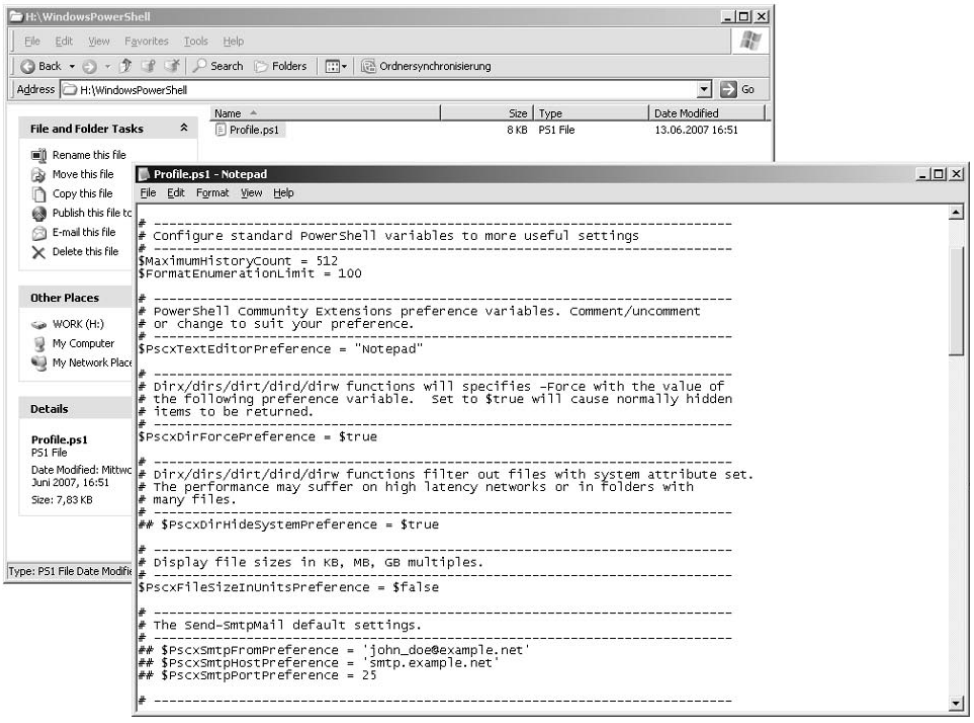


***Abbildung 1.11**: Das PSCX-Profilskript, das während des PSCX-Setups erstellt wird*

Get-Domain-Controller

Starten Sie die PowerShell und geben Sie „Get-DomainController" ein (wenn Ihr Computer Mitglied eines Active Directory ist) oder testen Sie die PSCX mit dem Befehl „Ping-Host", der auf jedem Computer im Netzwerk funktioniert.

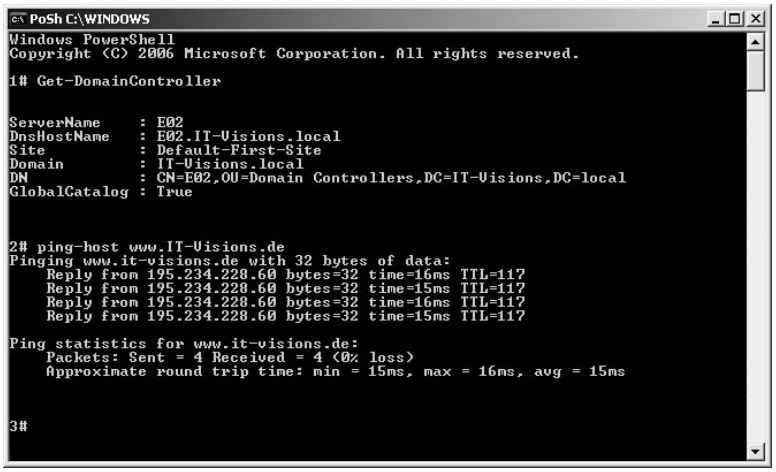


Abbildung 1.12 Die Befehle Get-Domain Controller und Ping-Host testen

1.7 Die ISE verwenden

PowerShell ISE

Integrated Scripting Environment (ISE) ist der Name des (einfachen) Skripteditors, den Microsoft mit der PowerShell 2.0 mitliefert. Die ISE startet man, indem man in der PowerShell „ise" ausführt.

Die ISE verfügt – entsprechend der PowerShell-Konsole – über zwei Modi: einen interaktiven Modus und einen Skriptmodus. Nach dem Start zeigt die ISE drei Bereiche: ganz oben einen Skripteditor, ganz unten einen interaktiven Eingabebereich und in der Mitte einen Ausgabebereich sowohl für Skripte als auch interaktive Ausgaben.

Geben Sie unten in der ISE ein:

```
Get-Process
```

Nachdem Sie mindestens einen Buchstaben eingegeben haben, können Sie die Eingabe mit der Tabulatortaste vervollständigen.

Script Editor

Um die ISE im Skriptmodus zu verwenden, erstellen Sie eine neue Skriptdatei (Menü „File/New") oder öffnen Sie eine vorhandene *.ps1*-Datei (Menü „File/Open"). Öffnen Sie als Beispiel die Skriptdatei *CreateUser.ps1*, die Sie zuvor erstellt haben. Es sind Zeilennummern zu sehen. Die verschiedenen Bestandteile des Skripts sind in unterschiedlichen Farben dargestellt. Auch hier funktioniert die Eingabeunterstützung mit der Tabulatortaste.

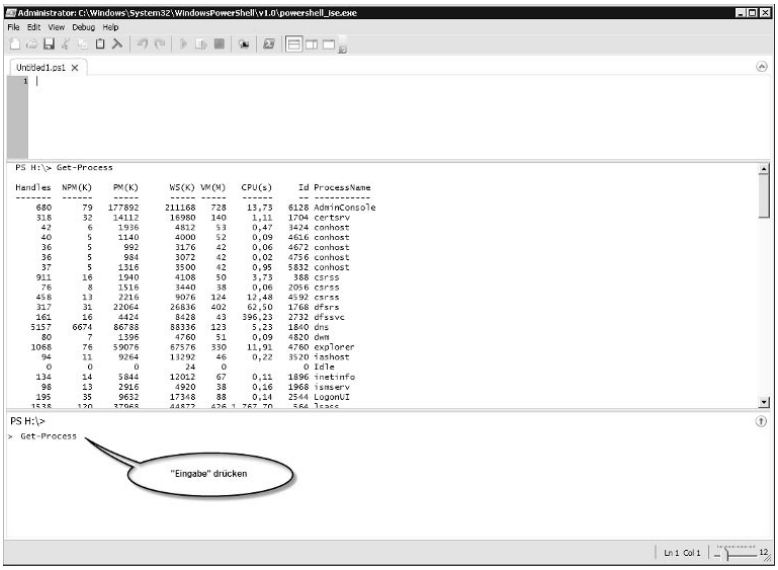


Abbildung 1.13
Zustand der der ISE nach Eingabe von „Get-Process“ im interaktiven Bereich

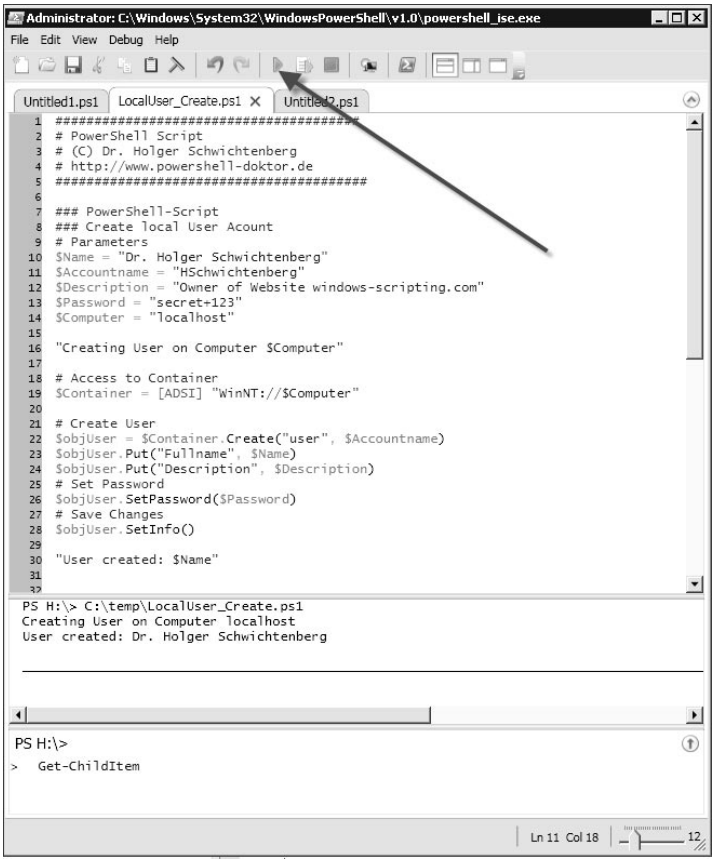


Abbildung 1.14
Die ISE im Skriptmodus

Um das Skript auszuführen, klicken Sie auf das Start-Symbol in der Symbolleiste (siehe die Screenshots) oder drücken Sie F5. Auch hier wird das Ergebnis im mittleren Bereich angezeigt. Stellen Sie sicher, dass das Benutzerkonto noch nicht existiert, bevor Sie das Skript ausführen.

Debugging

Ein interessantes Feature ist das Debugging. Setzen Sie den Cursor auf eine beliebige Zeile in Ihrem Skript und tippen Sie dann auf F9 (oder wählen Sie „Toogle Breakpoint" im Kontextmenü oder im Menü „Debug"). Daraufhin erscheint die Zeile in Rot – ein sogenannter „Haltepunkt". Starten Sie das Skript. Die ISE stoppt in der Zeile mit dem Haltepunkt. Im interaktiven Bereich können Sie nun den aktuellen Zustand der Variablen abfragen, indem Sie dort z.B. eingeben

```
$Computer
```

oder

```
$Container
```

Man kann auch Werte interaktiv ändern. Um das Skript fortzusetzen, drücken Sie wieder F5. Über das Menü „Debug" sind weitere Steuerbefehle möglich.



Abbildung 1.15
Skript-Debugging mit der ISE

2 Architektur der Windows PowerShell

Die Windows PowerShell ist eine Symbiose aus:

- dem DOS-Kommandozeilenfenster,
- den bekannten Skript- und Shell-Sprachen wie Perl, Ruby, ksh und bash,
- dem .NET Framework und
- der Windows Management Instrumentation (WMI).

Basis ist .NET

Die PowerShell ist implementiert auf dem .NET Framework. Sie ist jedoch kein .NET Runtime Host mit der Möglichkeit, Befehle der Common Intermediate Language (CIL) auf der Common Language Runtime (CLR) auszuführen.

Die PowerShell verwendet ein völlig anderes Host-Konzept mit Commandlets, Objekt-Pipelines und einer neuen Sprache, die von Microsoft als PowerShell Language (PSL) bezeichnet wird. Sie ist Perl, Ruby, C# und einigen Unix-Shell-Sprachen sehr ähnlich, aber mit keiner Unix-Shell kompatibel. Nutzer der WMI Command Shell (*wmic.exe*), die mit Windows XP eingeführt wurde, werden sich in der PowerShell schnell zurechtfinden.

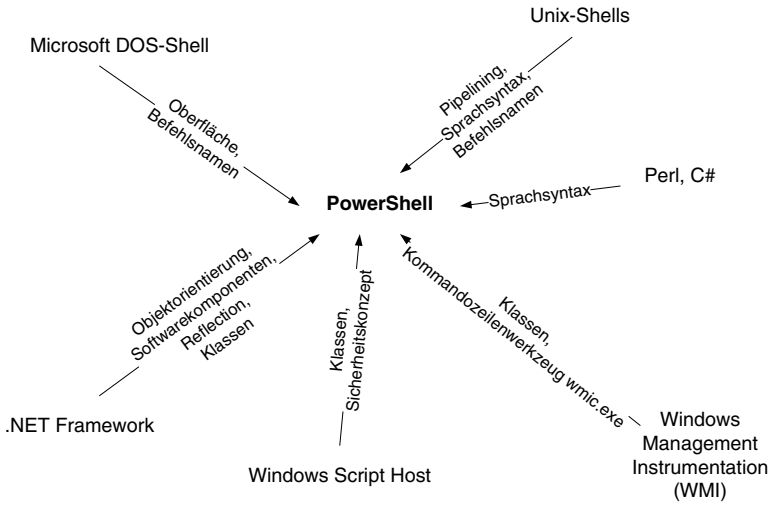


Abbildung 2.1 *Einflussfaktoren auf die Architektur und die Umsetzung der PowerShell*



Die PowerShell ist angetreten, vom Administrator weniger Kenntnisse in Objektorientierung und über Softwarekomponenten zu verlangen, als dies der Vorgänger Windows Script Host (WSH) tat. Tatsächlich kann man in der PowerShell viel erreichen, ohne sich mit dem zu Grunde liegenden .NET Framework zu beschäftigen. Dennoch: Wer alle Möglichkeiten der PowerShell nutzen will, braucht dann aber doch etwas Verständnis für objektorientiertes Programmieren und Erfahrung mit dem .NET Framework.

Wenn Sie sich hier noch nicht auskennen, lesen Sie bitte zuerst im Anhang dieses Buchs die Anhang A „Crashkurs „Objektorientierung““ und Anhang B „Crashkurs „.NET Framework“ “.

Anbindung an Klassenbibliotheken

Nutzung von Klassen

Die Version 1.0 der PowerShell enthielt sehr viele Commandlets für die Pipelining-Infrastruktur, aber nur sehr wenige Befehle, die tatsächlich Bausteine des Betriebssystems in die Pipeline werfen. Prozesse, Systemdienste, Dateien, Zertifikate und Registrierungsdatenbankeinträge sind die magere Ausbeute beim ersten Blick in die Commandlet-Liste. Zwei Commandlets eröffnen der PowerShell aber neue Dimensionen: `New-Object` (für .NET- und COM-Objekte) und `Get-Wmiobject` (für WMI-Objekte). In Version 2.0 gibt es – zumindest in Verbindung mit Windows 7 und Windows Server 2008 Release 2 – mehr PowerShell-Befehle, die tatsächlich auf das Betriebssystem zugreifen.



Die Option, nicht nur alle WMI-Klassen, sondern auch alle .NET-Klassen direkt benutzen zu können, ist Segen und Fluch zugleich. Ein Segen, weil dem Skriptentwickler damit mehr Möglichkeiten als jemals zuvor zur Verfügung stehen. Ein Fluch, weil nur der Skriptentwickler die PowerShell-Entwicklung richtig beherrschen kann, der auch das .NET Framework kennt. Um die Ausmaße von .NET zu beschreiben, sei die Menge der Klassen genannt. In .NET 2.0 waren es 5684, in .NET 3.5 sind es rund 10.000.

PowerShell versus WSH

Administratoren fragen sich oft, wie sich die PowerShell im Vergleich zum Windows Script Host (WSH) positioniert, womit man neue Scripting-Projekte beginnen sollte und ob der WSH bald aus Windows verschwinden wird. Die folgende Tabelle trägt Fakten zusammen und bewertet auch die beiden Scripting-Plattformen.

	Windows Script Host (WSH)	Windows PowerShell (WPS)
Erstmals erschienen	1998	2006
Aktueller Versionsstand	5.8	2.0
Basisplattform	COM	.NET
Derzeitiger Funktionsumfang	Sehr umfangreich	Direkter Funktionsumfang in Form von Commandlets noch spärlich; über Interoperabilität zu COM und .NET sehr umfangreiche Funktionen; Zahl der Commandlets wächst aber rasch. Erst in einigen Jahren ist eine breite Unterstützung durch die Produktteams von Microsoft zu erwarten.
Weiterentwicklung der Laufzeitumgebung	Nein, nicht mehr geplant	Ja, Version 3.0 in Arbeit
Weiterentwicklung der Bibliotheken	Ja, umfangreich (COM wird auch in Zukunft noch eine wichtige Rolle spielen)	Ja, zahlreiche Commandlet-Erweiterungen erscheinen mit kommenden Microsoft-Produkten.
Weiterentwicklung der Werkzeuge	Nein	Ja
Plattformen	Alle Windows-Betriebssysteme ab Windows 95/ NT 4.0	Nur XP, 2003, Vista, Windows Server 2008, Windows 7
Basissyntax	Mächtig	Sehr mächtig
Direkte Scripting-Möglichkeiten	Alle COM-Komponenten mit IDispatch-Schnittstelle	Alle .NET-Komponenten, alle COM-Komponenten
Scripting-Möglichkeiten über Wrapper	Alle Betriebssystemfunktionen	Alle Betriebssystemfunktionen
Werkzeuge von Microsoft	Scriptgeneratoren, Debugger	Bislang keine (*)
Werkzeuge von Drittanbietern	Editoren, Debugger, Scriptgeneratoren	Editoren, Debugger, Scriptgeneratoren
Erfahrung der Nutzer	Einige Administratoren kennen sich gut aus, es gibt aber immer noch viele, die jetzt erst mit WSH beginnen, was sich daran zeigt, dass der WSH immer noch ständig unter den 20 meistgeladenen Downloads bei MSDN ist.	Bisher kaum Erfahrung im Markt vorhanden.
Einarbeitungsaufwand	Hoch	Mittel bis hoch (je nach Art der PowerShell-Nutzung)

***Tabelle 2.1**: Vergleich WSH und PowerShell*

	Windows Script Host (WSH)	Windows PowerShell (WPS)
Migrationsoptionen	Gering	Keine
Informationsverfügbarkeit	Hoch zur Laufzeitumgebung, gering zu spezielleren Themen	Gering, aber wachsend
Zu erwartender Zeitraum mit breitem Einsatz des Produkts	ca. 2015	Mindestens 2030 (vgl. DOS-Shell)

***Tabelle 2.1**: Vergleich WSH und PowerShell (Forts.)*



Hinweise zur Umstellung von WSH/VBScript auf die PowerShell finden Sie unter [TNET03].

3 Einzelbefehle der PowerShell

Die PowerShell kennt folgende Arten von Einzelbefehlen:

- Commandlets
- Aliase
- Ausdrücke
- Externe Befehle
- Dateinamen

3.1 Commandlets

Commandlet

Ein „normaler" PowerShell-Befehl heißt *Commandlet* (kurz: *Cmdlet*) oder *Funktion* (*Function*). Hier in diesem Kapitel geht es zunächst nur um Commandlets. Eine Funktion ist eine Möglichkeit, in der PowerShell selbst wieder einen Befehl zu erstellen, der funktioniert wie ein Commandlet. Da die Unterscheidung zwischen Commandlets und Funktionen aus Nutzersicht zum Teil akademischer Art ist, erfolgt hier zunächst keine Differenzierung.

Ein Commandlet besteht typischerweise aus drei Teilen:

- einem Verb,
- einem Substantiv und
- einer (optionalen) Parameterliste.

Verb und Substantiv werden durch einen Bindestrich „–" voneinander getrennt, die optionalen Parameter durch Leerzeichen. Daraus ergibt sich der folgende Aufbau:

```
Verb-Substantiv [-Parameterliste]
```

Die Groß- und Kleinschreibung ist bei den Commandlet-Namen nicht relevant.

Ein einfaches Beispiel ohne Parameter lautet:

```
Get-Process
```

Dieser Befehl holt eine Liste aller Prozesse.



Die Tabulatorvervollständigung in der PowerShell-Konsole funktioniert bei Commandlets, wenn man das Verb und den Strich bereits eingegeben hat, z.B. `Export-` ↹. Auch Platzhalter kann man dabei verwenden. Die Eingabe `Get-?e*` ↹ liefert `Get-Help` ↹ `Get-Member` ↹ `Get-Service`.

Commandlet-Parameter

Parameter

Durch Angabe eines Parameters werden nur diejenigen Prozesse angezeigt, deren Name auf das angegebene Muster zutrifft:

```
Get-Process  i*
```

Ein weiteres Beispiel für einen Befehl mit Parameter ist:

```
Get-ChildItem c:\daten
```

`Get-ChildItem` listet alle Unterobjekte des angegebenen Objekts (*c:\daten*) auf, also alle Dateien und Ordner unterhalb dieses Dateiordners.

Parameter werden als Zeichenkette aufgefasst – auch wenn sie nicht explizit in Anführungszeichen stehen. Die Anführungszeichen sind optional. Man muss Anführungszeichen nur verwenden, wenn Leerzeichen vorkommen, denn das Leerzeichen dient als Trennzeichen zwischen Parametern:

```
Get-ChildItem "C:\Program Files"
```

Alle Commandlets haben zahlreiche Parameter, die durch Namen voneinander unterschieden werden. Ohne die Verwendung von Parameternamen werden vordefinierte Standardattribute belegt, d.h., die Reihenfolge ist entscheidend.

```
Get-ChildItem C:\temp *.doc
```

bedeutet das Gleiche wie:

```
Get-ChildItem -Path C:\temp -Filter *.doc
```

Wenn ein Commandlet mehrere Parameter besitzt, ist die Reihenfolge der Parameter entscheidend oder der Nutzer muss die Namen der Parameter mit angeben. Alle folgenden Befehle sind gleichbedeutend:

```
Get-ChildItem C:\temp *.doc
Get-ChildItem -Path C:\temp -Filter *.doc
Get-ChildItem -Filter *.doc -Path C:\temp
```

Bei der Angabe von Parameternamen kann man die Reihenfolge der Parameter ändern:

```
Get-ChildItem -Filter *.doc -Path C:\temp
```

Hingegen ist Folgendes falsch, weil die Parameter nicht benannt sind und die Reihenfolge falsch ist:

```
Get-ChildItem *.doc C:\temp
```

Schalter-Parameter

Schalter-Parameter (engl. Switch) sind Parameter, die keinen Wert haben. Durch die Verwendung des Parameternamens wird die Funktion aktiviert, z.B. das rekursive Durchlaufen durch einen Dateisystembaum mit `-recurse`:

```
Get-ChildItem h:\demo\powershell -recurse
```

Berechnungen in Parametern

Parameter können berechnet, d.h. aus Teilzeichenketten zusammengesetzt sein, die mit einem Pluszeichen verbunden werden. (Dies macht insbesondere Sinn in Zusammenhang mit Variablen, die aber erst später in diesem Buch eingeführt werden.)

Der folgende Ausdruck führt jedoch nicht zum gewünschten Ergebnis, da auch hier das Trennzeichen vor und nach dem + ein Parametertrenner ist.

```
Get-ChildItem "c:\" + "Windows" *.dll -Recurse
```

Auch ohne die beiden Leerzeichen vor und nach dem + geht es nicht. In diesem Fall muss man durch eine runde Klammer dafür sorgen, dass die Berechnung erst ausgeführt wird:

```
Get-ChildItem ("c:\" + "Windows") *.dll -Recurse
```

Es folgt dazu noch ein Beispiel, bei dem Zahlen berechnet werden. Der folgende Befehl liefert den Prozess mit der ID 2900:

```
Get-Process -id (2800+100)
```

Weitere Beispiele

```
Get-Service -exclude "[k-z]*"
```

zeigt nur diejenigen Systemdienste an, deren Name nicht mit den Buchstaben „k“ bis „z“ beginnt.

Auch mehrere Parameter können der Einschränkung dienen. Der folgende Befehl liefert nur die Benutzereinträge aus einem bestimmten Active-Directory-Pfad. (Das Beispiel setzt die Installation der PSCX voraus.)

```
Get-ADObject -dis "LDAP://E02/ou=Geschäftsführung,
OU=www.IT-Visions.de,dc=IT-Visions,dc=local" -class user
```



Tabulatorvervollständigung klappt auch bei Parametern. Versuchen Sie einmal folgende Eingabe an der PowerShell-Konsole: `Get-ChildItem -` ⇆

Platzhalter

An vielen Stellen sind Platzhalter bei den Parameterwerten erlaubt.

Eine Liste aller Prozesse, die mit einem „i“ anfangen, erhält man so:

```
Get-Process i*
```

Weitere Aspekte zu Commandlets

Beachten Sie, dass bei den Commandlets das Substantiv im Singular steht, auch wenn eine Menge von Objekten abgerufen wird. Das Ergebnis muss nicht immer eine Objektmenge sein. Beispielsweise liefert

```
Get-Location
```

nur ein Objekt mit dem aktuellen Pfad.

Mit

```
Set-Location c:\windows
```

wechselt man den aktuellen Pfad. Diese Operation liefert gar kein Ergebnis.



Die Groß- und Kleinschreibung der Commandlet-Namen und der Parameternamen ist irrelevant.

Verben

Gemäß der PowerShell-Konventionen soll es nur eine begrenzte Menge wiederkehrender Verben geben: `Get`, `Set`, `Add`, `New`, `Remove`, `Clear`, `Push`, `Pop`, `Write`, `Export`, `Select`, `Sort`, `Update`, `Start`, `Stop`, `Invoke` usw. Außer diesen Basisoperationen gibt es auch Ausgabekommandos wie `Out` und `Format`. Auch Bedingungen werden durch diese Syntax abgebildet (`Where-Object`).

Prozessmodell

Die PowerShell erzeugt beim Start einen Prozess. In diesem Prozess laufen alle Commandlets. Dies ist ein Unterschied zum DOS-ähnlichen Windows-Kommandozeilenfenster, bei dem die ausführbaren Dateien (*.exe*) in eigenen Prozessen laufen.

3.2 Aliase

Namensersetzungen

Durch sogenannte Aliase kann die Eingabe von Commandlets verkürzt werden. So ist `ps` als Alias für `Get-Process` oder `help` für `Get-Help` vordefiniert. Statt `Get-Process i*` kann also auch geschrieben werden: `ps i*`.

Aliase auflisten

Durch `Get-Alias` (oder den entsprechenden Alias `aliases`) erhält man eine Liste aller vordefinierten Abkürzungen in Form von Instanzen der Klasse `System.Management. Automation.AliasInfo`.

Durch Angabe eines Namens bei `Get-Alias` erhält man die Bedeutung eines Alias:

```
Get-Alias pgs
```

Möchte man zu einem Commandlet alle Aliase wissen, muss man allerdings schreiben:

```
Get-Alias | Where-Object { $_.definition -eq "Get-Process " }
```

Dies erfordert schon den Einsatz einer Pipeline, die erst im nächsten Kapitel besprochen wird.

Alias	Commandlet
%	ForEach-Object
?	Where-Object
Ac	Add-Content
Asnp	Add-PSSnapIn
Cat	Get-Content
Cd	Set-Location
Chdir	Set-Location
Clc	Clear-Content
Clear	Clear-Host
Clhy	Clear-History
Cli	Clear-Item
Clp	Clear-ItemProperty
Cls	Clear-Host
Clv	Clear-Variable
Compare	Compare-Object
Copy	Copy-Item
Cp	Copy-Item
Cpi	Copy-Item
Cpp	Copy-ItemProperty
Cvpa	Convert-Path
Dbp	Disable-PSBreakpoint
Del	Remove-Item
Diff	Compare-Object
Dir	Get-ChildItem
Ebp	Enable-PSBreakpoint
Echo	Write-Output
Epal	Export-Alias
epcsv	Export-Csv
epsn	Export-PSSession
erase	Remove-Item
etsn	Enter-PSSession

***Tabelle 3.1**: Vordefinierte Aliase in der PowerShell 2.0*

Alias	Commandlet
exsn	Exit-PSSession
fc	Format-Custom
fl	Format-List
foreach	ForEach-Object
ft	Format-Table
fw	Format-Wide
gal	Get-Alias
gbp	Get-PSBreakpoint
gc	Get-Content
gci	Get-ChildItem
gcm	Get-Command
gcs	Get-PSCallStack
gdr	Get-PSDrive
ghy	Get-History
gi	Get-Item
gjb	Get-Job
gl	Get-Location
gm	Get-Member
gmo	Get-Module
gp	Get-ItemProperty
gps	Get-Process
group	Group-Object
gsn	Get-PSSession
gsnp	Get-PSSnapIn
Gsv	Get-Service
Gu	Get-Unique
Gv	Get-Variable
Gwmi	Get-WmiObject
H	Get-History
History	Get-History
Icm	Invoke-Command
Iex	Invoke-Expression
Ihy	Invoke-History
Ii	Invoke-Item
Ipal	Import-Alias
Ipcsv	Import-Csv
Ipmo	Import-Module

***Tabelle 3.1**: Vordefinierte Aliase in der PowerShell 2.0 (Forts.)*

Alias	Commandlet
Ipsn	Import-PSSession
Ise	powershell_ise.exe
Iwmi	Invoke-WMIMethod
Kill	Stop-Process
Lp	Out-Printer
Ls	Get-ChildItem
Man	help
Md	mkdir
Measure	Measure-Object
Mi	Move-Item
Mount	New-PSDrive
Move	Move-Item
Mp	Move-ItemProperty
Mv	Move-Item
Nal	New-Alias
ndr	New-PSDrive
ni	New-Item
nmo	New-Module
nsn	New-PSSession
nv	New-Variable
ogv	Out-GridView
oh	Out-Host
popd	Pop-Location
ps	Get-Process
pushd	Push-Location
pwd	Get-Location
R	Invoke-History
rbp	Remove-PSBreakpoint
rcjb	Receive-Job
rd	Remove-Item
rdr	Remove-PSDrive
ren	Rename-Item
ri	Remove-Item
rjb	Remove-Job
rm	Remove-Item
rmdir	Remove-Item
rmo	Remove-Module

***Tabelle 3.1**: Vordefinierte Aliase in der PowerShell 2.0 (Forts.)*

Alias	Commandlet
rni	Rename-Item
rnp	Rename-ItemProperty
rp	Remove-ItemProperty
rsn	Remove-PSSession
rsnp	Remove-PSSnapin
rv	Remove-Variable
Rvpa	Resolve-Path
Rwmi	Remove-WMIObject
Sajb	Start-Job
Sal	Set-Alias
Saps	Start-Process
Sasv	Start-Service
Sbp	Set-PSBreakpoint
Sc	Set-Content
Select	Select-Object
Set	Set-Variable
Si	Set-Item
Sl	Set-Location
Sleep	Start-Sleep
Sort	Sort-Object
Sp	Set-ItemProperty
Spjb	Stop-Job
Spps	Stop-Process
Spsv	Stop-Service
Start	Start-Process
Sv	Set-Variable
Swmi	Set-WMIInstance
Tee	Tee-Object
Type	Get-Content
Where	Where-Object
Wjb	Wait-Job
Write	Write-Output

***Tabelle 3.1**: Vordefinierte Aliase in der PowerShell 2.0 (Forts.)*

Neue Aliase anlegen

Set-Alias, New-Aliase

Einen neuen Alias definiert der Nutzer mit `Set-Alias` oder `New-Alias`, z.B.:

```
Set-Alias procs Get-Process
New-Alias procs Get-Process
```

Der Unterschied zwischen `Set-Alias` und `New-Alias` ist marginal: `New-Alias` erstellt einen neuen Alias und liefert einen Fehler, wenn der zu vergebende Alias schon existiert. `Set-Alias` erstellt einen neuen Alias oder überschreibt einen Alias, wenn der zu vergebende Alias schon existiert. Mit dem Parameter `-description` kann man jeweils auch einen Beschreibungstext setzen.

Man kann einen Alias nicht nur für Commandlets, sondern auch für klassische Anwendungen vergeben, z.B.:

```
Set-Alias np notepad.exe
```



Beim Anlegen eines Alias wird nicht geprüft, ob es das zugehörige Commandlet bzw. die Anwendung überhaupt gibt. Der Fehler würde erst beim Aufruf des Alias auftreten.

Man kann in Aliasdefinitionen keinen Parameter mit Werten vorbelegen. Möchten Sie zum Beispiel definieren, dass die Eingabe von „Temp" die Aktion „Get-ChildItem c:\Temp" ausführt, brauchen Sie dafür eine Funktion. Mit einem Alias geht das nicht.

```
Function Temp { Get-Childitem c:\temp }
```

Funktionen werden später (siehe *Kapitel 6 „PowerShell-Skripte"*) noch ausführlich besprochen. Die Windows PowerShell enthält zahlreiche vordefinierte Funktionen, z.B. `c:`, `d:`, `e:` sowie `mkdir` und `help`.

Die neu definierten Aliase gelten jeweils nur für die aktuelle Instanz der PowerShell-Konsole. Man kann die eigenen Alias-Definitionen exportieren mit `Export-Alias` und später wieder importieren mit `Import-Alias`. Als Speicherformate stehen das CSV-Format und das PowerShell-Skriptdateiformat (*.ps1* – siehe spätere Kapitel) zur Verfügung. Bei dem PS1-Format ist zum späteren Reimport der Datei das Skript mit dem Punkt-Operator (engl. „Dot Sourcing") aufzurufen.

	Dateiformat CSV	**Dateiformat .ps1**
Speichern	`Export-Alias c:\meinealias.csv`	`Export-Alias c:\meinealias.ps1 -as script`
Laden	`Import-Alias c:\meinealias.csv`	`. c:\meinealias.ps1`

Die Anzahl der Aliase ist im Standard auf 4096 beschränkt. Dies kann durch die Variable `$MaximumAliasCount` geändert werden.

Aliase für Eigenschaften

Aliase sind auch auf Ebene von Eigenschaften definiert. So kann man statt

```
Get-Process processname, workingset
```

auch schreiben:

```
Get-Process  name, ws
```

Diese Aliase der Attribute sind definiert in der Datei *types.ps1xml* im Installationsordner der PowerShell.

Abbildung 3.1 types.ps1xml



3.3 Ausdrücke

Mathematik

Ebenfalls als Befehl direkt in die PowerShell eingeben kann man Ausdrücke, z.B. mathematische Ausdrücke wie

```
10* (8 + 6)
```

Oder Zeichenkettenausdrücke wie:

```
"Hello "+ " " + "World"
```

Microsoft spricht hier vom Expression Mode der PowerShell im Kontrast zum Command Mode, der verwendet wird, wenn man

```
Write-Output 10* (8 + 6)
```

aufruft.

Command Mode versus Expression Mode

Die PowerShell kennt zwei Verarbeitungsmodi für Befehle: einen Befehlsmodus (Command Mode) und einen Ausdrucksmodus (Expression Mode). Im Befehlsmodus werden alle Eingaben als Zeichenketten behandelt. Im Ausdrucksmodus werden Zahlen und Operationen verarbeitet. Als Faustregel gilt: Wenn eine Zeile mit einem Buchstaben oder den Sonderzeichen kaufmännisches Und (&), Punkt (.) oder Schrägstrich ("\") beginnt, dann ist die Zeile im Befehlsmodus. Wenn die Zeile mit einer Zahl, einem Anführungszeichen (" oder '), einer Klammer ("(") oder dem @-Zeichen („Klammeraffe“) beginnt, dann ist die Zeile im Ausdrucksmodus.

Befehls- und Ausdrucksmodus können gemischt werden. Dabei muss man in der Regel runde Klammern zur Abgrenzung verwenden. In einen Befehl kann ein Ausdruck durch Klammern eingebaut werden. Außerdem kann eine Pipeline mit einem Ausdruck beginnen. Die folgende Tabelle zeigt verschiedene Beispiele zur Erläuterung.

Beispiel	Bedeutung
`2+3`	Ist ein Ausdruck. Die PowerShell führt die Berechnung aus und liefert 5.
`echo 2+3`	Ist ein reiner Befehl. „2+3“ wird als Zeichenkette angesehen und ohne Auswertung auf dem Bildschirm ausgegeben.
`echo (2+3)`	Ist ein Befehl mit integriertem Ausdruck. Auf dem Bildschirm erscheint 5.
`2+3 | echo`	Ist eine Pipeline, die mit einem Ausdruck beginnt. Auf dem Bildschirm erscheint 5.
`echo 2+3 | 7+6`	Ist eine unerlaubte Eingabe. Ausdrücke dürfen in der Pipeline nur als erstes Element auftauchen.
`$a = Get-Process`	Ist ein Ausdruck mit integriertem Befehl. Das Ergebnis wird einer Variablen zugewiesen.

***Tabelle 3.2**: Ausdrücke in der Windows PowerShell*

Beispiel	Bedeutung
`$a | Get-Process`	Ist eine Pipeline, die mit einem Ausdruck beginnt. Der Inhalt von $a wird als Parameter an Get-Process übergeben.
`Get-Process | $a`	Ist eine unerlaubte Eingabe. Ausdrücke dürfen in der Pipeline nur als erstes Element auftauchen.
"Anzahl der laufenden Prozesse: (Get-Process).Count"	Es ist wohl nicht das, was gewünscht ist, denn die Ausgabe ist: Anzahl der laufenden Prozesse: (Get-Process).Count
"Anzahl der laufenden Prozesse: $((Get-Process).Count)"	Jetzt ist die Ausgabe "Anzahl der laufenden Prozesse: 95", weil $(…) einen Unterausdruck (Subexpression) einleitet und dafür sorgt, dass Get-Process ausgeführt wird.

***Tabelle 3.2**: Ausdrücke in der Windows PowerShell (Forts.)*

3.4 Externe Befehle

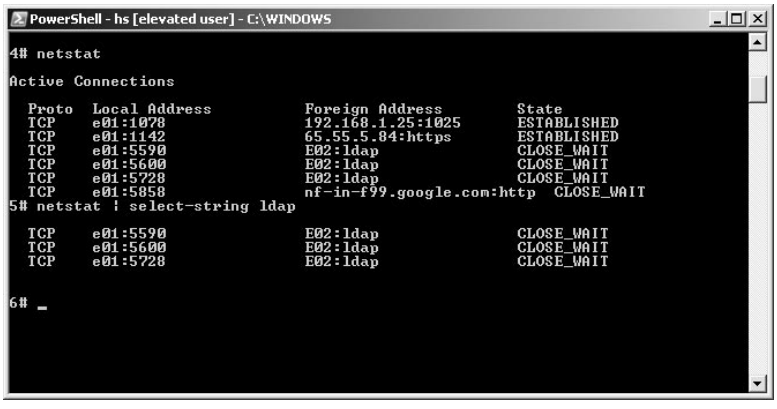
Windows-Anwendungen, DOS-Befehle, WSH-Skripte

Alle Eingaben, die nicht als Commandlets oder mathematische Formeln erkannt werden, werden als externe Anwendungen behandelt. Es können sowohl klassische Kommandozeilenbefehle (wie *ping.exe*, *ipconfig.exe* und *netstat.exe*) als auch Windows-Anwendungen ausgeführt werden.

Die Eingabe `c:\Windows\Notepad.exe` ist daher möglich, um den „beliebten" Windows-Editor zu starten. Auf gleiche Weise können auch WSH-Skripte aus der PowerShell heraus gestartet werden.

Die folgende Bildschirmabbildung zeigt den Aufruf von *netstat.exe*. Zuerst wird die Ausgabe nicht gefiltert. Im zweiten Beispiel kommt zusätzlich das Commandlet `Select-String` zum Einsatz, das nur die Zeilen ausgibt, die das Wort „LDAP" enthalten.

***Abbildung 3.2** Ausführung von netstat*



Wenn ein Leerzeichen im Pfad zu einer .exe-Datei vorkommt, dann kann man die Datei so nicht aufrufen (hier wird nach einem Befehl „T:\data\software\Windows“ gesucht):

```
T:\data\software\Windows Tools\ImageEditor.exe
```

Auch die naheliegende Lösung der Verwendung von Anführungszeichen funktioniert nicht (hier wird die Zeichenkette ausgegeben):

```
"T:\data\software\Windows Tools\ImageEditor.exe"
```

Korrekt ist die Verwendung des kaufmännischen Und (`&`), das dafür sorgt, dass der Inhalt der Zeichenkette als Befehl betrachtet und ausgeführt wird:

```
& "T:\data\software\Windows Tools\ImageEditor.exe"
```



Grundsätzlich könnte es passieren, dass ein interner Befehl der PowerShell (Commandlet, Alias oder Function) genauso heißt wie ein externer Befehl. Die PowerShell warnt in einem solchen Fall nicht vor der Doppeldeutigkeit, sondern die Ausführung erfolgt nach folgender Präferenzliste:

- Aliase
- Funktionen
- Commandlets
- Externe Befehle

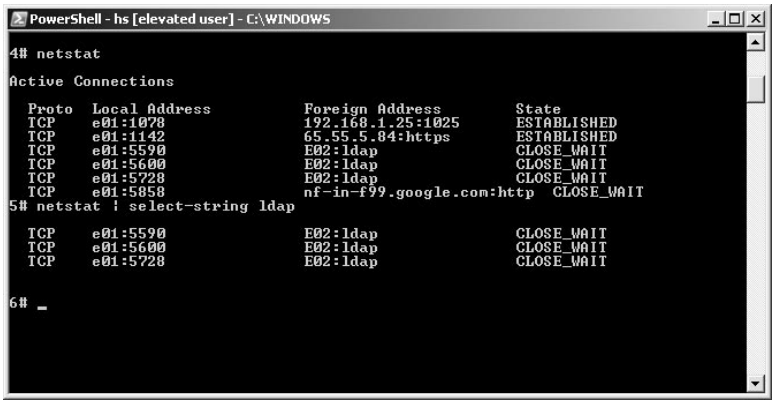


Abbildung 3.3
Ausführung von netstat

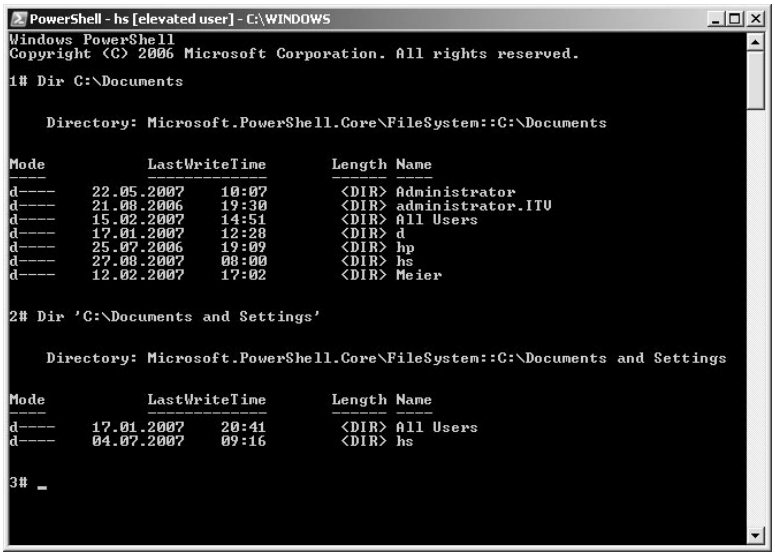
3.5 Dateinamen

Beim direkten Aufruf von Datendateien (z.B. `.doc`-Dateien) wird entsprechend den Windows-Einstellungen in der Registrierungsdatenbank die Standardanwendung gestartet und damit das Dokument geladen.



Dateinamen und Ordnerpfade müssen nur in Anführungszeichen (einfache oder doppelte) gesetzt werden, wenn sie Leerzeichen enthalten.

Abbildung 3.4
Anführungszeichen bei Pfadangaben



4 Hilfefunktionen

Dieses Kapitel beschreibt die Hilfefunktionen der Windows PowerShell.

4.1 TechNet-Dokumentation

Im Microsoft TechNet gibt es eine Dokumentation zur PowerShell 2.0, die aber leider auch zum Erscheinen der PowerShell 2.0 im Rahmen von Windows 7 und Windows Server 2008 R2 im August 2009 noch auf dem Stand der Vorabversionen von Mai 2009 war. Derzeit stehen (Stand Februar 2010) Aktualisierungen vom 16. Dezember 2009 bereit.

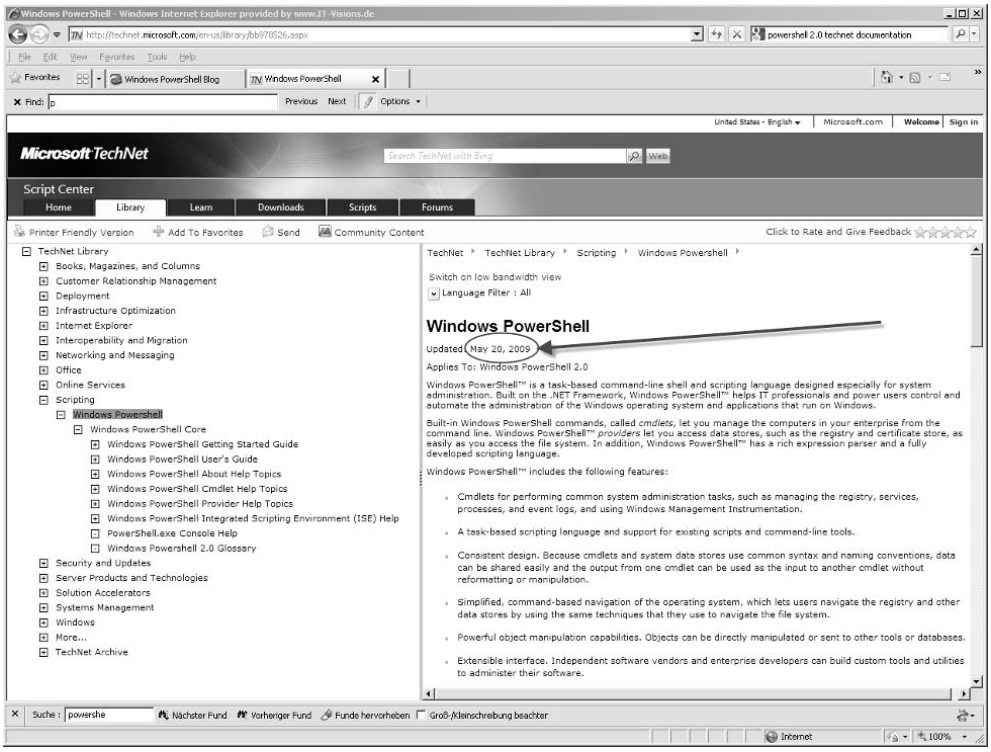


***Abbildung 4.1**: TechNet-Dokumentation der PowerShell 2.0 (Bildschirmabbildung vom 9.9.2009)*

4.2 Verfügbare Befehle

Liste der Commandlets

Die Liste aller verfügbaren Commandlets erhält man in der PowerShell auch durch

```
Get-Command
```

Dabei sind auch Muster erlaubt.

- `Get-Command Get-*` liefert alle Befehle, die mit „get“ anfangen.
- `Get-Command [gs]et-*` liefert alle Befehle, die mit „get“ oder „set“ anfangen.
- `Get-Command *-Service` liefert alle Befehle, die das Substantiv „Service“ besitzen.
- `Get-Command -noun Service` liefert ebenfalls alle Befehle, die das Substantiv „Service“ besitzen.

Das Commandlet `Get-Command` kann auch verwendet werden, um die Information zu erhalten, was die PowerShell unter einem Befehl versteht. `Get-Command` sucht nach angegebenen Namen in Commandlets, Aliasen, Funktionen, Skriptdateien und ausführbaren Dateien (siehe *Abbildung 4.2*).

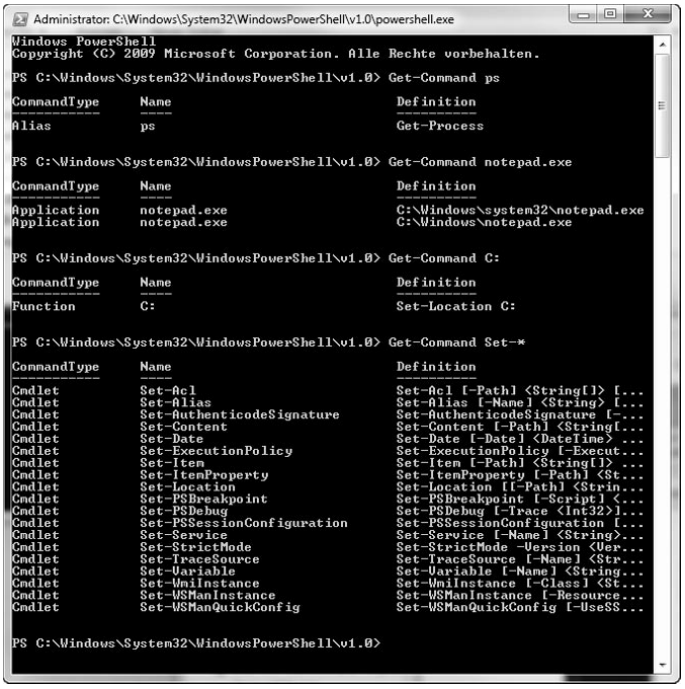


Abbildung 4.2 *Beispiele zum Einsatz von Get-Command*

Gibt man nach `Get-Command` den Namen einer *.exe*-Datei an, zeigt die PowerShell, in welchem Pfad die ausführbare Datei gefunden werden kann. Gesucht wird dabei nur in den Pfaden gemäß der Umgebungsvariablen `%Path%`.

```
Get-Command *.exe
```

zeigt eine Liste aller direkt aufrufbaren ausführbaren Dateien.

Vergleich PowerShell 1.0 und 2.0

Die PowerShell 2.0 enthält in der Grundausstattung 236 Commandlets sowie 37 eingebaute Funktionen.

In Windows PowerShell 1.0 waren es 129 Commandlets und 34 eingebaute Funktionen.

Ermitteln kann man diese Zahlen mit:

```
(Get-Command) | group commandtype
```

`Get-Command` liefert unter Windows PowerShell 2.0 sowohl Commandlets als auch eingebaute Funktionen (deren Handhabung oft der von Commandlets entspricht, nur die Art der Implementierung ist anders). Unter PowerShell 1.0 muss man die Funktionen separat zählen mit:

```
(dir function:).count
```

Wenn Sie wissen möchten,

- welche Commandlets zwischen zwei Versionen hinzugekommen sind oder
- hinsichtlich welcher Commandlets sich zwei Systeme unterscheiden,

können Sie dies wie folgt ermitteln.

Auf dem einen System exportieren Sie eine Liste der Commandlets in eine Textdatei.

Auf einem System mit PowerShell 1.0 führen Sie folgende Befehle aus, um Commandlets und Funktionen zu exportieren:

```
Get-Command | ft name -hide | out-file h:\wps1_commandlets.txt

dir function: | ft Name -hide | out-file h:\wps1_commandlets.txt -
Append
```

Auf einem System mit PowerShell 2.0 brauchen Sie nur einen Befehl (dieser exportiert Commandlets und Funktionen):

```
Get-Command | ft name -hide | out-file h:\wps\wps2_commandlets.txt
```

Dann führt man beide Textdateien auf einem System zusammen und führt dort aus:

```
$wps1 = Get-content H:\wps1_Commandlets.txt | sort
$wps2 = Get-content H:\wps2_Commandlets.txt | sort
compare-object $wps1 $wps2  -syncwindow 2000 | foreach {
[string]$_.Inputobject).Trim() } | out-file h:\wps2_Commandlets_
neu.txt
```

4.3 Erläuterungen zu den Befehlen

Get-Help

Einen Hilfetext zu einem Commandlet bekommt man über `Get-Help commandletname`, z.B.:

```
Get-Help Get-Process
```

Dabei kann man durch die Parameter `-detailed` und `-full` mehr Hilfe erhalten.

Hingegen listet `Get-Help get` alle Commandlets auf, die das Verb `Get` verwenden.

Die Hilfe erscheint abhängig von der installierten Sprachversion der Windows PowerShell. Der Autor dieses Buchs verwendet jedoch primär englische Betriebssysteme und Anwendungen.

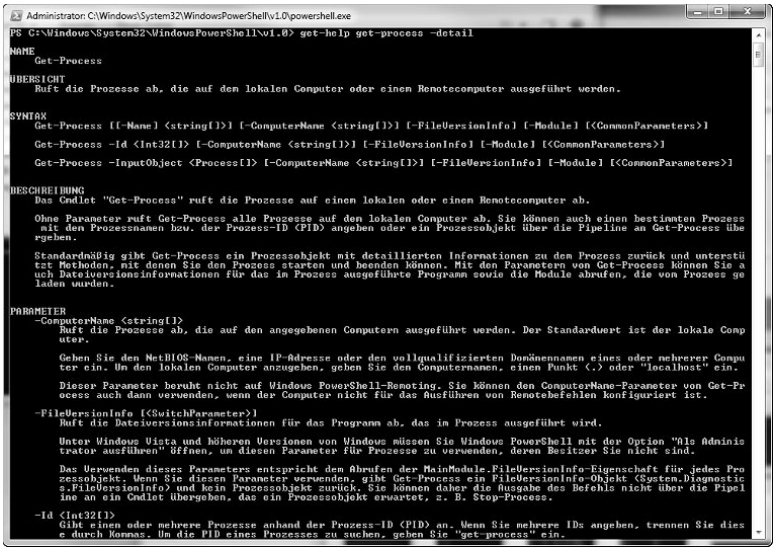


Abbildung 4.3 *Ausschnitt aus dem Hilfetext zum Commandlet Get-Process*



Alternativ zum Aufruf von `Get-Help` kann man auch den allgemeinen Parameter `-?` an das Commandlet anhängen, z.B. `Get-Process -?`. Dann erhält man die Kurzversion der Hilfe, hat aber keine Option für die ausführlicheren Versionen.

Eine grafische Hilfedatei im *.chm*-Dateiformat zur PowerShell gab es für die PowerShell 1.0 nur als Zusatz [MS01]. In PowerShell 2.0 ist die Hilfedatei direkt enthalten.

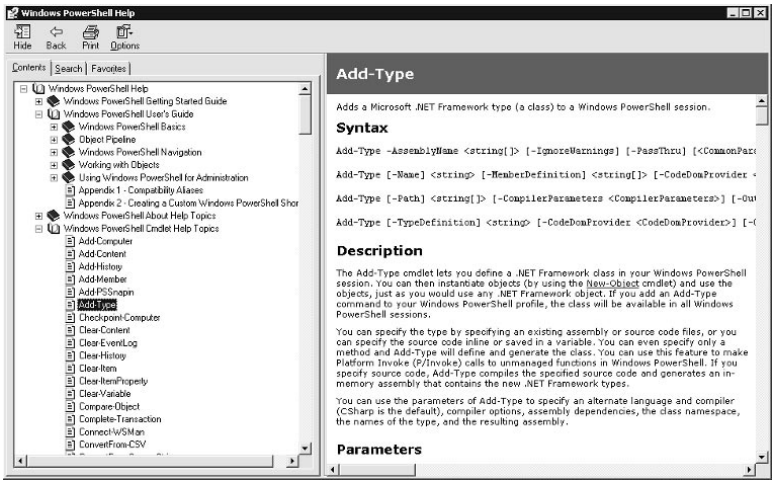


Abbildung 4.4
Die Hilfedatei zur PowerShell enthält auch die Dokumentation aller Commandlets.

Die PowerShell 1.0-Hilfedatei [MS01] ist dennoch weiterhin relevant, denn sie enthält im Gegensatz zu ihrem Nachfolger auch Hinweise zur manuellen Übersetzung von VBScript in Windows PowerShell.

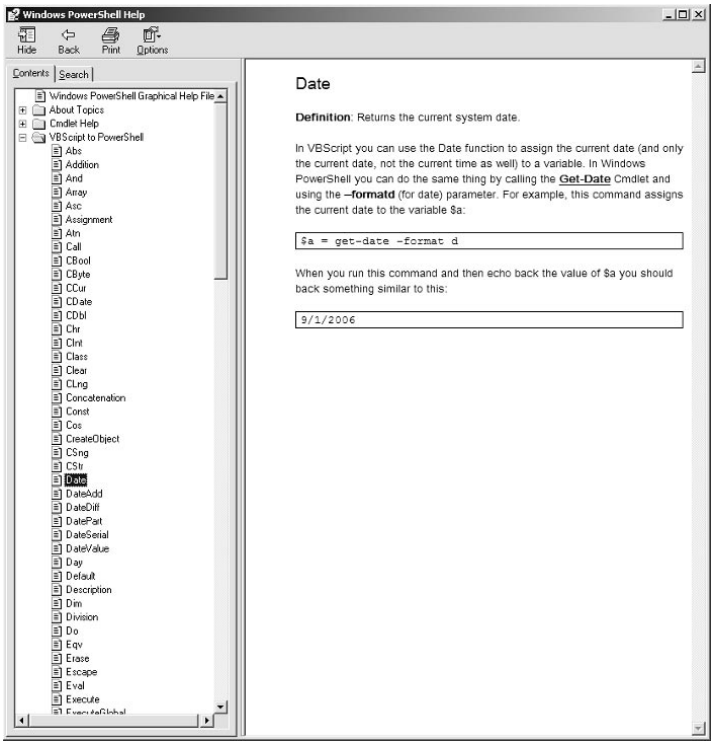


Abbildung 4.5
Hilfe zum Transfer von VBScript nach PowerShell

4.4 Hilfe zu Parametern

Um zu sehen, welche Parameter ein Befehl bietet, kann man `Get-Help` mit dem Parameter `-Parameter` verwenden:

```
Get-Help Get-Process -parameter "*" | ft name, type
```

Einige Commandlets (z.B. `New-Button` aus dem WPK (Windows Presentation Foundation (WPF) PowerShell Kit), siehe *Kapitel 45 „Grafische Benutzeroberflächen"*) haben sehr viele Parameter (in diesem Fall 180!). Hier kann man auch filtern:

```
Get-Help New-Button -parameter "on_*" | ft name, type
```

4.5 Dokumentation der .NET-Klassen

.NET-SDK

Informationen zu den .NET-Klassen, mit denen die Windows PowerShell arbeitet, finden Sie an folgenden Stellen:

- PowerShell-Dokumentation für den Namensraum `System.Management.Automation`
- .NET Framework Software Development Kit bzw. Windows Software Development Kit für .NET
- Produktspezifische Dokumentationen, z.B. Exchange Server 2007/2010-Dokumentation

Die Dokumentation zeigt die verfügbaren Klassenmitglieder (Attribut, Methoden, Ereignisse, Konstruktoren) (siehe *Abbildung 4.4*). Destruktoren werden nicht dokumentiert.



Da die Dokumentation der .NET-Klassen für Entwickler geschrieben wurde, ist sie häufig zu detailliert für PowerShell-Entwickler. Leider ist derzeit noch keine für die Bedürfnisse von Administratoren angepasste Version absehbar.

Die englische Dokumentation ist der deutschen vorzuziehen, weil es in den deutschen Übersetzungen viele Übersetzungsfehler gibt, die das Verständnis erschweren.

Die folgende Abbildung zeigt die Dokumentation der Klasse `Process` im Namensraum `System.Diagnostics`. In dem Baum links erkennt man die verschiedenen Arten von Mitgliedern: *Methoden* (Methods), *Eigenschaften* (Properties) und *Ereignisse* (Events).

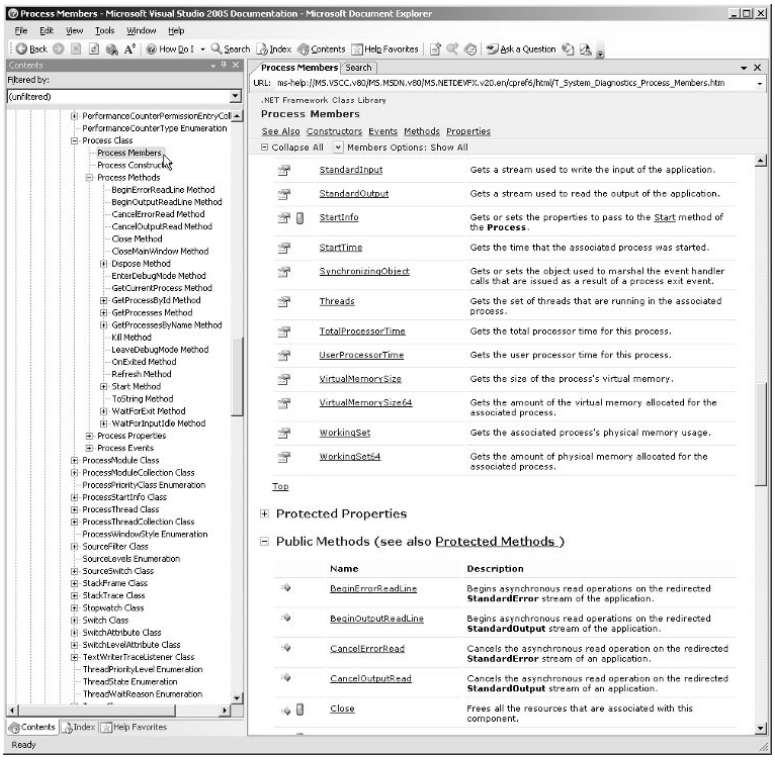


Abbildung 4.6
Ausschnitt aus der Dokumentation der .NET-Klasse System.Diagnostics.Process

5 Objektorientiertes Pipelining

Ihre Mächtigkeit entfaltet die PowerShell erst durch das objektorientierte Pipelining , also durch die Weitergabe von strukturierten Daten von einem Commandlet zum anderen.



Dieses Kapitel setzt ein Grundverständnis des Konzepts der Objektorientierung voraus. Wenn Sie diese Grundkenntnisse nicht besitzen, lesen Sie bitte zuvor im Anhang den Crashkurs „Objektorientierung" sowie den Crashkurs „.NET Framework" oder vertiefende Literatur.

5.1 Grundlagen

Pipelines

Für eine Pipeline wird – wie auch in Unix-Shells üblich und in der normalen Windows-Konsole möglich – der vertikale Strich „|" verwendet.

```
Get-Process | Format-List
```

bedeutet, dass das Ergebnis des `Get-Process`-Commandlets an `Format-List` weitergegeben werden soll. Die Standardausgabeform von `Get-Process` ist eine Tabelle. Durch `Format-List` werden die einzelnen Attribute der aufzulistenden Prozesse untereinander statt in Spalten ausgegeben.

5.2 Pipelining mit Objekten

Typisierte Objekte

Objektorientierung ist die herausragende Eigenschaft der Windows PowerShell: Commandlets können durch Pipelines mit anderen Commandlets verbunden werden. Anders als Pipelines in Unix-Shells tauschen die Commandlets der PowerShell keine Zeichenketten, sondern typisierte .NET-Objekte aus. Das objektorientierte Pipelining ist im Gegensatz zum in den Unix-Shells und in der normalen Windows-Shell (*cmd.exe*) verwendeten zeichenkettenbasierten Pipelining nicht abhängig von der Position der Informationen in der Pipeline.

Typisierte Datenweitergabe

In einer Pipeline wie

```
Get-Process | Where-Object {$_.name -eq "iexplore"} | Format-Table
ProcessName, WorkingSet64
```

ist das dritte Commandlet daher nicht auf eine bestimmte Anordnung und Formatierung der Ausgabe von vorherigen Commandlets angewiesen, sondern es greift über den sogenannten Reflection-Mechanismus (den eingebauten Komponentenerforschungsmechanismus des .NET Frameworks) direkt auf die Eigenschaften der Objekte in der Pipeline zu.



Genau genommen bezeichnet Microsoft das Verfahren als „Extended Reflection" bzw. „Extended Type System (ETS)", weil die PowerShell in der Lage ist, Objekte um zusätzliche Eigenschaften anzureichern, die in der Klassendefinition gar nicht existieren.

.NET-Objekte

Im obigen Beispiel legt `Get-Process` ein .NET-Objekt der Klasse `System.Diagnostics.Process` für jeden laufenden Prozess in die Pipeline. `System.Diagnostics.Process` ist eine Klasse aus der .NET-Klassenbibliothek. Commandlets können aber jedes beliebige .NET-Objekt in die Pipeline legen, also auch einfache Zahlen oder Zeichenketten, da es in .NET keine Unterscheidung zwischen elementaren Datentypen und Klassen gibt. Eine Zeichenkette in die Pipeline zu legen, wird aber in der PowerShell die Ausnahme bleiben, denn der typisierte Zugriff auf Objekte ist wesentlich robuster gegenüber möglichen Änderungen als die Zeichenkettenauswertung mit regulären Ausdrücken.

Deutlicher wird der objektorientierte Ansatz, wenn man als Attribut keine Zeichenkette heranzieht, sondern eine Zahl. `WorkingSet64` ist ein 64 Bit langer Zahlenwert, der den aktuellen Speicherverbrauch eines Prozesses repräsentiert. Der folgende Befehl liefert alle Prozesse, die aktuell mehr als 20 Megabyte verbrauchen:

```
Get-Process | Where-Object {$_.WorkingSet64 -gt 20*1024*1024 }
```

Anstelle von 20*1024*1024 hätte man auch das Kürzel „20MB" einsetzen können. Außerdem kann man `Where-Object` mit einem Fragezeichen abkürzen. Die kurze Variante des Befehls wäre dann also:

```
ps | ? {$_.ws -gt 20MB }
```

Wenn nur ein einziges Commandlet angegeben ist, dann wird das Ergebnis auf dem Bildschirm ausgegeben. Auch wenn mehrere Commandlets in einer Pipeline zusammengeschaltet sind, wird das Ergebnis des letzten Commandlets auf dem Bildschirm ausgegeben. Wenn das letzte Commandlet keine Daten in die Pipeline wirft, erfolgt keine Ausgabe.

Zugriff auf einzelne Objekte

Die meisten Commandlets legen ganze Mengen von Objekten in die Pipeline (z.B. `Get-Process` eine Liste der Prozesse und Get-Service eine Liste der Dienste). Einige Commandlets legen aber nur einzelne Objekte in die Pipeline. Ein Beispiel dafür ist `Get-Date`, das ein einziges Objekt des Typs `System.DateTime` in die Pipeline legt. Ruf man `Get-Date` ohne Weiteres auf, werden das aktuelle Datum und die aktuelle Zeit ausgegeben.

```
PS C:\Windows\System32> Get-Date
Mittwoch, 9. September 2009 14:01:51
```

Abbildung 5.1
Das aktuelle Datum mit Zeit

Bei einer Objektmenge kann man, wie oben bereits gezeigt, mit `Where-Object` filtern. Es ist aber auch möglich, gezielt einzelne Objekte über ihre Position in der Pipeline anzusprechen. Die Positionsangabe ist in eckige Klammern zu setzen und die Zählung beginnt bei 0. Der Pipeline-Ausdruck ist in runde Klammern zu setzen.

Beispiele:

Der erste Prozess:

```
(Get-Process)[0]
```

Der dreizehnte Prozess:

```
(Get-Process)[12]
```

Der dreizehnte Prozess in der Liste der Prozesse, die mehr als 20 MB Hauptspeicher brauchen:

```
(Get-Process | where-object { $_.WorkingSet64 -gt 20mb } )[12]
```

```
PS C:\Windows\System32> (get-process)[0]

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
     20       2     1968       2664    17     0,03   2784 cmd

PS C:\Windows\System32> (get-process)[12]

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
     69       9     1484       4196    41     0,03   2100 dlpwdnt

PS C:\Windows\System32> (get-process | where-object { $_.WorkingSet64 -gt 20mb } )[12]

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
    685      29    53924      59544   291    34,39   4984 powershell

PS C:\Windows\System32>
```

Abbildung 5.2
Zugriff auf einzelne Prozessobjekte

Zugriff auf einzelne Werte

Manchmal möchte man nicht ein komplettes Objekt bzw. eine komplette Objektmenge verarbeiten, sondern nur eine einzelne Eigenschaft.

Oben wurde bereits gezeigt, wie man mit `Format-Table` auf einzelne Eigenschaften zugreifen kann:

```
Get-Process | Format-Table ProcessName, WorkingSet64
```

Hat man nur ein einzelnes Objekt in Händen, geht das auch:

```
(Get-Process)[0] | Format-Table ProcessName, WorkingSet64
```

In diesem Fall kann man aber anstelle der tabellarischen Ausgabe auch direkt auf Eigenschaften zugreifen. Dafür verwendet man die in objektorientierten Sprachen üblich Punktnotation, d.h., man trennt das Objekt und die abzurufende Eigenschaft durch einen Punkt.

Beispiele:

```
(Get-Process)[0].Processname
```

Die Ausgabe ist eine einzelne Zeichenkette.

```
(Get-Process)[0].WorkingSet64
```

Die Ausgabe ist eine einzelne Zahl.

Mit den Einzelwerten kann man weiterrechnen, z.B. errechnet man so die Speichernutzung in Megabyte:

```
(Get-Process)[0].WorkingSet64 / 1MB
```

Abbildung 5.3
Ausgabe zu den obigen Beispielen

```
PS C:\Windows\System32> (get-process)[0] ! Format-Table ProcessName, WorkingSet64

ProcessName                                                           WorkingSet64
-----------                                                           ------------
cmd                                                                        2727936


PS C:\Windows\System32> (get-process)[0].Processname
cmd
PS C:\Windows\System32> (get-process)[0].WorkingSet64
2727936
PS C:\Windows\System32> (get-process)[0].WorkingSet64 / 1MB
2,6015625
PS C:\Windows\System32> _
```

Weitere Anwendungsfälle seien am Beispiel `Get-Date` gezeigt (siehe *Abbildung 5.4*). `Year`, `Day`, `Month`, `Hour` und `Minute` sind einige der zahlreichen Eigenschaften der Klasse `DateTime`, die `Get-Date` liefert.

Abbildung 5.4
Zugriff auf einzelne Werte aus dem aktuellen Datum/der aktuellen Zeit

```
PS C:\Windows\System32> (Get-Date).Year
2009
PS C:\Windows\System32> (Get-Date).Day
9
PS C:\Windows\System32> (Get-Date).Month
9
PS C:\Windows\System32> (Get-Date).Hour
14
PS C:\Windows\System32> (Get-Date).Minute
4
```

Für die Objektmenge in der Pipeline als Ganzes gesehen kann man in der Regel nur einen sinnvollen Wert abrufen: `Count`.

Beispiel: Wie viele Prozesse gibt es, die mehr als 20 MB Speicher verbrauchen?

```
(Get-Process | where-object { $_.WorkingSet64 -gt 20mb }).Count
```

Abbildung 5.5
Aufruf von Count für eine Pipeline

```
PS C:\Windows\System32> (get-process ! where-object { $_.WorkingSet64 -gt 20mb }).Count
21
PS C:\Windows\System32> _
```

Methoden ausführen

Stop-Process

Der folgende PowerShell-Pipeline-Befehl beendet alle Instanzen des Internet Explorers auf dem lokalen System, indem das Commandlet `Stop-Process` die Instanzen des betreffenden Prozesses von `Get-Process` empfängt.

```
Get-Process  iexplore | Stop-Process
```

Kill()

Die Objekt-Pipeline der PowerShell hat noch weitere Möglichkeiten: Gemäß dem objektorientierten Paradigma haben .NET-Objekte nicht nur Attribute, sondern auch Methoden. In einer Pipeline kann der Administrator daher auch die Methoden der Objekte aufrufen. Objekte des Typs `System.Diagnostics.Process` besitzen zum Beispiel eine Methode `Kill()`. Der Aufruf dieser Methode ist in der PowerShell gekapselt in der Methode `Stop-Process`.

Wer sich mit dem .NET Framework gut auskennt, könnte die `Kill()`-Methode auch direkt aufrufen. Dann ist aber eine explizite `ForEach`-Schleife notwendig. Die Commandlets iterieren automatisch über alle Objekte der Pipeline, die Methodenaufrufe aber nicht.

```
Get-Process iexplore | Foreach-Object { $_.Kill() }
```

Durch den Einsatz von Aliasen geht das auch kürzer:

```
ps | ? { $_.name -eq "iexplore" } | % { $_.Kill() }
```

Der Einsatz der Methode `Kill()` diente hier nur zur Demonstration, dass die Pipeline tatsächlich Objekte befördert. Eigentlich ist die gleiche Aufgabe besser mit dem eingebauten Commandlet `Stop-Process` zu lösen.



Vergessen Sie beim Aufruf von Methoden nicht die runden Klammern, auch wenn die Methoden keine Parameter besitzen. Ohne die Klammern erhalten Sie Informationen über die Methode, es erfolgt aber kein Aufruf.

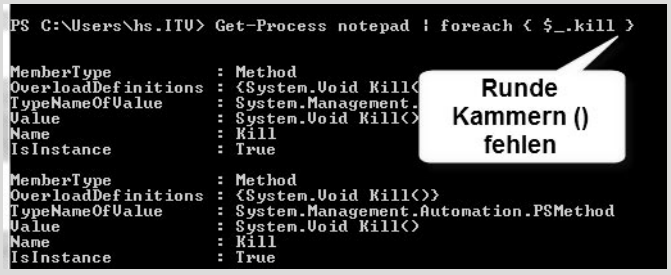


***Abbildung 5.6**: Folgen des vergessenen Klammernpaars*

Dies funktioniert aber nur dann gut, wenn es auch Instanzen des Internet Explorers gibt. Wenn alle beendet sind, meldet `Get-Process` einen Fehler. Dies kann das gewünschte Verhalten sein. Mit einer etwas anderen Pipeline wird dieser Fehler jedoch unterbunden:

```
Get-Process | Where-Object { $_.Name -eq "iexplore" } |
Stop-Process
```

Die zweite Pipeline unterscheidet sich von der ersten dadurch, dass das Filtern der Prozesse aus der Prozessliste nun nicht mehr von `Get-Process` erledigt wird, sondern durch ein eigenes Commandlet mit Namen `Where-Object` in der Pipeline selbst durchgeführt wird. `Where-Object` ist toleranter als `Get-Process` in Hinblick auf die Möglichkeit, dass es kein passendes Objekt gibt.

`ps` ist ein Alias für `Get-Process`, `Kill` für `Stop-Process`. Außerdem hat `Get-Process` eine eingebaute Filterfunktion. Um alle Instanzen des Internet Explorers zu beenden, kann man also statt

```
Get-Process | Where-Object { $_.Name -eq "iexplore" } |
Stop-Process
```

auch schreiben:

```
ps -p "iexplore" | Kill
```

Weitere Beispiele für die Aufrufe von Methoden seien am Beispiel von `Get-Date` gezeigt, das ja nur ein Objekt der Klasse `DateTime` liefert. Die Klasse `DateTime` bietet zahlreiche Methoden an, um Datum und Zeit auf bestimmte Weise darzustellen, z.B. `GetShortDateString()`, `GetLongDateString()`, `GetShortTimeString()` und `GetLongTimeString()`. Die Ausgaben zeigt die Bildschirmabbildung.

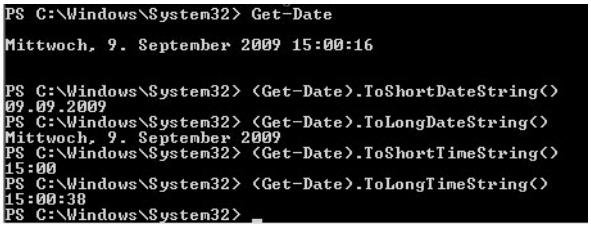


Abbildung 5.7
Ausgaben der Methoden der Klasse DateTime

Pipelining von Parametern

Die Pipeline kann jegliche Art von Information befördern, auch einzelne elementare Daten. Einige Commandlets unterstützen es, dass auch die Parameter aus der Pipeline ausgelesen werden. Der folgende Pipeline-Befehl führt zu einer Auflistung aller Windows-Systemdienste, die mit dem Buchstaben „I“ beginnen.

```
"i*" | Get-Service
```

Pipelining von klassischen Befehlen

Grundsätzlich dürfen auch klassische Kommandozeilenanwendungen in der PowerShell verwendet werden. Wenn man einen Befehl wie *netstat.exe* oder *ping.exe* ausführt, dann legen diese eine Menge von Zeichenketten in die Pipeline: Jede Ausgabezeile ist eine Zeichenkette.

Diese Zeichenketten kann man sehr gut mit dem Commandlet `Select-String` auswerten. `Select-String` lässt nur diejenigen Zeilen die Pipeline passieren, die auf den angegebenen regulären Ausdruck zutreffen.

In dem folgenden Beispiel werden nur diejenigen Zeilen der Ausgabe von *netstat.exe* gefiltert, die ein großes „E“ gefolgt von zwei Ziffern enthalten.

Die Syntax der regulären Ausdrücke in .NET wird in *Kapitel 7 „PowerShell-Skriptsprache“* noch etwas näher beschrieben werden.

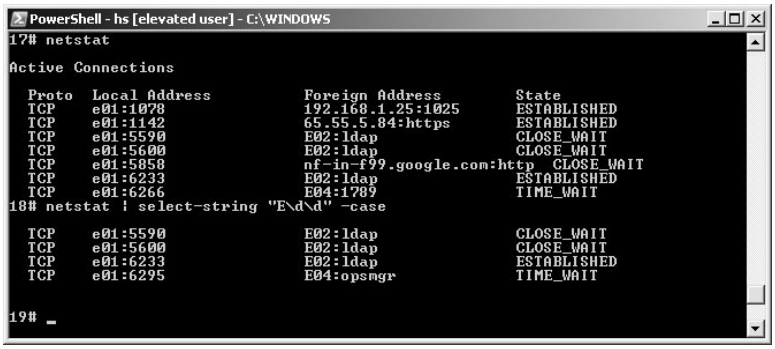


Abbildung 5.8 *Einsatz von Select-String zur Filterung von Ausgaben klassischer Kommandozeilenwerkzeuge*

5.3 Pipeline Processor

PowerShell Pipeline Processor

Für die Übergabe der .NET-Objekte zwischen den Commandlets sorgt der *PowerShell Pipeline Processor* (siehe folgende Grafik). Die Commandlets selbst müssen sich weder um die Objektweitergabe noch um die Parameterauswertung kümmern.

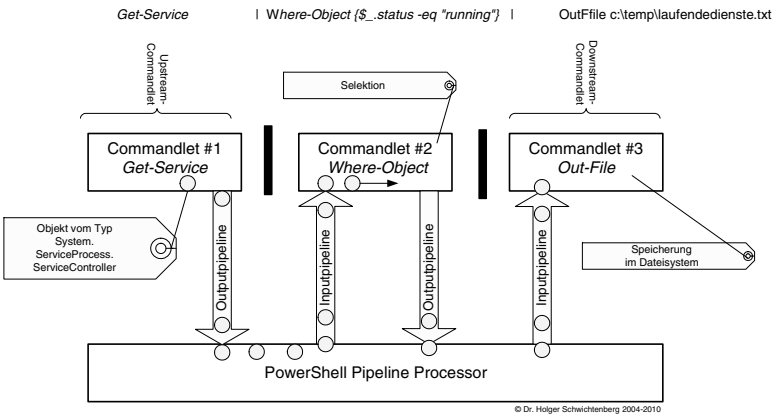


Abbildung 5.9 *Der Pipeline Processor befördert die Objekte vom Downstream-Commandlet zum Upstream-Commandlet. Die Verarbeitung ist in der Regel asynchron.*

Streaming

Wie das obige Bild schon zeigt, beginnt ein nachfolgendes Commandlet mit seiner Arbeit, sobald es ein erstes Objekt aus der Pipeline erhält. Es kann also sein, dass das erste Commandlet noch gar nicht alle Objekte erzeugt hat, bevor die folgenden Commandlets schon die ersten Objekte asynchron weiterverarbeiten. Ein Commandlet wird sofort aufgerufen, sobald das erste Objekt bereitsteht. Man nennt dies „Streaming-Verarbeitung". Streaming-Verarbeitung ist viel schneller als die klassische sequentielle Verarbeitung, weil die folgenden Commandlets in der Pipeline nicht auf vorhergehende warten müssen.

Aber nicht alle Commandlets beherrschen die asynchrone Streaming-Verarbeitung. Commandlets, die alle Objekte naturgemäß erst mal kennen müssen, bevor sie überhaupt ihren Zweck erfüllen können (z.B. `Sort-Object` zum Sortieren und `Group -Object` zum Gruppieren), blockieren die asynchrone Verarbeitung.

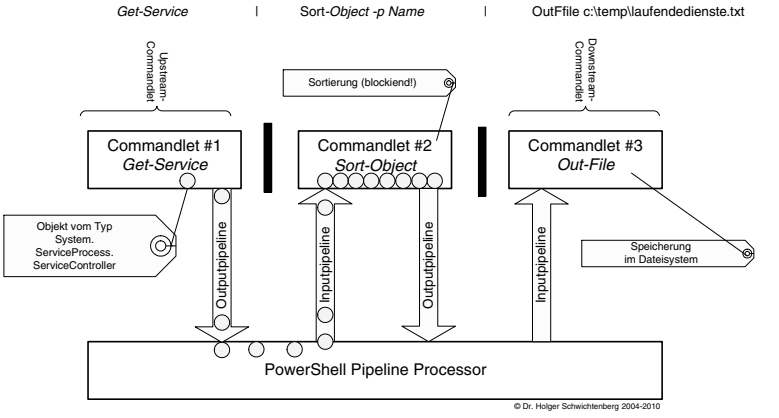


Abbildung 5.10 *Sort-Object blockiert die direkte Weitergabe. Erst wenn alle Objekte angekommen sind, kann das Commandlet sortieren.*

Es gibt auch einige Commandlets, die zwar asynchron arbeiten könnten, aber leider nicht so programmiert wurden, um dies zu unterstützen.

Die folgende Bildschirmabbildung zeigt einige Parameter des Commandlets `Get-Service`. Diese Liste erhält man durch den Befehl `Get-Help Get-Service -Parameter *`.



Abbildung 5.11
Hilfe zu den Parametern des Commandlets Get-Service

Interessant sind die mit gelbem Pfeil markierten Stellen. Nach „Accept pipeline Input" kann man jeweils nachlesen, ob der Parameter des Commandlets aus den vorhergehenden Objekten in der Pipeline „befüttert" werden kann.

Bei „-Name" steht `ByValue` und `ByPropertyName`. Dies bedeutet, dass der Name sowohl das ganze Objekt in der Pipeline sein darf als auch Teil eines Objekts.

ByPropertyName

Im Fall von

```
"BITS" | Get-Service
```

ist der Pipeline-Inhalt eine Zeichenkette (ein Objekt vom Typ `String`), die als Ganzes auf `Name` abgebildet werden kann.

ByValue Es funktioniert aber auch folgender Befehl, der alle Dienste ermittelt, deren Name genauso lautet wie der Name eines laufenden Prozesses:

```
Get-Process | Get-Service -ea silentlycontinue | ft name
```

Dies funktioniert über die zweite Option (`ByPropertyName`), denn `Get-Process` liefert Objekte des Typs `Process`, die ein Attribut namens `Name` haben. Der Parameter `Name` von `Get-Service` wird auf dieses `Name`-Attribut abgebildet.

Beim Parameter `-InputObject` ist hingegen nur „ByValue“ angegeben. Hier erwartet `Get-Service` gerne Instanzen der Klasse `ServiceController`. Es gibt aber keine Objekte, die ein Attribut namens `InputObject` haben, in dem dann `ServiceController`-Objekte stecken.

Zahlreiche Commandlets besitzen einen Parameter `-InputObject`, insbesondere die allgemeinen Verarbeitungscommandlets wie `Where-Object`, `Select-Object` und `Measure-Object`, die Sie im nächten Kapitel kennenlernen werden. Der Name `-InputObject` ist eine Konvention.

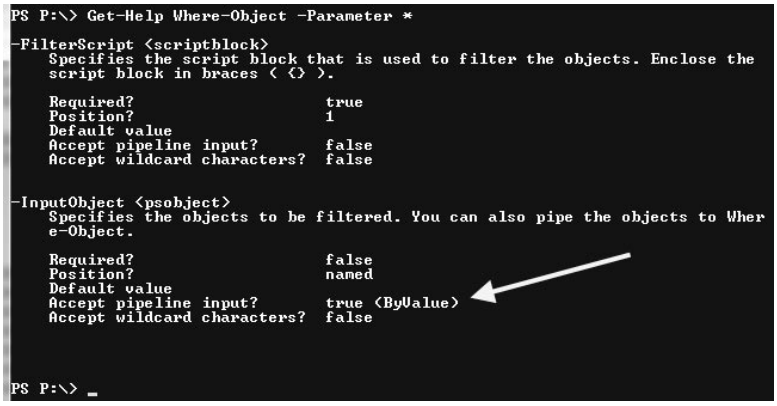


Abbildung 5.12 Parameter des Commandlets Where-Object

5.4 Komplexe Pipelines

Die Pipeline kann beliebig lang sein, d.h., die Anzahl der Befehle in einer einzigen Pipeline ist nicht begrenzt. Ein Beispiel für eine komplexere Pipeline lautet:

```
Get-ChildItem h:\daten -r -filter *.doc
| Where-Object { $_.Length -gt 40000 }
| Select-Object Name, Length
| Sort-Object Length
| Format-List
```

`Get-ChildItem` ermittelt alle Microsoft-Word-Dateien im Ordner *h:\Daten* und in seinen Unterordnern. Durch das zweite Commandlet (`Where-Object`) wird die Ergebnismenge auf diejenigen Objekte beschränkt, bei denen das Attribut `Length` größer ist als 40000. `Select-Object` beschneidet alle Attribute aus `Name` und `Length`. Durch das vierte Commandlet in der Pipeline wird die Ausgabe nach dem Attribut `Length` sortiert. Das letzte Commandlet schließlich erzwingt eine Listendarstellung.

Die Reihenfolge der einzelnen Befehle in der Pipeline ist dabei nicht beliebig. Keineswegs kann man im obigen Befehl die Sortierung hinter die Formatierung setzen, weil nach dem Formatieren zwar noch ein Objekt existiert, dieses aber einen Textstrom repräsentiert. `Where-Object` und `Sort-Object` könnte man vertauschen; aus Gründen des Ressourcenverbrauchs sollte man aber erst einschränken und dann die verringerte Liste sortieren.

Ein Commandlet kann auf alle Attribute und Methoden der .NET-Objekte, die das vorhergehende Commandlet in die Pipeline gelegt hat, zugreifen. Die Mitglieder der Objekte können entweder durch Parameter der Commandlets (z.B. in `Sort-Object Length`) oder durch den expliziten Verweis auf das aktuelle Pipeline-Objekt (`$_`) in einer Schleife oder Bedingung (z.B. `Where-Object { $_.Length -gt 40000 }`) genutzt werden.



Nicht alle Aneinanderreihungen von Commandlets ergeben einen Sinn. Einige Aneinanderreihungen sind auch gar nicht erlaubt. Ein Commandlet kann erwarten, dass es bestimmte Arten von Eingabeobjekten gibt. Am besten sind aber Commandlets, die jede Art von Eingabeobjekt verarbeiten können.

5.5 Analyse des Pipeline-Inhalts

Eine der größten Herausforderungen bei der Arbeit mit der PowerShell besteht darin, die folgenden Fragen zu beantworten:

- Welchen Typ haben die Objekte, die ein Commandlet in die Pipeline legt?
- Welche Attribute und Methoden haben diese Objekte?

Die Hilfe der Commandlets ist hier nicht immer hilfreich. Bei `Get-Service` kann man lesen:

```
RETURN TYPE
   System.ServiceProcess.ServiceController
```

Aber bei `Get-Process` heißt es nur wenig hilfreich:

```
RETURN TYPE
   Object
```

In keinem Fall sind in der PowerShell-Benutzerdokumentation ([MS01] und [MS02]) die Attribute und die Methoden der resultierenden Objekte genannt. Diese findet man nur in der MSDN-Dokumentation des .NET Frameworks.

Im Folgenden werden zwei hilfreiche Commandlets sowie eine Methode aus dem .NET Framework vorgestellt, die im Alltag helfen, zu erforschen, was man in der Pipeline hat:

- `GetType()`
- `Get-PipelineInfo`
- `Get-Member`

Methode GetType()

System.Type

Da jede PowerShell-Variable eine Instanz einer .NET-Klasse ist, besitzt jedes Objekt in der Pipeline die Methode `GetType()`, die es von der Mutter aller .NET-Klassen (`System.Object`) erbt. `GetType()` liefert ein `System.Type`-Objekt mit zahlreichen Informationen. Meistens interessiert man sich nur für den Klassennamen, den man aus `Fullname` (mit Namensraum) oder `Name` (ohne Namensraum) auslesen kann. `GetType()` ist eine Methode und daher muss der Pipeline-Inhalt in runden Klammern stehen.

Beispiele zeigt die folgende Bildschirmabbildung.

Abbildung 5.13
Einsatz von GetType()

```
PS C:\Users\HS> (Get-Date).GetType()

IsPublic IsSerial Name                                     BaseType
-------- -------- ----                                     --------
True     True     DateTime                                 System.ValueType

PS C:\Users\HS> (Get-Process).GetType()

IsPublic IsSerial Name                                     BaseType
-------- -------- ----                                     --------
True     True     Object[]                                 System.Array

PS C:\Users\HS> (Get-Process)[0].GetType()

IsPublic IsSerial Name                                     BaseType
-------- -------- ----                                     --------
True     False    Process                                  System.ComponentModel.Component

PS C:\Users\HS> (Get-Process)[0].GetType().Fullname
System.Diagnostics.Process
PS C:\Users\HS> _
```

Erläuterung: „Name“ ist der Name der Klasse, zu der die Objekte in der Pipeline gehören. „BaseType“ ist der Name der Oberklasse. .NET unterstützt Vererbung, d.h., eine Klasse kann von einer anderen erben. Dies ist für die PowerShell meist aber irrelevant und Sie können diese Information ignorieren.

Bei `Get-Date()` ist ein `DateTime`-Objekt in der Pipeline. Der zweite Aufruf liefert nur die Information, dass eine Menge von Objekten in der Pipeline ist. Bei der Anwendung von `GetType()` auf eine Objektmenge in der Pipeline kann man leider noch nicht den Typ erkennen. Hintergrund ist, dass in einer Pipeline Objekte verschiedener Klassen sein können. Der dritte Aufruf, bei dem gezielt ein Objekt (das erste) herausgenommen wird, zeigt dann wieder an, dass es sich um `Process`-Objekte handelt. Den ganzen Klassennamen inklusive des Namensraums bekommt man nur, wenn man explizit die Eigenschaft `FullName` abfragt.

Get-PipelineInfo

Get-PipelineInfo

Das Commandlet `Get-PipelineInfo` aus den PowerShell Extensions von *www.IT-Visions.de* liefert drei wichtige Informationen über die Pipeline-Inhalte:

- Anzahl der Objekte in der Pipeline (die Objekte werden durchnummeriert)
- Typ der Objekte in der Pipeline (ganzer Name der .NET-Klasse)
- Zeichenkettenrepräsentation der Objekte in der Pipeline

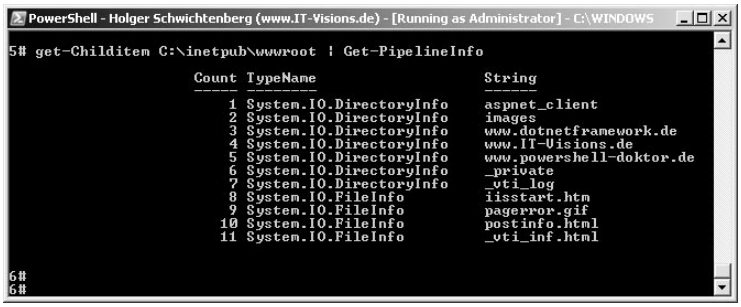


Abbildung 5.14 *Get_PipelineInfo liefert Informationen, dass sich in dem Dateisystemordner elf Objekte befinden. Davon sind sieben Unterordner (Klasse DirectoryInfo) und vier Dateien (Klasse FileInfo).*

Das Stichwort Zeichenkettenrepräsentation (Spalte „String" in der Bildschirmabbildung) ist erklärungsbedürftig: Jedes .NET-Objekt besitzt eine Methode `ToString()`, die das Objekt in eine Zeichenkette umwandelt, denn `ToString()` ist in der „Mutter aller .NET-Klassen" `System.Object` implementiert und wird an alle .NET-Klassen und somit auch deren Instanzen weitergegeben. Ob `ToString()` eine sinnvolle Ausgabe liefert, hängt von der jeweiligen Klasse ab. Im Fall von `System.Diagnostics.Process` werden der Klassenname und der Prozessname ausgegeben. Dies kann man leicht mit `gps | foreach { $_ .ToString () }` ermitteln (siehe *Abbildung 22.35*). Bei der Klasse `System.ServiceProcess.ServiceController`, deren Instanzen von `Get-Service` geliefert werden, ist die Konvertierung hingegen nicht so gut, denn die Zeichenkette enthält nur den Klassennamen, so dass die einzelnen Instanzen gar nicht unterschieden werden können.



Die Konvertierung in den Klassennamen ist das Standardverhalten, das von `System.Object` geerbt wird, und dieses Standardverhalten ist leider auch üblich, da sich die Entwickler der meisten .NET-Klassen bei Microsoft nicht die „Mühe" gemacht haben, eine sinnvolle Zeichenkettenrepräsentanz zu definieren.

`ToString()` ist üblicherweise keine Serialisierung des kompletten Objektinhalts, sondern spiegelt in der Regel nur den Primärschlüssel des Objekts wider.

Abbildung 5.15 *Anwendung von ToString() auf Instanzen der Klasse System.Diagnostics.Process*

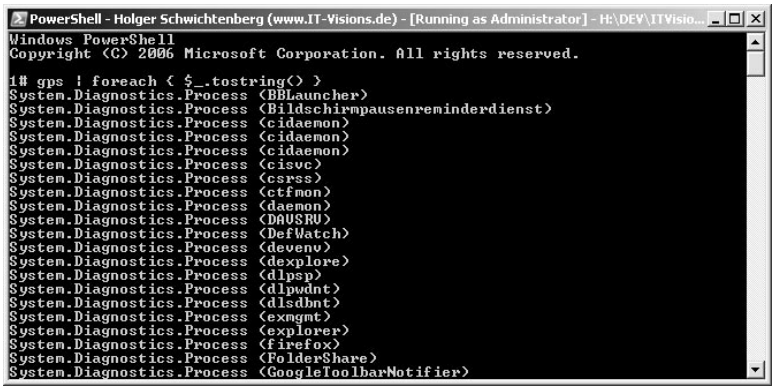
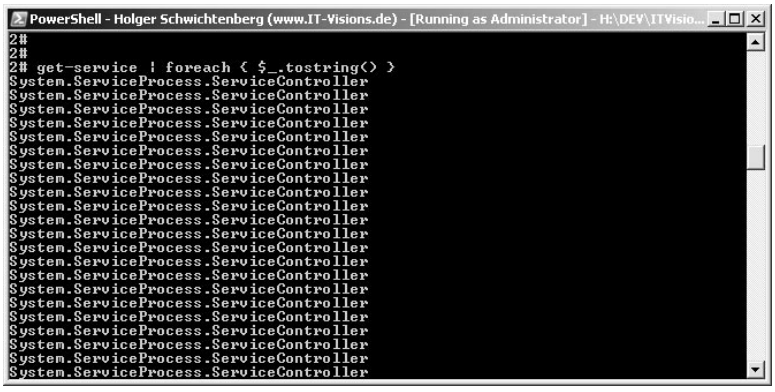


Abbildung 5.16 *Anwendung von ToString() auf Instanzen der Klasse System.ServiceProcess.ServiceController*



Get-Member

Get-Member

Das Commandlet `Get-Member` (Alias: `gm`) ist hier hilfreich: Es zeigt den .NET-Klassennamen für die Objekte in der Pipeline sowie die Attribute und Methoden dieser Klasse. Für `Get-Process | Get-Member` ist die Ausgabe so lang, dass man dazu zwei Bildschirmabbildungen braucht.

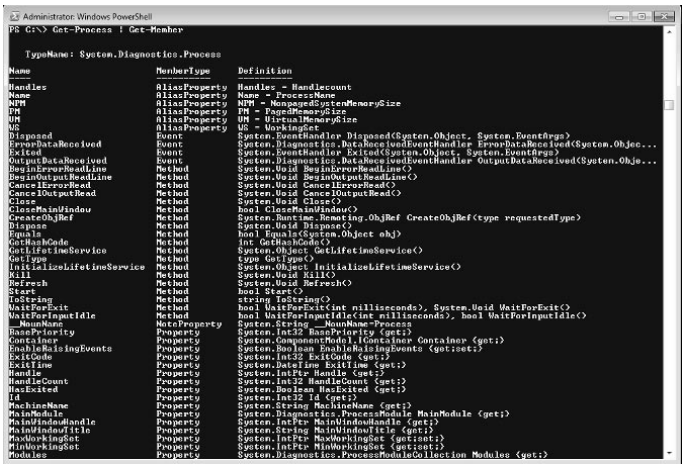


Abbildung 5.17
Teil 1 der Ausgabe von Get-Process | Get-Member

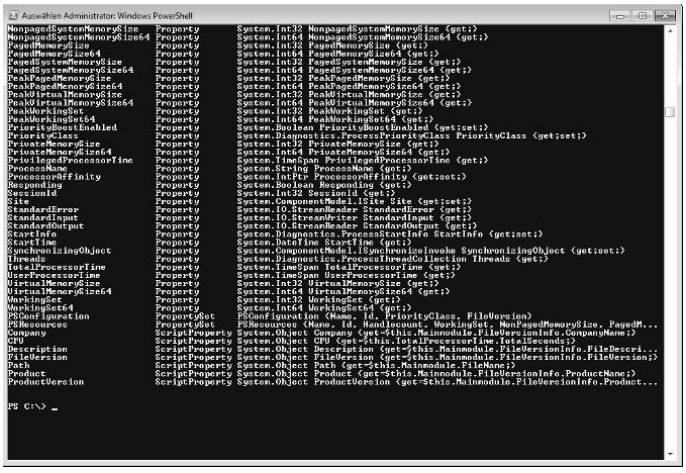
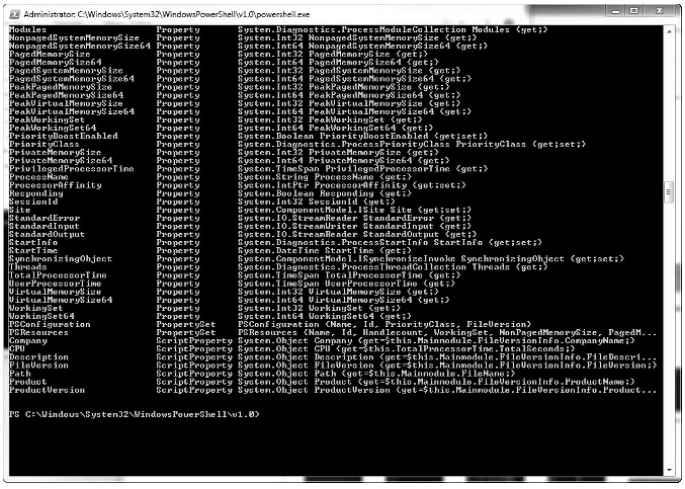


Abbildung 5.18
Teil 2 der Ausgabe von Get-Process | Get-Member





Wenn sich mehrere verschiedene Objekttypen in der Pipeline befinden, werden die Mitglieder aller Typen ausgegeben, gruppiert durch die Kopfsektion, die mit „TypeName:“ beginnt.

Die Ausgabe zeigt, dass aus der Sicht der PowerShell eine .NET-Klasse sieben Arten von Mitgliedern hat:

- Method (Methode)
- Property (Eigenschaft)
- PropertySet (Eigenschaftssatz)
- NoteProperty (Notizeigenschaft)
- ScriptProperty (Skripteigenschaft)
- CodeProperty (Codeeigenschaft)
- AliasProperty (Aliaseigenschaft)



Von den oben genannten Mitgliedsarten sind nur „Method“ und „Property“ tatsächliche Mitglieder der .NET-Klasse. Alle anderen Mitgliedsarten sind Zusätze, welche die PowerShell mittels des bereits erwähnten Extended Type System (ETS) dem .NET-Objekt hinzugefügt hat.

Methoden (Mitgliedsart Method)

Method Methoden (Mitgliedsart Method) sind Operationen, die man auf dem Objekt aufrufen kann und die eine Aktion auslösen, z.B. `Kill()` beendet den Prozess. Methoden können aber auch Daten liefern oder Daten in dem Objekt verändern.



Beim Aufruf von Methoden sind immer runde Klammern anzugeben, auch wenn es keine Parameter gibt. Ohne die runden Klammern erhält man Informationen über die Methode, man ruft aber nicht die Methode selbst auf.

Eigenschaften (Mitgliedsart Property)

Property Eigenschaften (Mitgliedsart Property) sind Datenelemente, die Informationen aus dem Objekt enthalten oder mit denen man Informationen an das Objekt übergeben kann, z.B. `MaxWorkingSet`.

Wichtig: In PowerShell 1.0 sah die Aussage von `Get-Member` noch etwas anders aus (siehe Bildschirmabbildung). Man sieht dort, dass es zu jedem Property zwei Methoden gibt, z.B. `get_MaxWorkingSet()` und `set_MaxWorkingSet()`. Die Ursache dafür liegt in den Interna des .NET Frameworks: Dort werden Properties (nicht aber sogenannte Fields, eine

andere Art von Eigenschaften) durch ein Methodenpaar abgebildet: eine Methode zum Auslesen der Daten (genannt „Get-Methode“ oder „Getter“), eine andere Methode zum Setzen der Daten (genannt „Set-Methode“ oder „Setter“). Einige Anfänger störte die „Aufblähung“ der Liste durch diese Optionen. Ab PowerShell 2.0 zeigte `Get-Member` die Getter-Methoden (get_) und Setter-Methoden (set_) nur noch an, wenn man den Parameter `-force` verwendet.

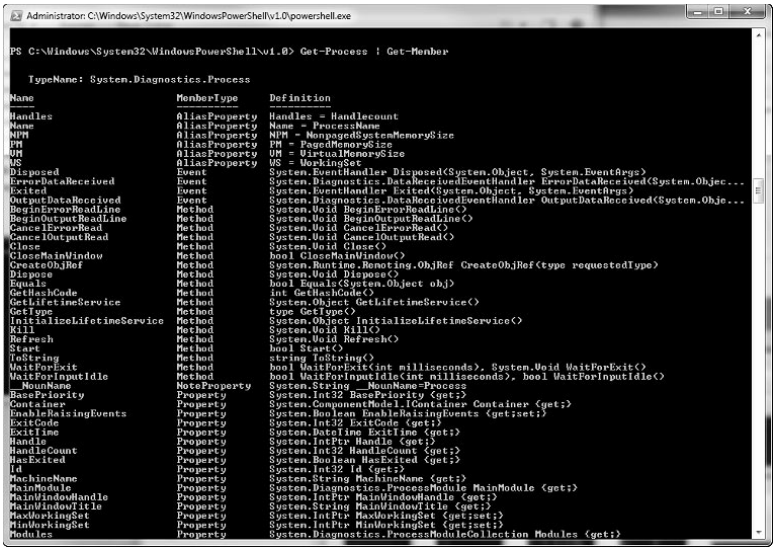


Abbildung 5.19
Anzeige der Getter und Setter in PowerShell 1.0

Fortgeschrittene Benutzer bevorzugen die Auflistung der Getter und Setter. Man kann erkennen, welche Aktionen auf einem Property möglich sind. Fehlt der Setter, kann die Eigenschaft nicht verändert werden (z.B. `StartTime` bei der Klasse `Process`). Fehlt der Getter, kann man die Eigenschaft nur setzen. Dafür gibt es kein Beispiel in der Klasse `Process`. Dieser Fall kommt auch viel seltener vor, wird aber z.B. bei Kennwörtern eingesetzt, die man nicht wiedergewinnen kann, weil sie nicht im Klartext, sondern nur als Hash-Wert abgespeichert werden.

Für den PowerShell-Nutzer bedeutet die Existenz von Gettern und Settern, dass er zwei Möglichkeiten hat, Daten abzurufen. Über die Eigenschaft (Property):

```
Get-Process | Where-Object { $_.name -eq "iexplore" } |
Foreach-Object { $_.MaxWorkingSet }
```

oder die entsprechende "Get"-Methode:

```
Get-Process | Where-Object { $_.name -eq "iexplore" } |
Foreach-Object { $_.get_MaxWorkingSet() }
```

Analog gibt es für das Schreiben die Option über die Eigenschaft:

```
Get-Process | Where-Object { $_.name -eq "iexplore" } |
Foreach-Object { $_.MaxWorkingSet = 1413120 }
```

oder die entsprechende "Set"-Methode:

```
Get-Process | Where-Object { $_.name -eq "iexplore" } |
Foreach-Object { $_.set_MaxWorkingSet(1413120) }
```

Eigenschaftssätze (PropertySet)

PropertySet

Eigenschaftssätze (PropertySet) sind eine Zusammenfassung einer Menge von Eigenschaften unter einem gemeinsamen Dach. Beispielsweise umfasst der Eigenschaftssatz `psRessources` alle Eigenschaften, die sich auf den Ressourcenverbrauch eines Prozesses beziehen. Dies ermöglicht es, dass man nicht alle Eigenschaften einzeln nennen muss, sondern schreiben kann:

```
Get-Process | Select-Object psRessources | Format-Table
```

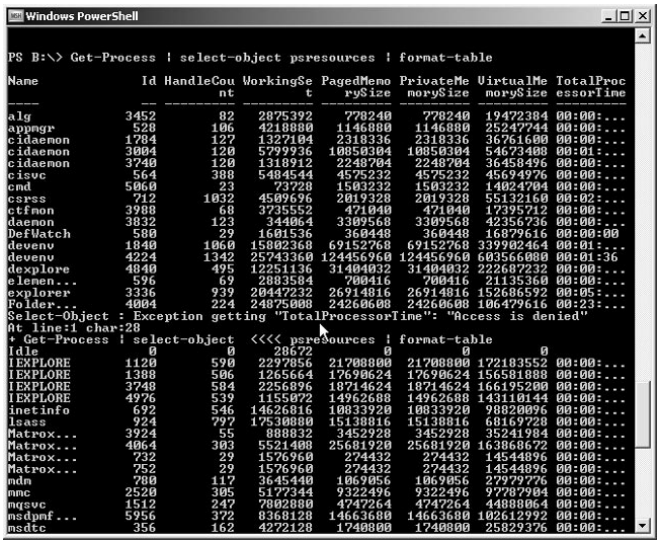


Abbildung 5.20 *Die PowerShell-Entwickler haben nicht an den Sonderstatus des Pseudoprozesses „Idle“ gedacht.*

Die Entwickler der PowerShell haben zwar an vieles gedacht, aber nicht an alles. Zum Beispiel führt der obige Befehl bei einem der Prozesse zu der Fehlermeldung „Access is denied“, den Pseudoprozess „Idle“ kann man nicht nach `TotalProcessorTime` fragen.

Die Eigenschaftssätze gibt es nicht im .NET Framework; sie sind eine Eigenart der PowerShell und definiert in der Datei *types.ps1xml* im Installationsordner der PowerShell.

```
<PropertySet>
    <Name>PSConfiguration</Name>
    <ReferencedProperties>
        <Name>Name</Name>
        <Name>Id</Name>
        <Name>PriorityClass</Name>
        <Name>FileVersion</Name>
    </ReferencedProperties>
</PropertySet>
<PropertySet>
    <Name>PSResources</Name>
    <ReferencedProperties>
        <Name>Name</Name>
        <Name>Id</Name>
        <Name>Handlecount</Name>
        <Name>WorkingSet</Name>
        <Name>NonPagedMemorySize</Name>
        <Name>PagedMemorySize</Name>
        <Name>PrivateMemorySize</Name>
        <Name>VirtualMemorySize</Name>
        <Name>Threads.Count</Name>
        <Name>TotalProcessorTime</Name>
    </ReferencedProperties>
</PropertySet>
```

Abbildung 5.21 *Definition der Eigenschaftssätze für die Klasse System.Diagnostics.Process in types.ps1ml*

Notizeigenschaften (NoteProperty)

NoteProperties

Notizeigenschaften (NoteProperties) sind zusätzliche Datenelemente, die nicht der Datenquelle entstammen, sondern welche die PowerShell-Infrastruktur hinzugefügt hat. Im obigen Beispiel ist dies `__NounName`, der einen Kurznamen der Klasse liefert. Andere Klassen haben zahlreiche Notizeigenschaften. Notizeigenschaften gibt es nicht im .NET Framework; sie sind eine Eigenart der PowerShell.

Man kann einem Objekt zur Laufzeit ein NoteProperty hinzufügen, siehe *Kapitel 15 „Dynamische Objekte "*.

Skripteigenschaften (ScriptProperty)

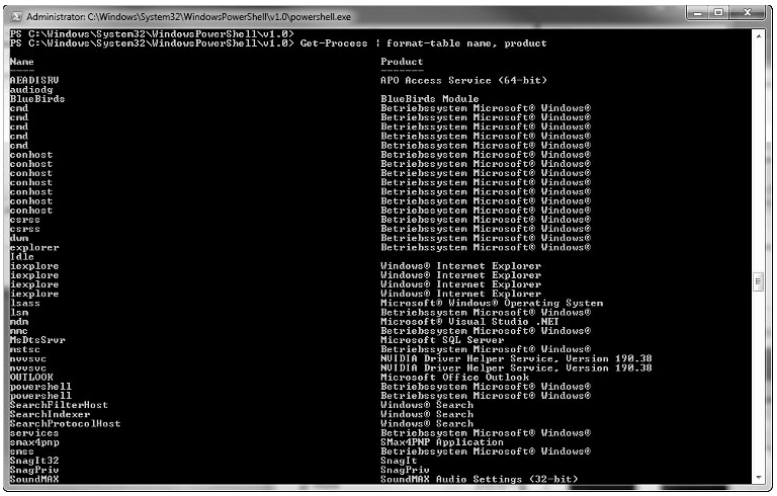
ScriptProperty

Eine **Skripteigenschaft (ScriptProperty)** ist eine berechnete Eigenschaft, also eine Information, die nicht im Objekt selbst gespeichert ist. Dabei muss die Berechnung nicht notwendigerweise eine mathematische Berechnung sein; es kann sich auch um den Zugriff auf die Eigenschaften eines untergeordneten Objekts handeln. Der Befehl

```
Get-Process | Select-Object name, product
```

listet alle Prozesse mit den Produkten auf, zu denen der Prozess gehört (siehe *Abbildung 5.22*). Dies ist gut zu wissen, wenn man auf seinem System einen Prozess sieht, den man nicht kennt und von dem man befürchtet, dass es sich um einen Schädling handeln könnte.

Abbildung 5.22 *Auflistung der berechneten Eigenschaft „Product“*



Die Information über das Produkt steht nicht in dem Prozess (Windows listet diese Information im Taskmanager ja auch nicht auf), aber in der Datei, die den Programmcode für den Prozess enthält. Das .NET Framework bietet über die `MainModule.FileversionInfo.ProductName` einen Zugang zu dieser Information. Anstelle des Befehls

```
Get-Process | Select-Object name,
Mainmodule.FileVersionInfo.ProductName
```

bietet Microsoft durch die Skripteigenschaft eine Abkürzung an. Diese Abkürzung ist definiert in der Datei *types.ps1xml* im Installationsordner der PowerShell.

Abbildung 5.23 *Definition einer Skripteigenschaft in der types.ps1xml*

```
<ScriptProperty>
    <Name>Product</Name>
    <GetScriptBlock>$this.Mainmodule.FileVersionInfo.ProductName</GetScriptBlock>
</ScriptProperty>
```

Skripteigenschaften gibt es nicht im .NET Framework; sie sind eine Eigenart der PowerShell.

Man kann einem Objekt zur Laufzeit eine Skripteigenschaft hinzufügen, siehe *Kapitel 15 „Dynamische Objekte “*.

Codeeigenschaften (Code Property)

CodeProperty

Eine **Codeeigenschaft (CodeProperty)** entspricht einer Script Property, allerdings ist der Programmcode nicht als Skript in der PowerShell-Sprache, sondern als .NET-Programmcode hinterlegt.

Aliaseigenschaft (AliasProperty)

AliasProperty

Eine **Aliaseigenschaft** (AliasProperty) ist eine verkürzte Schreibweise für ein Property. Dahinter steckt keine Berechnung, sondern nur eine Verkürzung des Namens. Beispielsweise ist `WS` eine Abkürzung für `WorkingSet`. Auch die Aliaseigenschaften sind in der Datei *types.ps1xml* im Installationsordner der PowerShell definiert. Aliaseigenschaften sind ebenfalls eine PowerShell-Eigenart.

Weitere Informationen zu Get-Member

Die Ausgabe von `Get-Member` kann man verkürzen, indem man nur eine bestimmte Art von Mitgliedern ausgeben lässt. Diese erreicht man über den Parameter `-Membertype` (kurz: `-m`). Der folgende Befehl listet nur die Properties auf:

```
Get-Process | Get-Member -Membertype Properties
```

Außerdem ist eine Filterung beim Namen möglich:

```
Get-Process  | Get-Member *set*
```

Der obige Befehl listet nur solche Mitglieder der Klasse `Process` auf, deren Name das Wort „set“ enthält.

Die Aliaseigenschaften gibt es nicht im .NET Framework; sie sind eine Eigenart der PowerShell.

Hintergrundwissen: Extended Type System (ETS)

Wie bereits dargestellt, zeigt die PowerShell für viele .NET-Objekte mehr Mitglieder an, als eigentlich in der Klasse definiert sind. In einigen Fällen werden aber auch Mitglieder ausgeblendet. In beiden Fällen kommt das Extended Type System (ETS) zum Einsatz.

ETS

Die Ergänzung von Mitgliedern per ETS wird verwendet, um bei einigen .NET-Klassen, die Metaklassen für die eigentlichen Daten sind (z.B. `ManagementObject` für WMI-Objekte, `ManagementClass` für WMI-Klassen, `DirectoryEntry` für Einträge in Verzeichnisdiensten und `DataRow` für Datenbankzeilen), die Daten direkt ohne Umweg dem PowerShell-Nutzer zur Verfügung zu stellen.

Mitglieder werden ausgeblendet, wenn sie in der PowerShell nicht nutzbar sind oder es bessere Alternativen durch die Ergänzungen gibt.

In der Dokumentation nimmt das PowerShell-Entwicklungsteam dazu wie folgt Stellung: „Some .NET Object members are inconsistently named, provide an insufficient set of public members, or provide insufficient capability. ETS resolves this issue by introducing the ability to extend the .NET object with additional members.“ [MSDN04] Dies heißt im Klartext, dass das PowerShell-Team mit der Arbeit des Entwicklungsteams der .NET-Klassenbibliothek nicht ganz zufrieden ist.

PSObject Das Extended Type System (ETS) verpackt grundsätzlich jedes Objekt, das von einem Commandlet in die Pipeline gelegt wird, in ein PowerShell-Objekt des Typs `PSObject`. Die Implementierung der Klasse `PSObject` entscheidet dann, was für die folgenden Commandlets und Befehle sichtbar ist.

Diese Entscheidung wird beeinflusst durch verschiedene Instrumente:

- PowerShell-Objektadapter, die für bestimmte Typen wie `Management-Object`, `ManagementClass`, `DirectoryEntry` und `DataRow` implementiert wurden,
- die Deklarationen in der *types.ps1xml*-Datei,
- in den Commandlets hinzugefügte Mitglieder,
- die dem Commandlet `Add-Member` hinzugefügten Mitglieder.

5.6 Filtern

Where-Object Nicht immer will man alle Objekte weiterverarbeiten, die ein Commandlet liefert. Einschränkungskriterien sind Bedingungen (z.B. nur Prozesse, bei denen der Speicherbedarf größer ist als 10000000 Byte) oder die Position (z.B. nur die fünf Prozesse mit dem größten Speicherbedarf). Zur Einschränkung verwendet man das Commandlet `Where-Object` (Alias `where`).

```
Get-Process | Where-Object {$_.ws -gt 10000000 }
```

Select-Object Einschränkungen über die Position definiert man mit dem `Select-Object` (in dem nachfolgenden Befehl für das oben genannte Beispiel ist zusätzlich noch eine Sortierung eingebaut, damit die Ausgabe einen Sinn ergibt):

```
Get-Process | Sort-Object ws -desc | Select-Object -first 5
```

Analog dazu sind die kleinsten Speicherfresser zu ermitteln mit:

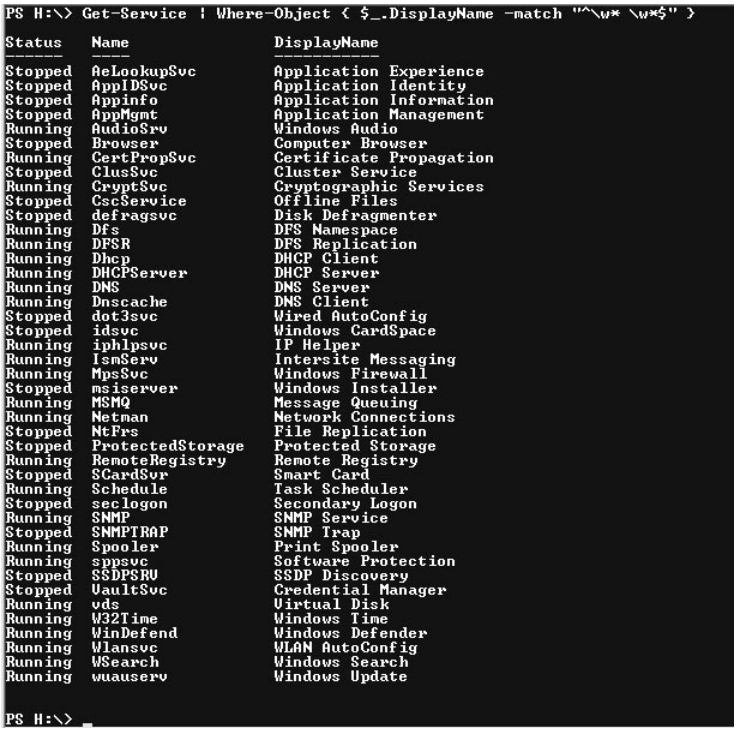
```
Get-Process | Sort-Object ws -desc | Select-Object -last 5
```

Vergleiche Etwas gewöhnungsbedürftig ist die Schreibweise der Vergleichsoperatoren: Statt >= schreibt man `-ge` (siehe Tabelle). Die Nutzung regulärer Ausdrücke ist möglich mit dem Operator `-Match`.

Dazu zwei Beispiele:

1. Der folgende Ausdruck listet alle Systemdienste, deren Beschreibung aus zwei durch ein Leerzeichen getrennten Wörtern besteht.

```
Get-Service | Where-Object { $_.DisplayName -match "^\w* \w*$" }
```



```
PS H:\> Get-Service | Where-Object { $_.DisplayName -match "^\w* \w*$" }

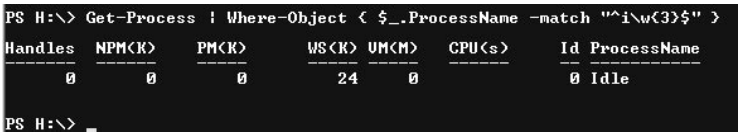
Status   Name               DisplayName
------   ----               -----------
Stopped  AeLookupSvc        Application Experience
Stopped  AppIDSvc           Application Identity
Stopped  Appinfo            Application Information
Stopped  AppMgmt            Application Management
Running  AudioSrv           Windows Audio
Stopped  Browser            Computer Browser
Running  CertPropSvc        Certificate Propagation
Stopped  ClusSvc            Cluster Service
Running  CryptSvc           Cryptographic Services
Stopped  CscService         Offline Files
Stopped  defragsvc          Disk Defragmenter
Running  Dfs                DFS Namespace
Running  DFSR               DFS Replication
Running  Dhcp               DHCP Client
Running  DHCPServer         DHCP Server
Running  DNS                DNS Server
Running  Dnscache           DNS Client
Stopped  dot3svc            Wired AutoConfig
Stopped  idsvc              Windows CardSpace
Running  iphlpsvc           IP Helper
Running  IsmServ            Intersite Messaging
Running  MpsSvc             Windows Firewall
Stopped  msiserver          Windows Installer
Running  MSMQ               Message Queuing
Running  Netman             Network Connections
Stopped  NtFrs              File Replication
Stopped  ProtectedStorage   Protected Storage
Running  RemoteRegistry     Remote Registry
Stopped  SCardSvr           Smart Card
Running  Schedule           Task Scheduler
Stopped  seclogon           Secondary Logon
Running  SNMP               SNMP Service
Stopped  SNMPTRAP           SNMP Trap
Running  Spooler            Print Spooler
Running  sppsvc             Software Protection
Stopped  SSDPSRV            SSDP Discovery
Stopped  VaultSvc           Credential Manager
Running  vds                Virtual Disk
Running  W32Time            Windows Time
Running  WinDefend          Windows Defender
Running  Wlansvc            WLAN AutoConfig
Running  WSearch            Windows Search
Running  wuauserv           Windows Update


PS H:\> _
```

Abbildung 5.24
Ausgabe zu obigem Beispiel

2. Der folgende Ausdruck listet alle Prozesse, die Namen mit einem "i" starten und danach aus drei Buchstaben bestehen.

```
Get-Process | Where-Object { $_.ProcessName -match "^i\w{3}$" }
```



```
PS H:\> Get-Process | Where-Object { $_.ProcessName -match "^i\w{3}$" }

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
      0       0        0         24     0               0 Idle


PS H:\> _
```

Abbildung 5.25
Ausgabe zu obigem Beispiel

Tabelle 5.1 *Vergleichsoperatoren der PowerShell*

Vergleich unter Berücksichtigung der Groß-/Kleinschreibung	Vergleich unter Ignorierung der Groß-/Kleinschreibung	Bedeutung
`-lt`	`-ilt`	Kleiner
`-le`	`-ile`	Kleiner oder gleich
`-gt`	`-igt`	Größer
`-ge`	`-ige`	Größer oder gleich
`-eq`	`-ieq`	Gleich
`-ne`	`-ine`	Nicht gleich
`-like`	`-ilike`	Ähnlichkeit zwischen Zeichenketten, Einsatz von Platzhaltern (* und ?) möglich
`-notlike`	`-inotlike`	Keine Ähnlichkeit zwischen Zeichenketten, Einsatz von Platzhaltern (* und ?) möglich
`-match`		Vergleich mit regulärem Ausdruck
`-notmatch`		Stimmt nicht mit regulärem Ausdruck überein
`-is`		Typvergleich

Tabelle 5.2 *Logische Operatoren in der PowerShell-Sprache*

Logischer Operator	Bedeutung
`-not` oder `!`	Nicht
`-and`	Und
`-or`	Oder

5.7 Zusammenfassung von Pipeline-Inhalten

$ Die Menge der Objekte in der Pipeline kann heterogen sein, d.h. verschiedenen Klassen angehören. Dies ist zum Beispiel automatisch der Fall, wenn man `Get-ChildItem` im Dateisystem ausführt: Die Ergebnismenge enthält sowohl `FileInfo`- als auch `DirectoryInfo`-Objekte.

Man kann auch zwei Befehle, die beide Objekte in die Pipeline senden, zusammenfassen, so dass der Inhalt in einer Pipeline wie folgt aussieht:

```
$( Get-Process  ; Get-Service )
```

Dies ist aber nur sinnvoll, wenn die nachfolgenden Befehle in der Pipeline korrekt mit heterogenen Pipeline-Inhalten umgehen können. Die Standardausgabe kann dies. In anderen Fällen bedingt der Typ des ersten Objekts in der Pipeline die Art der Weiterverarbeitung (z.B. bei `Export-Csv`).

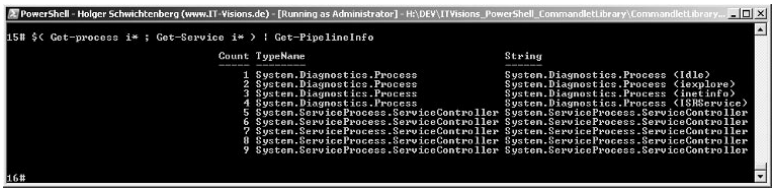


Abbildung 5.26 *Anwendung von GetPipelineInfo auf eine heterogene Pipeline*

5.8 „Kastrierung“ von Objekten in der Pipeline

Die Analyse des Pipeline-Inhalts zeigt, dass es oftmals sehr viele Mitglieder in den Objekten in der Pipeline gibt. In der Regel braucht man aber nur wenige. Nicht nur aus Gründen der Leistung und Speicherschonung, sondern auch in Bezug auf die Übersichtlichkeit lohnt es sich, die Objekte in der Pipeline zu „kastrieren“.

Select-Object

Mit dem Befehl `Select-Object` (Alias: `Select`) kann ein Objekt in der Pipeline „kastriert“ werden, d.h., (fast) alle Mitglieder des Objekts werden aus der Pipeline entfernt, mit Ausnahme der hinter `Select-Object` genannten Mitglieder.

Beispiel:

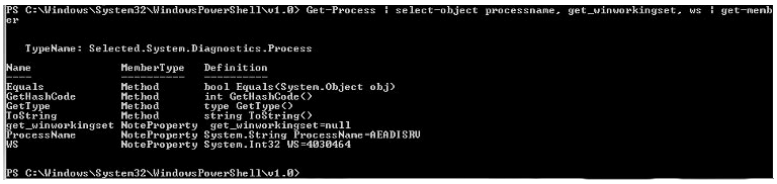
```
Get-Process | Select-Object processname, get_minworkingset, ws |
Get-Member
```

lässt von den `Process`-Objekten in der Pipeline nur die Mitglieder `processname` (Eigenschaft), `get_minworkingset` (Methode) und `workingset` (Alias) übrig (siehe *Abbildung 5.27*). Wie die Abbildung zeigt, ist das „Kastrieren“ mit drei Wermutstropfen verbunden:

- `Get-Member` zeigt nicht mehr den tatsächlichen Klassennamen an, sondern `PSCustomObject`, eine Klasse der PowerShell.
- Alle Mitglieder sind zu Notizeigenschaften degradiert.

Dass es neben den drei gewünschten Mitgliedern noch vier weitere in der Liste gibt, ist auch einfach erklärbar: Jedes, wirklich jedes .NET-Objekt hat diese vier Methoden, weil diese von der Basisklasse `System.Object` an jede .NET-Klasse vererbt und damit an jedes .NET-Objekt weitergegeben werden.

Abbildung 5.27
Wirkung der Anwendung von Select-Object





Mit dem Parameter `-exclude` kann man in `Select-Object` auch Mitglieder einzeln ausschließen.

5.9 Sortieren

Sort-Object

Mit `Sort-Object` (Alias `sort`) sortiert man die Objekte in der Pipeline nach den anzugebenden Eigenschaften. Die Standardsortierrichtung ist aufsteigend.

Der folgende Befehl sortiert die Prozesse absteigend nach ihrem Speicherverbrauch:

```
Get-Process | sort ws -desc
```

5.10 Gruppierung

Group-Object

Mit `Group-Object` kann man Objekte in der Pipeline nach Eigenschaften gruppieren.

Mit dem folgenden Befehl ermittelt man, wie viele Systemdienste laufen und wie viele gestoppt sind:

```
PS B:\Scripte> Get-Service | Group-Object status

Count Name                      Group
----- ----                      -----
   64 Running                   {AeLookupSvc, ALG, AppMgmt,
appmgr...}
   54 Stopped                   {Alerter, aspnet_state, ClipSrv,
clr_optimiz...
```

Das zweite Beispiel gruppiert die Dateien im *System32*-Verzeichnis nach Dateierweiterung und sortiert die Gruppierung dann absteigend nach Anzahl der Dateien in jeder Gruppe.

```
Get-ChildItem c:\windows\system32 | Group-Object extension |
Sort-Object count -desc
```

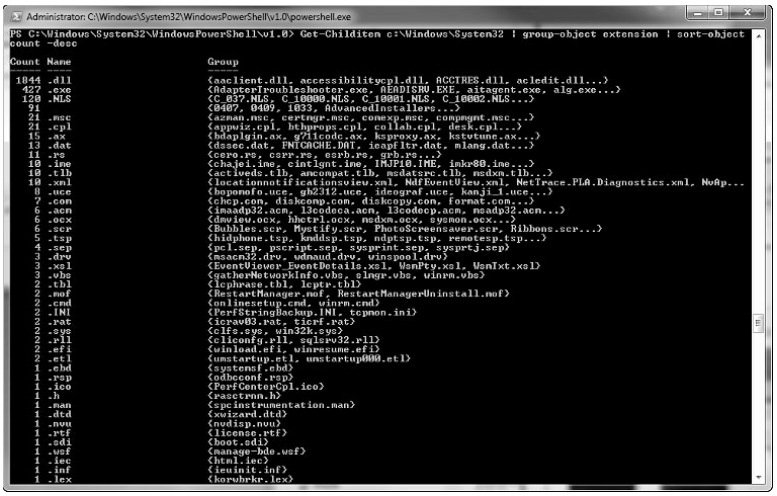


Abbildung 5.28
Einsatz von Group-Object und Sort-Object



Wenn es nur darum geht, die Gruppen zu ermitteln und nicht die Häufigkeit der Gruppenelemente, dann kann man auch `Select-Object` mit dem Parameter `-unique` zum Gruppieren einsetzen:

```
Get-ChildItem | Select-Object extension -Unique
```



Man kann bei `Group-Object` auch einen Ausdruck angeben, der wahr oder falsch liefert, und dadurch zwei Gruppen bilden. Beispiel:

```
Get-Childitem c:\Windows | Where { !$_.PsIsContainer } |
Group-Object { $_.Length -gt 1MB}
```

teilt alle Dateien im aktuellen Verzeichnis in zwei Gruppen ein: solche, die größer als 1 MByte sind, und solche, die es nicht sind (Verzeichnisse werden bereits vorher ausgeschlossen, auch wenn dies nicht erforderlich wäre, da sie die Größe 0 besitzen).



Abbildung 5.29*: Ergebnis des obigen Befehls (Zahlen können in Abhängigkeit vom Betriebssystem abweichen)*

5.11 Berechnungen

`Measure-Object` führt verschiedene Berechnungen (Anzahl, Durchschnitt, Summe, Minimum, Maximum) für Objekte in der Pipeline aus. Dabei sollte man die Eigenschaft nennen, über welche die Berechnung ausgeführt werden soll, da die erste Eigenschaft häufig ein Text ist, den man nicht mathematisch verarbeiten kann.

Beispiel: Informationen über die Dateien in *c:\Windows*

```
Get-ChildItem c:\windows | Measure-Object -Property length -min -
max -average -sum
```



Abbildung 5.30 *Beispiel für den Einsatz von Measure-Object*

5.12 Zwischenschritte in der Pipeline mit Variablen

PowerShell-Variablen

Ein Befehl mit Pipeline kann beliebig lang und damit auch beliebig komplex werden. Wenn der Befehl unübersichtlich wird oder man Zwischenschritte genauer betrachten möchte, bietet es sich an, den Inhalt der Pipeline zwischenzuspeichern. Die PowerShell ermöglicht es, den Inhalt der Pipeline in Variablen abzulegen. Variablen werden durch ein vorangestelltes Dollarzeichen ($) gekennzeichnet. Anstelle von

```
Get-Process  | Where-Object {$_.name -eq "iexplore"} | Foreach-
Object { $_.ws }
```

kann man die folgenden Befehle nacheinander in getrennte Zeilen eingeben:

```
$x = Get-Process
$y = $x | Where-Object {$_.name -eq "iexplore"}
$y | Foreach-Object { $_.ws }
```

Das Ergebnis ist in beiden Fällen gleich.

Der Zugriff auf Variablen, die keinen Inhalt haben, führt so lange nicht zum Fehler, wie man später in der Pipeline keine Commandlets verwendet, die unbedingt Objekte in der Pipeline erwarten.



Abbildung 5.31 *Zugriff auf Variablen ohne Inhalt*



Wenn ein Pipeline-Befehl keinen Inhalt liefert, dann erhält die Variable den Wert `$null`, der für „kein Wert“ steht.

Beispiel:

```
$x = Get-Service x*
```

Die Ausgabe für `$null` ist nichts.

5.13 Verzweigungen in der Pipeline

Tee-Object

Manchmal möchte man innerhalb einer Pipeline das Ergebnis nicht nur in der Pipeline weiterreichen, sondern auch in einer Variablen oder im Dateisystem zwischenspeichern. Der Verzweigung innerhalb der Pipeline dient das Commandlet `Tee-Object`, wobei hier das „Tee“ für „verzweigen“ steht. `Tee-Object` reicht den Inhalt der Pipeline unverändert zum nächsten Commandlet weiter, bietet aber an, den Inhalt der Pipeline wahlweise zusätzlich in einer Variablen oder im Dateisystem abzulegen.

Der folgende Pipeline-Befehl verwendet `Tee-Object` gleich zweimal für beide Anwendungsfälle:

```
Get-Service | Tee-Object -var a | Where-Object { $_.Status -eq
"Running" } | Tee-Object -filepath g:\dienste.txt
```

Die erste Verwendung von `Tee-Object` speichert die Liste der Dienste-Objekte in der Variablen `$a` und gibt die Objekte aber gleichzeitig weiter in die Pipeline.

Die zweite Verwendung speichert die Liste der laufenden Dienste in der Textdatei `g:\dienste.txt` und gibt sie zusätzlich an die Standardausgabe aus.

Nach der Ausführung des Befehls steht in der Variablen `$a` eine Liste aller Dienste und in der Textdatei *dienste.txt* eine Liste der laufenden Dienste.



Bitte beachten Sie, dass man bei `Tee-Object` beim Parameter `-variable` den Namen der Variablen ohne den üblichen Variablenkennzeichner `"$"` angeben muss.

5.14 Vergleiche zwischen Objekten

Compare-Object

Mit `Compare-Object` kann man den Inhalt von zwei Pipelines vergleichen.

Mit der folgenden Befehlsfolge werden alle zwischenzeitlich neu gestarteten Prozesse ausgegeben:

```
$ProzesseVorher = Get-Process
# Hier einen Prozess starten
$ProzesseNacher = Get-Process
Compare-Object $ProzesseVorher $ProzesseNacher
```

Abbildung 5.32
Vergleich von zwei Pipelines



5.15 Praxisbeispiele

Dieses Kapitel enthält einige Beispiele für die Anwendung von Pipelining und Ausgabebefehlen:

Pipelining-Beispiele

- Beende alle Prozesse durch Aufruf der Methode `Kill()`, die „iexplore“ heißen, wobei die Groß-/Kleinschreibung des Prozessnamens irrelevant ist.

  ```
  Get-Process | where { $_.processname -ieq "iexplore" } |
  foreach { $_.Kill() }
  ```

- Sortiere die Prozesse, die das Wort „iexplore“ im Namen tragen, gemäß ihrer CPU-Nutzung und beende den Prozess, der in der aufsteigenden Liste der CPU-Nutzung am weitesten unten steht (also am meisten Rechenleistung verbraucht).

  ```
  Get-Process  | where { $_.processname -ilike "*iexplore*" } |
  Sort-Object -p cpu | Select-Object -last 1 | foreach { $_.Kill() }
  ```

- Gib die Summe der Speichernutzung aller Prozesse aus.

  ```
  ps | Measure-Object workingset
  ```

- Gruppiere die Einträge im System-Ereignisprotokoll nach Benutzernamen.

```
Get-EventLog -logname system | Group-Object username
```

- Zeige die letzten zehn Einträge im System-Ereignisprotokoll.

```
Get-EventLog -logname system | Select-Object -last 10
```

- Zeige für die letzten zehn Einträge im System-Ereignisprotokoll die Quelle an.

```
Get-EventLog -logname system | Select-Object -first 10 |
Select-Object -p source
```

- Importiere die Textdatei `test.txt`, wobei die Textdatei als eine CSV-Datei mit dem Semikolon als Trennzeichen zu interpretieren ist und die erste Zeile die Spaltennamen enthalten muss. Zeige daraus die Spalten *ID* und *Url*.

```
Import-Csv d:\_work\test.txt -delimiter ";" | Select-Object -p
ID,Url
```

- Ermittle aus dem Verzeichnis `System32` alle Dateien, die mit dem Buchstaben „a“ beginnen. Beschränke die Menge auf diejenigen Dateien, die größer als 40.000 Byte sind, und gruppiere die Ergebnismenge nach Dateinamenerweiterungen. Sortiere die gruppierte Menge nach dem Namen der Dateierweiterung.

```
Get-ChildItem c:\windows\system32 -filter a*.* | Where-Object
{$_.Length -gt  40000} | Group-Object Extension | Sort-Object
name | Format-Table
```

- Ermittle aus dem Verzeichnis `System32` alle Dateien, die mit dem Buchstaben „b“ beginnen. Beschränke die Menge auf diejenigen Dateien, die größer als 40.000 Byte sind, und gruppiere die Ergebnismenge nach Dateierweiterungen. Sortiere die Gruppen nach der Anzahl der Einträge absteigend und beschränke die Menge auf das oberste Element. Gib für alle Mitglieder dieser Gruppe die Attribute `Name` und `Length` aus und passe die Spaltenbreite automatisch an.

```
Get-ChildItem c:\windows\system32 -filter b*.* | Where-Object
{$_.Length -gt 40000} | Group-Object Extension | Sort-Object
count -desc | Select-Object -first 1 | Select-Object group |
foreach {$_.group} | Select-Object name,length | Format-Table -
autosize
```

6 PowerShell-Skripte

Dieses Kapitel behandelt die Erstellung von PowerShell-Skripten aus Commandlets und der PowerShell-Skriptsprache. Beide Formen können in einem Skript beliebig gemischt werden. Zunächst werden der allgemeine Aufbau und der Start der Skripte erläutert. Die Befehle der Skriptsprache folgen im nächsten Kapitel.

6.1 Skriptdateien

.ps1

Befehlsabfolgen (Commandlets und/oder Elemente der Skriptsprache) können als PowerShell-Skripte im Dateisystem abgelegt und später (unbeaufsichtigt) ausgeführt werden. Diese Skripte sind reine Textdateien und haben die Dateinamenerweiterung *.ps1*. Die Zahl 1 steht dabei ursprünglich für die Version 1.0 der PowerShell. Microsoft hat in Hinblick auf die Langlebigkeit vieler Skripte vorgesehen, dass verschiedene Versionen der PowerShell auf einem System koexistieren können.



.ps1 gilt auch noch für PowerShell 2.0. Man kann .ps1 nun also als „erste Version des PowerShell-Dateiformats“ verstehen.

Beispiel

Beispiel

Das folgende Listing zeigt ein Skript, das verschiedene Informationen über den Computer ausgibt. Die Informationen werden aus den Commandlets `Get-Date`, `Get-Service` und `Get-Process` ermittelt. Dabei werden nur jeweils einzelne Werte durch den Aufruf von Methoden `ToShortDateString()` bzw. `ToLongTimeString()` sowie dem `Property Count` extrahiert. Zudem gibt es statische Ausgaben (Zeichenkettenliterale), die durch das Pluszeichen (+) mit den Ergebnissen der Commandlets verbunden werden. Die Raute (#) leitet Kommentare ein, die nicht ausgegeben werden.

Die einzelnen Befehle sind jeweils durch Zeilenumbrüche getrennt. Alternative Möglichkeiten der Befehlstrennung lernen Sie in *Kapitel 7 „PowerShell-Skriptsprache“* kennen.

```
# Mein erstes Skript
# Beispiel aus dem Buch "Windows PowerShell 2.0 - Das
Praxishandbuch"
"Informationen über diesen Computer:"
```

```
"Datum:" + (Get-Date).ToShortDateString()
"Zeit:" + (Get-Date).ToLongTimeString()
"Anzahl laufender Prozesse: " + (Get-Process).Count
"Anzahl gestarteter Dienste: " + (Get-Service | where { $_.Status -
eq "running" } ).Count
```

Editoren

Das Skript können Sie in einem beliebigen Texteditor (z.B. Notepad, Textpad) oder einem speziellen PowerShell-Editor (z.B. ISE, PowerShellPlus, PowerGUI) erfassen und im Dateisystem speichern. Details über die Editoren erfahren Sie in *Kapitel 11 „PowerShell-Werkzeuge“*.

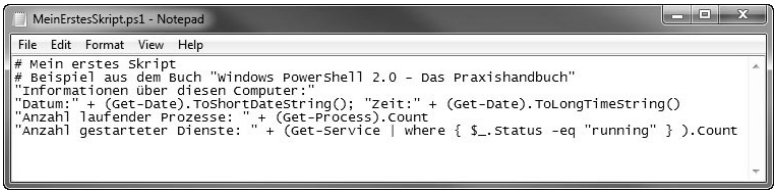


Abbildung 6.1 *Das Skript im Notepad*

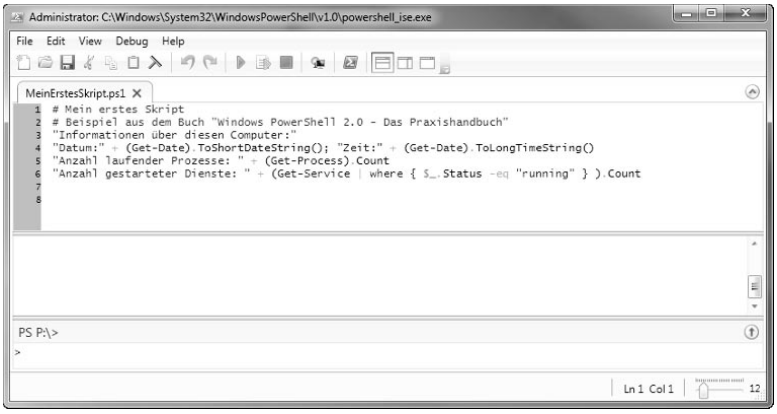


Abbildung 6.2 *Das Skript im „Integrated Scripting Environment“ (ISE), das Microsoft bei PowerShell 2.0 mitliefert*

6.2 Start eines Skripts

Skriptaufruf

Jeffrey Snover, der maßgebliche Architekt der Windows PowerShell, nannte als „Top-Sicherheitsfunktion“ der Windows PowerShell die Tatsache, dass man ein PowerShell-Skript nicht durch Doppelklick auf das Symbol in Windows starten kann. Grundsätzlich könnte man diese Startart definieren, sie ist aber nicht im Standardumfang der PowerShell-Installation enthalten. Ein PowerShell-Skript wird gestartet durch Eingabe des Namens mit oder ohne Dateierweiterung bzw. mit oder ohne ein vorangestelltes Commandlet in der PowerShell-Konsole.

`.\MeinErstesSkript.ps1` oder

`&.\MeinErstesSkript.ps1` oder

`Invoke-Expression .\MeinErstesSkript.ps1`

Wichtig ist, dass ein Skript immer eine Pfadangabe braucht, selbst wenn es nur ein „.\“ für den lokalen Pfad ist. Das „.ps1“ ist jeweils optional.

Bereits in *Kapitel 1 „Erste Schritte mit der Windows PowerShell“* haben Sie gelernt, dass die PowerShell im Standard gar keine Skriptausführung erlaubt. Wenn Sie den dort genannten Befehl `Set-ExecutionPolicy remotesigned` noch nicht ausgeführt haben, führen Sie ihn jetzt aus, um die Skriptausführung auf Ihrem System zu erlauben. Details dazu erfahren Sie in *Kapitel 6.6 „Sicherheitsfunktionen für PowerShell-Skripte“*.

Die folgende Bildschirmabbildung zeigt den Start des Skripts an der PowerShell-Konsole mit absolutem Pfad („P:\...“) und relativem Pfad („.\“).

```
PS P:\> P:\1_Basiswissen\Scripting\MeinErstesSkript.ps1
Informationen über diesen Computer:
Datum:23.01.2010
Zeit:13:27:27
Anzahl laufender Prozesse: 88
Anzahl gestarteter Dienste: 77
PS P:\> cd P:\1_Basiswissen\Scripting
PS P:\1_Basiswissen\Scripting> .\MeinErstesSkript.ps1
Informationen über diesen Computer:
Datum:23.01.2010
Zeit:13:27:39
Anzahl laufender Prozesse: 88
Anzahl gestarteter Dienste: 77
PS P:\1_Basiswissen\Scripting> _
```

Abbildung 6.3
Start des Skripts an der PowerShell-Konsole mit absolutem Pfad und relativem Pfad

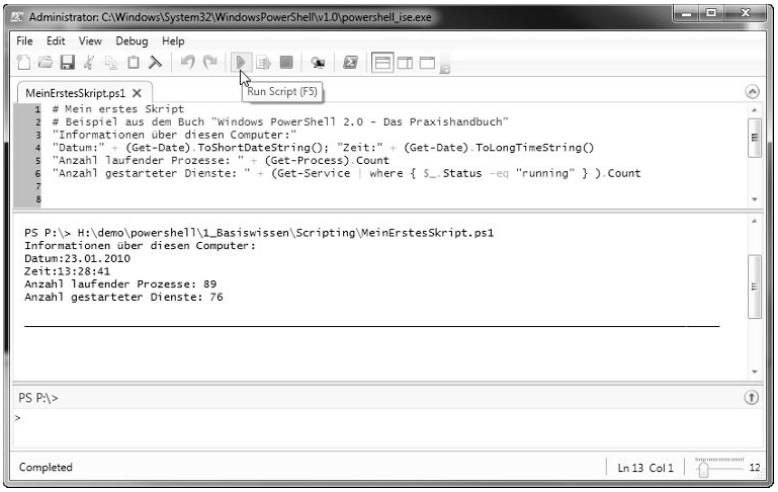


Abbildung 6.4
Start des Skripts in der ISE mit dem Symbolleistenbefehl „Run Script“ oder der Taste F5

Alternativ kann man ein PowerShell-Skript aus dem normalen Windows-Kommandozeilenfenster (cmd.exe) durch eine Verknüpfung aus dem Windows-Desktop oder als Anmeldeskript starten, indem man `powershell.exe` voranstellt:

```
powershell.exe p:\Skripte\MeinErstesSkript.ps1
```



Für PowerShell-Skripte gelten ab Windows Vista hinsichtlich der Benutzerkontensteuerung (User Account Control – UAC) die gleichen Einschränkungen und Lösungen wie für WSH-Skripte. Bitte lesen Sie für weitere Informationen das Kapitel über den Windows Script Host (WSH).

6.3 Aliase für Skripte verwenden

Set-Alias

Man kann Aliase auch für Skriptdateien inklusive Pfadangabe vergeben, z.B.

```
Set-Alias Get-ComputerInfo P:\1_Basiswissen\Scripting\
MeinErstesSkript.ps1
```

Danach kann man dieses Skript über `Get-ComputerInfo` aufrufen, als wäre es ein Commandlet.



Abbildung 6.5 *Setzen eines Alias für ein Skript*

6.4 Parameter für Skripte

Das oben gezeigte „MeinErstesSkript" fragt immer die Dienste und Prozesse des lokalen Systems ab. Schön wäre es, wenn man auch entfernte Computersysteme abfragen könnte. Sowohl `Get-Process` als auch `Get-Service` bieten dazu einen Parameter `-ComputerName`, bei dem man ein entferntes System angeben kann. Details zur Fernausführung (Remoting) erfahren Sie in *Teil 2 „PowerShell-Aufbauwissen"*. An dieser Stelle soll der Computername ein Parameter des Skripts werden.

Beim Aufruf eines Skripts kann man Parameter genauso übergeben wie beim Aufruf von Commandlets, d.h. ohne Klammern und getrennt durch Leerzeichen:

```
.\Skriptname.ps1 Parameter1 Parameter2 Parameter3
```

Ein Skript kann Parameter auf zwei Weisen verarbeiten:

- Über die Variable `$args`, die ein Array (Liste) der Parameter enthält. Die Zählung beginnt bei 0. Der erste Parameter steht in `$args[0]`, der zweite in `$args[1]`, der dritte in `$args[2]` usw.

- Durch eine explizite Deklaration einer Parameterliste mit Zuweisung von Variablennamen und optional Datentypen. Die Parameterliste definiert man am Beginn des Skripts mit `param($name1, $name2, $name3, usw.)` oder `param([Typ] $name1,[Typ] $name2,[Typ] $name3, usw.)`. Im Skript kann man dann über `$name1` etc. Bezug auf die Werte nehmen.

Die folgenden Listings zeigen beide Varianten beim Beispiel von `Get-ComputerInfo.ps1`, einer modifizierten Form von `MeinErstesSkript.ps1`. Der Aufruf erfolgt mit:

```
.\Get-ComputerInfo.ps1 E21
```



Wenn Sie den Computernamen vergessen sollten, beschwert sich das Skript selbst nicht. Es findet keinen Parameter und setzt den Wert `$null`. `$null` (nicht zu verwechseln mit der Zahl 0!) steht für „kein Wert“. Das ist in Ordnung für das Skript, aber die Commandlets `Get-Process` und `Get-Service` mögen es nicht, wenn man als `-ComputerName $null` übergibt, und beschweren sich: „The argument is null or empty.“.

```
# Get-ComputerInfo
# Skript mit Parametern
# Beispiel aus dem Buch "Windows PowerShell 2.0 - Das
Praxishandbuch"
"Informationen über den Computer: " + $args[0]
"Datum:" + (Get-Date).ToShortDateString(); "Zeit:" + (Get-
Date).ToLongTimeString()
"Anzahl laufender Prozesse: " + (Get-Process -computername
$args[0]).Count
"Anzahl gestarteter Dienste: " + (Get-Service -computername
$args[0]| where { $_.Status -eq "running" } ).Count
```

***Listing 6.1**: Get-ComputerInfo.ps1 Variante 1 unter Verwendung von args[x]*

```
# Get-ComputerInfo
# Skript mit Parametern
# Beispiel aus dem Buch "Windows PowerShell 2.0 - Das
Praxishandbuch"
param([string] $Computer)
"Informationen über den Computer: " + $Computer
"Datum:" + (Get-Date).ToShortDateString(); "Zeit:" + (Get-
Date).ToLongTimeString()
"Anzahl laufender Prozesse: " + (Get-Process -computername
$Computer).Count
"Anzahl gestarteter Dienste: " + (Get-Service -computername
$Computer| where { $_.Status -eq "running" } ).Count
```

***Listing 6.2**: Get-ComputerInfo.ps1 Variante 2 unter Verwendung von param[$x]*

6.5 Skripte dauerhaft einbinden (Dot Sourcing)

Dot Sourcing

Als „Dot Sourcing" wird eine Möglichkeit bezeichnet, eine Skriptdatei aufzurufen und permanent in die aktuelle Instanz der PowerShell einzubinden. Der Unterschied zu den oben genannten Möglichkeiten ist, dass nach dem „Dot Sourcing" alle in dem Skript deklarierten Variablen und alle dort enthaltenen Funktionen der PowerShell-Konsole sowie alle nachfolgend aufgerufenen Skripte zur Verfügung stehen. „Dot Sourcing" ist also eine Möglichkeit, die Funktionalität der PowerShell zu erweitern.

„Dot Sourcing" wird durch einen vorangestellten Punkt mit Leerzeichen aktiviert:

```
. Softwareinventar.ps1
```

Es kann ein relativer oder ein absoluter Pfad angegeben werden.



Wenn in dem eingebundenen Skript „freie" Befehle enthalten sind, also Befehle, die nicht Teil einer Funktion sind, dann werden diese Befehle sofort ausgeführt. Deklarierte Funktionen werden nicht ausgeführt, stehen aber dem einbindenden Skript und den folgenden eingebundenen Skripten zur Verfügung.

Man kann mit „Dot Sourcing" auch Skripte in andere einbinden. Die Einbindung erfolgt auch hier mit dem Punktoperator (.).

Listing 6.3 *Beispiel für ein PowerShell-Skript, das nur besteht, um andere Skripte einzubinden und aufzurufen [Einsatzgebiete/Benutzer/Run Demos.ps1]*

```
# Demo User Management

. ("H:\demo\PowerShell\Benutzer\Localuser_Create.ps1")
. ("H:\demo\PowerShell\Benutzer\LocalGroup.ps1")
. ("H:\demo\PowerShell\Benutzer\Localuser_Delete.ps1")
```

6.6 Sicherheitsfunktionen für PowerShell-Skripte

Ausführungsrichtlinie

Die bisherige Active-Scripting-Architektur im Internet Explorer, in Outlook und im Windows Script Host (WSH) hatte mit Sicherheitsproblemen zu kämpfen. Laut der Dokumentation ist die PowerShell „in der Grundeinstellung eine sichere Umgebung" [MS02]. Dass nicht jedes beliebige Skript zur Ausführung gebracht werden kann, zeigt sich, wenn man versucht, die PowerShell-Konsole nicht nur interaktiv, sondern zum Start eines Skripts zu verwenden. Die Standardausführungsrichtlinie (Execution Policy) lässt überhaupt keine Skripte zu.

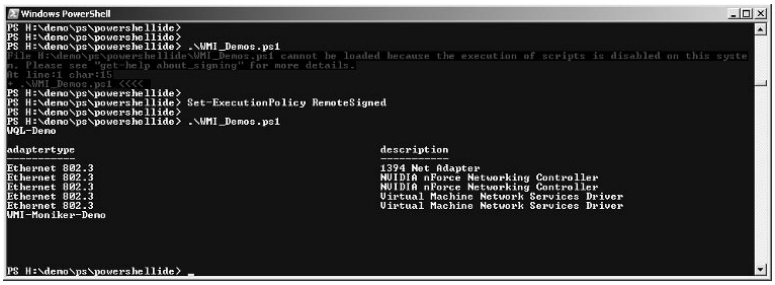


Abbildung 6.6
Die Skriptausführung muss man in der PowerShell erst explizit aktivieren.

Sicherheitsrichtlinien

Set-Executionpolicy

Ein Benutzer kann die Shell zunächst nur interaktiv verwenden, bis er die Ausführungsrichtlinie mit dem Commandlet `Set-Executionpolicy` auf eine niedrigere Sicherheitsstufe herabsetzt:

- Modus „Restricted“: Es werden keine Konfigurationsdateien geladen und keine Skripts ausgeführt. „Restricted“ ist der Standardwert.
- Modus „AllSigned“: Nur signierte Skripte starten und signierte Skripte von nichtvertrauten Quellen starten auf Nachfrage.
- Modus „RemoteSigned: Eine vertraute Signatur ist nur für Skripte aus dem Internet (via Browser, Outlook, Messenger) erforderlich; lokale Skripte starten auch ohne Signatur.
- Modus „Unrestricted“: Alle Skripte laufen.
- Modus „Bypass“: Keine Blockierung findet statt und es werden keine Warnungen oder Eingabeaufforderungen ausgegeben.
- Modus „Undefined“: Entfernt die gerade zugewiesene Ausführungsrichtlinie aus dem aktuellen Bereich. Mit diesem Parameter wird aber keine Ausführungsrichtlinie entfernt, die in einer Gruppenrichtlinie festgelegt wurde.

Neu in der PowerShell Version 2 ist die Möglichkeit, einen Gültigkeitsbereich für die Ausführungsrichtlinie festzulegen. Der Standardwert ist „LocalMachine“.

Gültige Werte sind:

- „Process“: Die Ausführungsrichtlinie wirkt sich nur auf den aktuellen Windows-PowerShell-Prozess aus.
- „CurrentUser“: Die Ausführungsrichtlinie wirkt sich nur auf den aktuellen Benutzer aus.
- „LocalMachine“: Die Ausführungsrichtlinie wirkt sich auf alle Benutzer des Computers aus.

Zum Entfernen einer Ausführungsrichtlinie aus einem bestimmten Bereich legen Sie die Ausführungsrichtlinie für diesen Bereich auf „Undefined“ fest. Hinweis: „Unrestricted“ sollte man nicht wählen, weil dies „bösen“ Skripten, die z.B. als E-Mail-Anhänge übermittelt

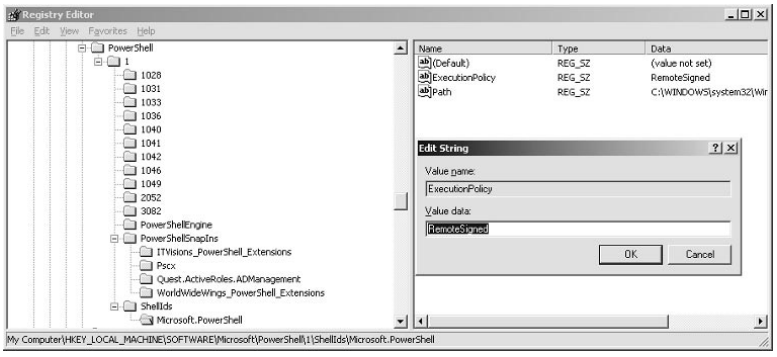
werden, die Tür öffnen würde. Wenn man sich nicht die Arbeit mit digitalen Signaturen machen will, ist die Option „RemoteSigned“ ein Kompromiss. Noch vor dem endgültigen Erscheinen der PowerShell gab es die ersten angeblichen PowerShell-Viren. Diese waren aber nur eine Bedrohung, wenn man sie explizit gestartet hat (siehe [MSSec01]).

Die Sicherheitsrichtlinie wird in der Registrierungsdatenbank auf System- oder Benutzerebene abgespeichert im Schlüssel: *HKEY_CURRENT_USER\Software\Microsoft\PowerShell\1\ShellIds\Microsoft.PowerShell\ExecutionPolicy* bzw. *HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\PowerShell\1\ShellIds\Microsoft.PowerShell\ExceutionPolicy.*



Bitte beachten Sie, dass durch die Speicherung in der Registrierungsdatenbank die Sicherheitsrichtlinie mit aktivierter Benutzerkontensteuerung nur geändert werden kann, wenn die PowerShell-Konsole unter vollen Rechten läuft.

Abbildung 6.7 *Persistierung der Sicherheitsrichtlinien in der Registrierungsdatenbank*



Skripte signieren

Set-AuthenticodeSignature

Im Unternehmenseinsatz sollte man digitale Signaturen verwenden. Zum Signieren von Skripten bietet die Shell das Commandlet `Set-AuthenticodeSignature`. Um ein Skript zu signieren, müssen Sie die folgenden Schritte ausführen:

- Wenn Sie kein digitales Zertifikat zum Signieren von Code besitzen, legen Sie sich ein Zertifikat an (z.B. mit dem Kommandozeilenwerkzeug *makecert.exe*).
- Lassen Sie sich Ihre eigenen Windows-Zertifikate in der PowerShell-Konsole auflisten:

  ```
  dir cert:/currentuser/my
  ```

- Ermitteln Sie die Position des Zertifikats, das Sie verwenden wollen, und speichern Sie dieses Zertifikat in einer Variablen (Achtung: Die Zählung beginnt bei 0!):

```
$cert = @(dir "cert:/currentuser/my/")[1]
```

- Signieren Sie das Skript:

```
Set-AuthenticodeSignature Softwareinventar3.ps1 $cert
```

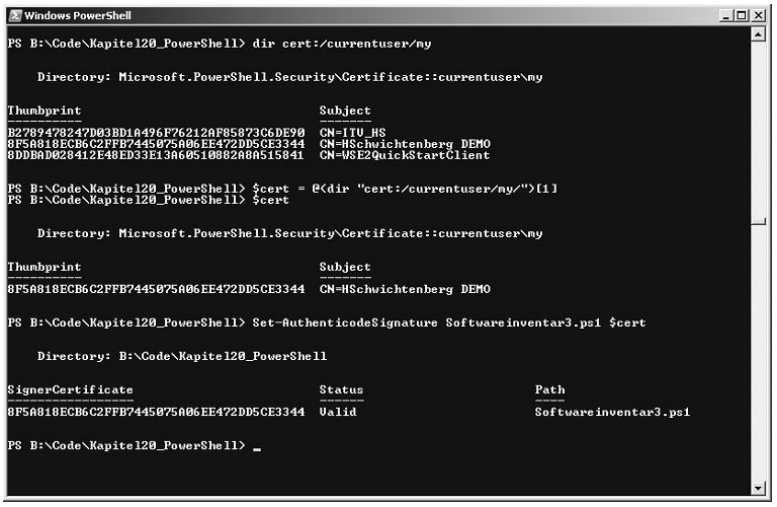


Abbildung 6.8
Signieren eines PowerShell-Skripts

Wenn Sie nun `Set-AuthenticodeSignature AllSigned` eingeben, sollte das von Ihnen signierte PowerShell-Skript laufen, alle anderen Skripte aber nicht.



Falls die PowerShell beim Start des Skripts noch einmal nachfragt, ob Sie das Skript wirklich laufen lassen wollen, dann bedeutet dies, dass das Skript zwar von jemandem signiert ist und Sie die Zertifizierungsstelle, die das Zertifikat ausgestellt hat, in Ihrer Stammzertifizierungsstelle kennen, aber dass Sie diesem Skriptautor noch nicht explizit vertrauen. Durch die Option *„Always Run"* würde der Skriptautor unter die „Vertrauenswürdigen Herausgeber" in die Zertifikatsverwaltung aufgenommen werden.



Abbildung 6.9
Nachfrage beim Skriptstart

6.7 Skripte anhalten

Start-Sleep

Ein PowerShell-Skript kann man eine Zeit lang anhalten. Die Zeit bemisst sich in Millisekunden oder Sekunden.

So wartet das Skript 10 Millisekunden:

```
Start-Sleep -m 10
```

So wartet das Skript 10 Sekunden:

```
Start-Sleep -s 10
```

7 PowerShell-Skriptsprache

PSL

Neben der Commandlet-Infrastruktur bietet die PowerShell auch eine eigene Skriptsprache zur Erstellung von Befehlsabfolgen im klassischen imperativen Programmierstil. Die PowerShell Language (PSL) kennt nicht nur Variablen, sondern auch übliche Programmkonstrukte wie Schleifen und Bedingungen. In der PSL kann man komplexe Befehlsabfolgen definieren und diese zu Skripten zusammenfassen.

Inspiration durch andere Sprachen

Microsoft greift dabei nicht auf eine bestehende Skriptsprache zurück, sondern hat sich für die Neuschöpfung nach eigenen Worten „inspirieren lassen" von Unix-Shell-Sprachen, PERL, PHP, Python und C#. Folglich verwendet die Sprache auch geschweifte Klammern; Semikola braucht man jedoch als Befehlstrenner nicht.



Bei der PowerShell-Skriptsprache gibt es in PowerShell 2.0 nur kleine Neuerungen gegenüber der Version 1.0, z.B. die Operatoren `-split` und `-join` und die Fehlerbehandlung mit Try/Catch alternativ zum bisher vorhandenen Trap. Auch mehrzeilige Kommentare mit <# ..#> sind neu.

7.1 Hilfe zur PowerShell-Skriptsprache

Get-Help

Die Sprachkonstrukte der PowerShell sind genauso wie die PowerShell-Commandlets in einfachen, rein textbasierten Hilfedokumenten erklärt, die mit der PowerShell installiert werden. Die Hilfedokumente zu den Sprachkonstrukten beginnen mit dem Wort „About", z.B. liefert der Befehl

```
Get-Help About_for
```

Hilfe zur for-Schleife.

Der Befehl

```
Get-Help About
```

zeigt eine Liste aller „About"-Dokumente.

7.2 Befehlstrennung

Pipe und Semikolon

Jede Zeile ist ein Befehl. Ein Befehl kann aus mehreren Commandlets bestehen, die durch das „Pipe-Symbol" (|) getrennt sind. Man kann mehrere Befehle durch ein Semikolon (;) getrennt in eine Zeile setzen. Man kann Semikola auch wie in C++ und C# am Ende jeder Zeile verwenden – man muss es aber nicht.

```
Get-Process ; Get-Service
```

Wenn sich ein Befehl über mehrere Zeilen erstrecken soll, kann man am Ende einer Zeile mit einem Hochkomma (`) bewirken, dass die nächste Zeile mit zum Befehl hinzugerechnet wird.

```
Get-Process | `
Format-List
```

7.3 Kommentare

Grundsätzlich sollte man bei der Programmierung dem eigentlichen Programmcode zusätzliche Kommentare hinzufügen, um das Programm für sich selbst und andere verständlich zu machen. Ein gutes Skript zeichnet sich dadurch aus, dass es gut lesbar ist, dass der Code also entsprechend kommentiert wurde. Zwar kann man anhand des Programmcodes oft schon sehen, was innerhalb eines Skripts geschieht, allerdings wird diese Lesbarkeit durch Kommentare wesentlich verbessert.

Kommentare werden in der PowerShell-Skriptsprache durch eine Raute # gekennzeichnet. Kommentare werden von der PowerShell komplett ignoriert. Hier kann jeder beliebige Text stehen.

```
# Ein beliebiger Kommentar
```



Die Raute gilt immer nur für den Rest der Zeile. Man kann nach einem Befehl einen Kommentar schreiben, z.B.

```
$s = Get-Service | where { $_.Status -eq "running" } # holt die
gestarteten Dienste
```

Aber man kann nicht in die gleiche Zeile nach der Raute noch einen Befehl schreiben, der ausgeführt werden soll.

```
$s = Get-Service # holt die gestarten Dienste Get-Process
```

Hier wird „Get-Process" ignoriert.

Seit der PowerShell 2.0 gibt es auch mehrzeilige Kommentare. Hier ist alles, was zwischen <# und #> steht, ein Kommentar, auch wenn es Zeilenumbrüche gibt.

Listing 7.1
Ein- und mehrzeilige Kommentare im Einsatz

```
<#
----------------------------------
PowerShell Script
HTTP-Download und Ausgabe am Bildschirm
(C) Dr. Holger Schwichtenberg
http://www.powershell-doktor.de
----------------------------------
#>

# Instanz erzeugen
$wc = new-object System.Net.WebClient # Es werden keine Parameter
benötigt
# HTML herunterladen aus ausgeben
$wc.DownloadString("http://www.powershell-doktor.de")
# Zusätzliche Ausgaben
"Zusätzliche Header-Informationen:"
$wc.ResponseHeaders.ToString() # Liefert eine Liste aller HTTP-
Header
```

7.4 Variablen

Untypisierte Variablen

Variablen sind Speicher für einzelne Werte, für ganze Objekte oder für Objektmengen. Variablen beginnen mit dem Variablenkennzeichner `$`. Ein Variablenbezeichner (der Name der Variable) kann aus Buchstaben und Zahlen sowie einem Unterstrich bestehen. Nicht erlaubt sind Namen, die bereits vordefinierten Variablen zugewiesen wurden, insbesondere nicht der Name `"$_"`.

Die Zuweisung des Inhalts an Variablen erfolgt mit dem Gleichheitszeichen:

```
$x = 5 # Zuweisung der Zahl 5 an die Variable $x
```

Bei der Verwendung (Auslesen des Inhalts) ist nichts weiter zu beachten:

```
"Die Zahl ist " + $x # Verwendung der Variablen
```

Man kann einer Variablen den Inhalt einer Pipeline zuweisen:

```
$laufendeDienste = Get-Service | where { $_.Status -eq "running" }
```

Und dann später darauf zugreifen, z.B.:

```
"Anzahl der laufenden Dienste: " + $laufendeDienste.Count
"Liste der Dienste: "
$laufendeDienste | ft Name, Status
```



Die Zuweisung an Variablen und das Auslesen kann alternativ mit den Commandlets `Set-Variable` bzw. `Get-Variable` erfolgen. Dabei ist jeweils zu beachten, dass in diesem Fall der Variablenname ohne „$“ anzugeben ist. Ebenso ist bei der Zuweisung kein Gleichheitszeichen zu verwenden. Die Verwendung dieser Commandlets ist jedoch umständlicher als der direkte Zugriff, da zusätzliche Klammerungen notwendig sind, um `Set-Variable` bzw. `Get-Variable` in Ausdrücken zu verwenden. Ebenso muss man bei `Get-Variable` immer noch auf `.Value` zugreifen, da `Get-Variable` nicht den Wert liefert, sondern ein Objekt vom Typ `PSVariable`, das den Wert verpackt.

```
Set-Variable x 5
"Die Zahl ist " + (Get-Variable x).Value # Verwendung der
Variablen

Set-Variable laufendeDienste  $(Get-Service | where { $_.Status
-eq "running" })
"Anzahl der laufenden Dienste: " + (Get-Variable
laufendeDienste).Value.Count
"Liste der Dienste: "
(Get-Variable laufendeDienste).Value | ft Name, Status
```



Eine befüllte Variable kann man wieder leeren mit dem Commandlet `Clear-Variable`. Dabei ist aber der Name der Variablen ohne *$* anzugeben, z.B.:

```
Clear-Variable x
```

Typisierung

Variablen sind entweder untypisiert

```
$a = 5
```

Typisierte Variablen

oder typisiert auf einen PowerShell-Datentyp oder eine beliebige .NET-Klasse

```
$a = [int] 5
$a = [System.DateTime] "1.8.1972"
```

Als Typbezeichner können alle .NET-Klassennamen verwendet werden sowie einige vordefinierte Typbezeichner (sogenannte „Accelerators“) der PowerShell. Beispielsweise sind `[int]`, `[System.Int32]` und `[int32]` völlig gleichbedeutend. `[int]` ist der eingebaute PowerShell-Typbezeichner für Ganzzahlen mit 32 Bit Länge. Dieser Typ wird abgebildet auf die .NET-Klassen `[System.Int32]`. Dieser Name kann wiederum mit `[int32]` abgekürzt werden. Wie viel man tippen möchte, ist also Geschmackssache.



Das Voranstellen des Typnamens

```
[int] $a = 5
```

bewirkt, dass die Variable nur Daten dieses Typs aufnehmen kann, und entspricht damit der klassischen Typisierung in Hochsprachen wie C++, Java und C#.

Die Tabelle zeigt die wichtigsten Datentypen.

PowerShell-Typ (Accelerator)	Äquivalente .NET-Klasse	Bedeutung
[byte]	[System.Byte]	Ganzzahl mit Länge 8 Bit, nur negative Werte (0 bis 255)
[byte[]]	[System.Byte[]]	Liste von Ganzzahlen mit Länge 8 Bit, nur negative Werte (0 bis 255)
[int]	[System.Int32]	Ganzzahl mit Länge 32 Bit, negative und positive Werte
[int[]]	[System.Int32[]]	Liste von Ganzzahlen mit Länge 32 Bit, negative und positive Werte
[long]	[System.Int64]	Ganzzahl mit Länge 64 Bit, negative und positive Werte
[long[]]	[System.Int64[]	Liste von Ganzzahlen mit Länge 64 Bit, negative und positive Werte
[string]	[System.String]	Zeichenkette, beliebige Länge
[string[]]	[System.String[]]	Liste beliebiger Länge von Zeichenketten beliebiger Länge
[char]	[System.Char]	Ein Zeichen
[char[]]	[System.Char[]]	Eine Liste von Zeichen
[bool]	[System.Boolean]	Ja-Nein-Wert ($true oder $false)
[bool[]]	[System.Boolean[]]	Liste von Ja-Nein-Werten
[float] und [single]	[System.Single]	Zahl mit Nachkommastellen, Länge 32 Bit
[double]	[System.Double]	Zahl mit Nachkommastellen, Länge 64 Bit
[double[]]	[System.Double[]]	Liste von Double-Werten
[decimal]	[System.Decimal]	Zahl mit Nachkommastellen, Länge 86 Bit
[decimal[]]	[System.Decimal[]]	Liste von Decimal-Werten

***Tabelle 7.1**: Die wichtigsten Datentypen der PowerShell*

PowerShell-Typ (Accelerator)	Äquivalente .NET-Klasse	Bedeutung
[regex]	[System.Text.RegularExpression.Regex]	Regulärer Ausdruck
[array]	[System.Array]	Array von beliebigen Objekten
[xml]	[System.Xml.XmlDocument]	XML-Dokument
[scriptblock]	[System.Management.Automation.ScriptBlock	PowerShell-Skriptblock
[hashtable]	[System.Collections.Hashtable]	Hash-Tabelle (Name-Wert-Paare) von beliebigen Objekten
[psobject]	[System.Management.Automation.PSObject]	PowerShell-Objekt. Ein PowerShell ist ein .NET-Objekt mit Erweiterungen wie NoteProperties, ScriptProperties etc.
[type]	[System.Type]	Metadaten zu einer Klasse
[type[]]	[System.Type[]]	Liste von Metadaten zu Klassen
[wmi]	[System.Mangement.ManagementObject]	WMI-Objekt
[wmiclass]	[System.Management.ManagementClass]	WMI-Klasse
[wmisearcher]	[System.Management.ManagementObjectSearcher]	WMI-Suchabfrage
[adsi]	[System.DirectoryService.DirectoryEntry]	Verzeichnisdienstobjekt
[adsisearcher]	[System.DirectoryServices.DirectorySearcher]	LDAP-Suchanfrage in einem Verzeichnisdienst

***Tabelle 7.1**: Die wichtigsten Datentypen der PowerShell (Forts.)*

Typisierungszwang

Variablen müssen nicht explizit deklariert werden und es besteht bei Schreibfehlern die Gefahr, dass es unerwünschte Effekte gibt. Mit der Anweisung `Set-PSDebug -Strict` können Sie erreichen, dass die PowerShell einen Fehler meldet, wenn man eine Variable ausliest, der man zuvor keinen Wert zugewiesen hat. Ab PowerShell 2.0 gibt es zusätz-

lich `Set-StrictMode`. Im Gegensatz zu `Set-PSDebug` wirkt sich `Set-StrictMode` nur auf den aktuellen Gültigkeitsbereich einschließlich der untergeordneten Bereiche aus und kann deshalb in einem Skript oder in einer Funktion verwendet werden, ohne globalen Einfluss zu haben.

Gültigkeitsbereiche

Eine Variable wird durch eine Zuweisung auch direkt deklariert und gilt dann innerhalb des entsprechenden Gültigkeitsraums, in dem sie deklariert wurde (z.B. einem Block, einer Unterroutine oder dem ganzen Skript). Mit `Remove-Variable` kann man eine Variablendeklaration wieder aufheben.

In dem folgenden Beispiel meldet die PowerShell beim letzten Befehl einen Fehler, weil `$y` nur innerhalb des durch die geschweiften Klammern deklarierten Blocks gültig ist:

```
Set-PSDebug -Strict
$x = 5
{
$y = 5
$x
}
$y
```

Ermittlung des Typs

Unabhängig davon, ob eine Variable typisiert wurde oder nicht, kann man den Datentyp jederzeit ermitteln. Bei untypisierten Variablen hat die Variable automatisch den Datentyp des zuletzt zugewiesenen Typs.

Den Datentyp liefert die Methode `GetType()` in Form eines .NET-Objekts vom Typ `System.Type`. Da jede PowerShell-Variable eine Instanz einer .NET-Klasse ist, besitzt jede PowerShell-Variable die Methode `GetType()`, die jedes .NET-Objekt von der Mutter aller .NET-Klassen (`System.Object`) erbt. Meistens interessiert man sich nur für den Klassennamen, den man aus `Fullname` (mit Namensraum) oder `Name` (ohne Namensraum) auslesen kann.

```
$b = [System.DateTime] "1.8.1972"
"$b hat den Typ: " + $b.GetType().Fullname
```

Vordefinierte Variablen

Vordefinierte Variablen

Die PowerShell kennt zahlreiche vordefinierte Variablen (alias eingebaute Variablen alias interne Variablen). Die nachstehende Tabelle zeigt nur eine Auswahl dieser Variablen.

Tabelle 7.2 Vordefinierte PowerShell-Variablen (Auswahl)

Variable	Bedeutung
`$true`	Wert „wahr“
`$false`	Wert „falsch“
`$OFS`	Trennzeichen für die Ausgabe von Objektmengen
`$Home`	Heimatordner des angemeldeten Benutzers
`$PSHome`	Installationsordner des PowerShell-Hosts
`$Args`	Parameter (zur Verwendung in Funktionen)
`$Input`	Aktueller Inhalt der Pipeline (zur Verwendung in Funktionen)
`$_`	Aktuelles Objekt der Pipeline (zur Verwendung in Schleifen)
`$StackTrace`	Aktuelle Aufrufreihenfolge
`$Host`	Informationen über den PowerShell-Host
`$LastExitCode`	Rückgabewert der zuletzt ausgeführten externen Windows- oder Konsolenanwendung
`$Error`	Komplette Liste aller aufgetretenen Fehler seit Start der PowerShell (die maximal gespeicherte Anzahl ist durch $MaximumErrorCount festgelegt)

Beispiel Ein Beispiel zum Einsatz von `$OFS`:

Der Befehl

```
$OFS="/" ; [string] ("a","b","c")
```

liefert die Ausgabe:

```
a/b/c
```



Alle deklarierten Variablen, sowohl die eingebauten als auch die selbst definierten, erhält man durch den Befehl `Get-ChildItem Variable`: alias `Dir Variable:`

`Dir Variable:p*` listet alle Variablen auf, die mit „p“ oder „P“ beginnen. `Get-Variable p*` hat den gleichen Effekt.

Konstanten Einige der eingebauten Variablen können nicht geändert werden. Für eigene Variablen kann man diesen Zustand erreichen mit:

```
Set-Variable variablenname -Option readonly
```



Dabei ist der Variablenname ohne das Dollarzeichen zu verwenden!

Variablenauflösung

Variablen werden nicht nur in Ausdrücken, sondern auch innerhalb von Zeichenketten aufgelöst.

Wenn Folgendes deklariert ist

```
[int] $count = 1
[string] $Computer = "E01"
```

dann kann man statt

```
$count.ToString() +". Zugriff auf Computer " + $Computer
```

auch einfacher schreiben:

```
"$count. Zugriff auf Computer $Computer"
```

In beiden Fällen ist das Ergebnis gleich:

```
"1. Zugriff auf Computer E01"
```

Die Variablenauflösung funktioniert auch in Parametern von Commandlets. Auch die beiden folgenden Befehle sind gleichbedeutend, d.h., in beiden Fällen wird der Verzeichnispfad `WinNT://E01` angesprochen:

```
Get-DirectoryEntry ("WinNT://" + $Computer)
Get-DirectoryEntry "WinNT://$Computer"
```

Die Variablenauflösung ist genau genommen keine Variablenauflösung, sondern eine Ausdruckauflösung. Das Dollarzeichen kann auch einen beliebigen Ausdruck einleiten, z.B.:

„1+3=$(1+3)“

```
"Aktuelle Uhrzeit: $((Get-Date).ToShortTimeString())"
```

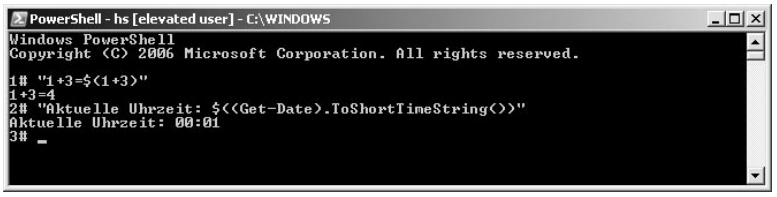


Abbildung 7.1
Ausgabe der obigen Beispiele



Eine Variablenauflösung findet nicht statt, wenn die Zeichenkette in einfachen Anführungszeichen steht:

```
'$count. Zugriff auf Computer $Computer'.
```

7.5 Zahlen

Zahlen können in der PowerShell entweder als einfache Zahlen, als Formeln oder als Wertebereiche angegeben werden. Hexadezimalzahlen können durch ein vorangestelltes 0x ausgedrückt (z.B. `0Xff = 255`) und dann verwendet werden wie Dezimalzahlen (z.B. `0Xff+1 = 256`).

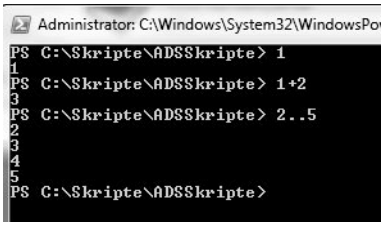


Abbildung 7.2 Zahlen in der PowerShell

Bei der Zuweisung eines Zahlenliterals zu einer untypisierten Variablen erzeugt die PowerShell im Standard eine Instanz des Typs `System.Int32`. Reicht der Wertebereich von `Int32` nicht aus, werden `Int64` oder `Decimal` erzeugt. Wenn das Zahlenliteral eine gebrochene Zahl ist (mit einem Punkt zur Trennung der Nachkommastellen), dann erzeugt die PowerShell `Double` oder `Decimal`.

Möchte man Kontrolle über den Datentyp der Variablen, muss man die Variable explizit typisieren, z.B. mit `[Byte]` oder `[Decimal]`. Für `Decimal` gibt es eine weitere Möglichkeit, indem man ein „d“ an das Literal anhängt (z.B. 5.1d).

```
# Implicit Integer
$i = 5
$i.GetType().Name

# Implicit Long
$i = 5368888888888888
$i.GetType().Name

# Implicit Decimal
$i = 536888888888888888888888888888
$i.GetType().Name

# Explicit Long
[Int64] $l = 5
$l.GetType().Name

# Explicit Byte
[Byte] $b = 5
$b.GetType().Name
```

```
# Implicit Double
$d = 5.1
$d.GetType().Name

# Implicit Decimal
$d = 5.1d
$d.GetType().Name

# Explicit Decimal
[Decimal] $d = 5.1
$d.GetType().Name
```

Beim expliziten Typisieren kann man wahlweise die PowerShell-Typen `[int]` und `[long]` oder die korrespondierenden .NET-Klassennamen `[int32]` und `[int64]` verwenden.

Mit den Kürzeln `KB`, `MB` und `GB` können die Maßeinheiten Kilobyte, Megabyte und Gigabyte zugewiesen werden, z.B. steht `5MB` für die Zahl 5242880 (5 * 1024 * 1024).



Diese Maßeinheiten gelten seit PowerShell 1.0 RC2. Vorher wurden die Kürzel `M`, `K` und `G` verwendet.

Zufallszahlen

Eine Zufallszahl kann man mit dem Commandlet `Get-Random` (ab PowerShell 2.0 oder in PowerShell 1.0 in den PowerShell Community Extensions [CODEPLEX01] erzeugen. `Get-Random` liefert eine Zahl zwischen 0 und 1. Mit den Parametern `-Min` und `-Max` kann man den Wertebereich beeinflussen (siehe *Abbildung 7.3*).

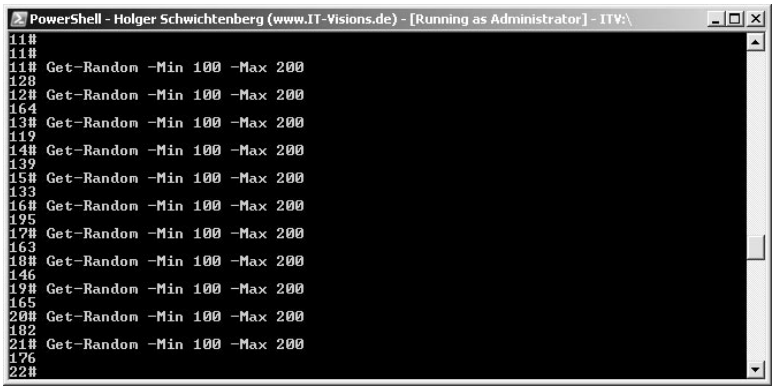


Abbildung 7.3 *Einsatz von Get-Random zur Erzeugung von Zufallszahlen zwischen 100 und 200*

7.6 Zeichenketten

Here-String

Zeichenketten sind in der PowerShell Instanzen der .NET-Klasse `System.String`. Sie werden begrenzt durch Anführungszeichen oder '@ @'. Die letzte Variante, die auch Zeilenumbrüche erlaubt, nennt Microsoft „Here-String".

Listing 7.2 *Beispiel für einen Here-String [Basiswissen/PowerShell Language/strings.ps1]*

```
#Here-String
@'
Eine lange Zeile
kann in spezielle
Begrenzer
verpackt werden
'@
```

In beiden Fällen dürfen die Zeichenketten Variablen enthalten, die automatisch aufgelöst werden.

```
$a = 10
$b= "Der aktuelle Wert ist $a!"
Write-Warn $b
```

Listing 7.3*: Beispiel für eine Variablenauflösung in einer Zeichenkette [Basiswissen/PowerShell Language/strings.ps1*



Bei der Parameterübergabe an Commandlets sind Zeichenketten nur in Anführungszeichen zu schreiben, wenn die Parameterabgrenzung sonst nicht mehr klar wäre, also wenn ein Leerzeichen darin vorkommt.

Bearbeitungsmöglichkeiten

Zeichenkettenbearbeitung

Für Zeichenketten in der PowerShell stehen alle Bearbeitungsmöglichkeiten der Klasse `System.String` in der PowerShell zur Verfügung.

Folgende Methoden stehen zur Verfügung (siehe auch *Abbildung 7.4*):

- `Clone()`
- `CompareTo()`
- `Contains()`
- `CopyTo()`
- `EndsWith()`
- `Equals()`
- `IndexOf()`
- `IndexOfAny()`
- `Insert()`

- LastIndexOf()
- LastIndexOfAny()
- Length()
- PadLeft()
- PadRight()
- Remove()
- Replace()
- Split()
- StartsWith()
- Substring()
- ToCharArray()
- ToLower()
- ToLowerInvariant()
- ToString()
- ToUpper()
- ToUpperInvariant()
- Trim()
- TrimEnd()
- TrimStart()

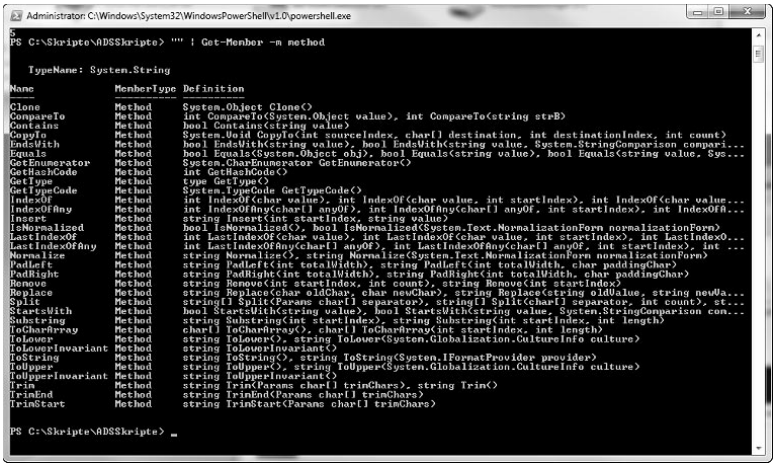


Abbildung 7.4
Methoden der Klasse System.String

Das nächste Beispiel zeigt folgende Zeichenkettenoperationen: **Beispiel**

- Umwandlung in Großbuchstaben
- Einfügen eines Textes
- Extrahieren eines Textteils als einzelne Zeichen

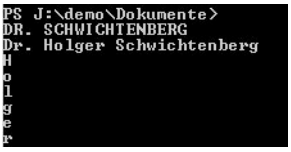
Listing 7.4 *Beispiel für die Veränderung von Zeichenketten[Basiswissen/PowerShell Language/strings.ps1]*

```
# Umwandlung in Großbuchstaben
$a = "Dr. Schwichtenberg"
$a.ToUpper()


# Einfügen eines Textes
$a = $a.Insert(4, "Holger ")
$a

# Extrahieren eines Textteils
$c = $a[4..9]
$c
```

Abbildung 7.5 *Ausgabe des obigen Skripts*



Zeichenketten trennen und verbinden

Trennen

Manchmal muss man eine Zeichenkette trennen, z.B.:

```
"Holger;Schwichtenberg;Essen;Germany;www.IT-Visions.de"
```

Das .NET Framework stellt dazu die Methode `Split()` in der Klasse `System.String` bereit.

Listing 7.5 *Einsatz der Methode Split()[Basiswissen/PowerShell Language/Strings.ps1]*

```
[String] $CSVString = "Holger;Schwichtenberg;Essen;Germany;www.IT-
Visions.de"
$CSVArray = $CSVString.Split(";")
$Surname = $CSVArray[1]
$Surname
```

Alternativ kann man das Commandlet `Split-String` aus den PSCX verwenden. Das macht es ein wenig kürzer:

Listing 7.6 *Einsatz des Commandlets Split-String[Basiswissen/PowerShell Language/Strings.ps1]*

```
[String] $CSVString = "Holger;Schwichtenberg;Essen;Germany;www.IT-
Visions.de"
$CSVArray = Split-String $CSVString -Separator ";"
$Surname = $CSVArray[1]
$Surname
```

Verbinden

Das Gegenstück zum Verbinden von Zeichenketten sind die Methoden `Join()` und das Commandlet `Join-String`. Bei `Join()` ist zu beachten, dass dies eine statische Methode der Klasse `System.String` ist.

Listing 7.7 *Einsatz der statischen Methode Join() [Basiswissen/PowerShell Language/Strings.ps1]*

```
$Array = "Holger", "Schwichtenberg", "Essen", "Germany", "www.IT-
Visions.de"
$CSVString = [System.String]::Join(";", $Array)
$CSVString
```

```
$Array = "Holger", "Schwichtenberg", "Essen", "Germany", "www.IT-
Visions.de"
$CSVString = Join-String $Array -Separator ";"
$CSVString
```

Listing 7.8 *Einsatz des Commandlets Join-String [Basiswissen/PowerShell Language/Strings.ps1]*

7.7 Reguläre Ausdrücke

Das Microsoft .NET Framework und die PowerShell enthalten eine gute Unterstützung für Mustererkennung in Texten durch reguläre Ausdrücke. Reguläre Ausdrücke sind eine komplexe Sprache, die hier nicht vollständig beschrieben werden kann. Dazu gibt es komplette Bücher. Sie finden eine gute Dokumentation dazu auch im WWW unter [MSDN08].

Ein regulärer Ausdruck ist eine Zeichenkette, die mit Platzhaltern ein Muster beschreibt. Man kann nach dieser Musterdefinition eine Zeichenkette mit dem Muster vergleichen und erhält als Ergebnis, ob die Zeichenkette auf das Muster passt. Dabei kann das Muster auch auf einen Teil der Zeichenkette zutreffen. Mit regulären Ausdrücken kann man auch Teile von Zeichenketten durch andere Zeichenketten ersetzen (Austauschfunktion).

PowerShell-Operatoren

In der PowerShell prüft man mit folgenden Operatoren auf ein Muster:

- Mustervergleich ohne Unterscheidung zwischen Groß- und Kleinschrift: `-match` und `-imatch`
- Mustervergleich mit Unterscheidung zwischen Groß- und Kleinschrift: `-cmatch`
- Prüfung, ob ein Muster NICHT enthalten ist, ohne Unterscheidung zwischen Groß- und Kleinschrift: `-notmatch` und `-inotmacht`
- Prüfung, ob ein Muster NICHT enthalten ist, ohne Unterscheidung zwischen Groß- und Kleinschrift: `-cnotmatch`

Die Anwendung sieht so aus:

```
$Ergebnis = $Zeichenkette -match $Muster
```

Das Ergebnis ist $true oder $false.

Beispiel: Prüfung einer E-Mail-Adresse

Ein erstes Beispiel dient der Verdeutlichung. Geprüft werden soll, ob die eingegebenen Zeichenketten den Aufbau einer E-Mail-Adresse haben. Die Zeichenketten werden in diesem Fall im Quellcode des Skripts hinterlegt. Die Entgegennahme der Eingabe mit `Read-Host` wird in *Kapitel 9 „Eingabefunktionen“* später behandelt.

Listing 7.9 *RegEx.ps1*

```
$Muster = "^[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,6}$"
$Eingabe1 = "hs@IT-Visions.de"
$Eingabe2 = "hsIT-Visions.de"

$Ergebnis1 = $Eingabe1 -match $Muster
$Ergebnis2 = $Eingabe2 -match $Muster
"Auswertung: {0} ist eine E-Mail-Adresse: {1}" -f $Eingabe1,
$Ergebnis1
"Auswertung: {0} ist eine E-Mail-Adresse: {1}" -f $Eingabe2,
$Ergebnis2
```

Die Ausgabe sieht so aus:

```
Auswertung: hs@IT-Visions.de ist eine E-Mail-Adresse: True
Auswertung: hsIT-Visions.de ist eine E-Mail-Adresse: False
```

Interpretation

Die Interpretation von „^[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,6}$“ sieht so aus:

- ^ steht für den Anfang und $ für das Ende der Zeichenkette.
- Eine eckige Klammer beschreibt eine Menge von Zeichen, wobei Abkürzungen erlaubt sind, also A-Z steht für A, B, C, D usw. bis Z. [A-Z0-9._%+-] umfasst alle Buchstaben, alle Zahlen sowie die Sonderzeichen Punkt, Unterstrich, Prozentzeichen, Plus und Minus. Wichtig ist, dass eine eckige Klammer nicht für eine Zeichenfolge steht, sondern nur für ein einzelnes Zeichen. Es darf also nur ein Zeichen aus der Liste vorkommen.
- Ein „+“ außerhalb von eckigen Klammern steht für „ein oder mehrmalige Wiederholung“ und bezieht sich auf die eckige Klammer davor.
- „@“ ist ein Literal. Es steht tatsächlich für den „Klammeraffen“.
- [A-Z0-9.-]+ beschreibt die Second-Level-Domäne und bedeutet dann also mindestens einmaliges Vorkommen eines der Zeichen aus der Liste in eckigen Klammern.
- \. steht für einen Punkt, der vorkommen muss. „\“ ist das „Escape“-Zeichen. Ein Punkt in regulären Ausdrücken steht eigentlich für „beliebiges Zeichen“. In diesem Fall geht es aber darum, dass wirklich nur ein Punkt vorkommen darf, daher \.
- [A-Z]{2,6} beschreibt die Top-Level-Domäne. Diese besteht nur aus Buchstaben (daher: [A-Z]) und ist immer mindestens zwei Zeichen lang. Es gibt aber auch längere Top-Level-Domänes (z.B. „museum“). Hier wird erlaubt, dass die Top-Level-Domäne zwischen 2 und 6 Zeichen lang ist.



Der oben genannte reguläre Ausdruck ist noch nicht perfekt, denn er trifft auch auf Fälle zu, in denen die E-Mail-Adresse unsinnig ist, z.B. `abc@abc.unsinn`. Es gibt hier zahlreiche Möglichkeiten, genauer zu prüfen, z.B. ^[a-z0-9!#$%&'*+/=?^_`{|}~-]+(?:\.[a-z0-9!#$%&'*+/=?^_`{|}~-]+)*@(?:[a-z0-9](?:[a-z0-9-]*[a-z0-9])?\.)+(?:[A-Z]{2}|com|org|net|gov|mil|biz|info|mobi|name|aero|jobs|museum)$.

Dieser Ausdruck ist aber schwerer verständlich.

Allgemeiner Aufbau von regulären Ausdrücken

Reguläre Ausdrücke sind Zeichenketten, in denen sowohl übliche Buchstaben als auch viele Sonderzeichen zur Musterbeschreibung eingesetzt werden. Allgemein besteht ein regulärer Ausdruck aus fünf Arten von Musterelementen:

Musterelemente

- Zeichenklasse: zum Beispiel Buchstaben oder Zahlen
- Quantifizierer (alias Quantoren): beschreibt, wie oft sich eine Zeichenklasse wiederholen muss oder darf, z.B. dreimal oder beliebig oft oder zwischen zwei- und sechsmal
- Ankerelemente (Positionselemente): z.B. Anfang und Ende einer Zeichenkette
- Runde Klammern: dienen der Gruppierung von Elementen
- Das Escape-Zeichen „\“, das einem Sonderzeichen wieder zu seiner ursprünglichen Bedeutung verhilft, z.B. „\.“ oder „\$“

Die Elemente der Sprache für reguläre Ausdrücke

Die folgenden Tabellen geben die wichtigsten Sprachelemente für reguläre Ausdrücke wieder.

Zeichenklasse	Erläuterung
[character_group]	(Positive Zeichengruppe) Entspricht einem beliebigen Zeichen in der angegebenen Zeichengruppe. Die Zeichengruppe besteht aus einem oder mehreren Literalzeichen, Escapezeichen, Zeichenbereichen oder Zeichenklassen, die miteinander verkettet sind. Zur Angabe aller Vokale verwenden Sie z.B. [aeiou]. Wenn Sie die gesamte Interpunktion und alle Dezimalziffern angeben möchten, codieren Sie [\p{P}\d].

***Tabelle 7.3**: Zeichenklassen. Diese Tabelle ist aus der MSDN-Entwicklerbibliothek entnommen. Quelle: http://msdn.microsoft.com/de-de/library/3206d374.aspx*

Zeichenklasse	Erläuterung
[^character_group]	(Negative Zeichengruppe) Entspricht einem beliebigen Zeichen, das sich nicht in der angegebenen Zeichengruppe befindet. Die Zeichengruppe besteht aus einem oder mehreren Literalzeichen, Escapezeichen, Zeichenbereichen oder Zeichenklassen, die miteinander verkettet sind. Das führende Zirkumflexzeichen (^) ist obligatorisch und gibt an, dass es sich bei der Zeichengruppe um eine negative und nicht um eine positive Zeichengruppe handelt. Zur Angabe aller Zeichen mit Ausnahme von Vokalen verwenden Sie z.B. [^aeiou]. Wenn Sie alle Zeichen außer Interpunktion und Dezimalziffern angeben möchten, verwenden Sie [^\p{P}\d].
[firstCharacter-lastCharacter]	(Zeichenbereich) Entspricht einem beliebigen Zeichen in einem Zeichenbereich. Ein Zeichenbereich ist eine Folge zusammenhängender Zeichen, die definiert wird, indem das erste Zeichen in der Folge, ein Bindestrich (-) und das letzte Zeichen in der Folge angegeben werden. Zwei Zeichen sind zusammenhängend, wenn sie benachbarte Unicode-Codepunkte haben. Es können zwei oder mehr Zeichenbereiche miteinander verkettet werden. Wenn Sie beispielsweise den Bereich der Dezimalziffern von '0' bis '9', den Bereich der Kleinbuchstaben von 'a' bis 'f' und den Bereich der Großbuchstaben von 'A' bis 'F' angeben möchten, verwenden Sie [0-9a-fA-F].
.	(Punkt) Entspricht allen Zeichen mit Ausnahme von \n. Bei Modifikation durch die Singleline-Option entspricht ein Punkt einem beliebigen Zeichen. Weitere Informationen hierzu finden Sie unter Optionen für reguläre Ausdrücke. Beachten Sie, dass ein Punkt in einer positiven oder negativen Zeichengruppe (Punkt in eckigen Klammern) als Literalzeichen und nicht als Zeichenklasse behandelt wird.
\p{name}	Entspricht einem beliebigen Zeichen in der allgemeinen Unicode-Kategorie oder einem durch name angegebenen benannten Block (z.B. Ll, Nd, Z, IsGreek und IsBoxDrawing).
\P{name}	Entspricht einem beliebigen Zeichen, das sich nicht in der allgemeinen Unicode-Kategorie oder einem in name angegebenen benannten Block befindet.
\w	Entspricht einem beliebigen Wortzeichen. Entspricht den allgemeinen Unicode-Kategorien [\p{Ll}\p{Lu}\p{Lt}\p{Lo}\p{Nd}\p{Pc}\p{Lm}]. Wenn mit der ECMAScript-Option ECMAScript-konformes Verhalten angegeben wurde, ist \w gleichbedeutend mit [a-zA-Z_0-9].
\W	Entspricht einem beliebigen Nichtwortzeichen. Entspricht den allgemeinen Unicode-Kategorien [^\p{Ll}\p{Lu}\p{Lt}\p{Lo}\p{Nd}\p{Pc}\p{Lm}]. Wenn mit der ECMAScript-Option ECMAScript-konformes Verhalten angegeben wurde, ist \W gleichbedeutend mit [^a-zA-Z_0-9].

***Tabelle 7.3**: Zeichenklassen. Diese Tabelle ist aus der MSDN-Entwicklerbibliothek entnommen. Quelle: http://msdn.microsoft.com/de-de/library/3206d374.aspx (Forts.)*

Zeichenklasse	Erläuterung
\s	Entspricht einem beliebigen Leerraumzeichen. Entspricht den Escapesequenzen und den allgemeinen Unicode-Kategorien [\f\n\r\t\v\x85\p{Z}]. Wenn mit der ECMAScript-Option ECMAScript-konformes Verhalten angegeben wurde, ist \s gleichbedeutend mit [\f\n\r\t\v].
\S	Entspricht einem beliebigen Nicht-Leerraumzeichen. Entspricht den Escapesequenzen und den allgemeinen Unicode-Kategorien [^\f\n\r\t\v\x85\p{Z}]. Wenn mit der ECMAScript-Option ECMAScript-konformes Verhalten angegeben wurde, ist \S gleichbedeutend mit [^ \f\n\r\t\v].
\d	Entspricht einer beliebigen Dezimalziffer. Entspricht \p{Nd} für Unicode und [0-9] für Nicht-Unicode mit ECMAScript-Verhalten.
\D	Entspricht einer beliebigen Nichtziffer. Entspricht \P{Nd} für Unicode und [^0-9] für Nicht-Unicode mit ECMAScript-Verhalten

***Tabelle 7.3**: Zeichenklassen. Diese Tabelle ist aus der MSDN-Entwicklerbibliothek entnommen. Quelle: http://msdn.microsoft.com/de-de/library/3206d374.aspx (Forts.)*

Quantifizierer	Erläuterung
*	Stimmt mit dem vorangehenden Element nicht oder mehrmals überein. Ist äquivalent zu {0,}. * ist ein gieriger Quantifizierer, dessen nicht gieriges Äquivalent *? ist. Der reguläre Ausdruck \b91*9*\b sucht beispielsweise nach Entsprechungen der Ziffer 9, die auf eine Wortgrenze folgen. Auf die 9 können keine oder mehrere Instanzen der Ziffer 1 folgen, auf die ihrerseits keine oder mehrere Instanzen der Ziffer 9 folgen können. Im folgenden Beispiel wird dieser reguläre Ausdruck veranschaulicht. Von den neun Ziffern in der Eingabezeichenfolge stimmen fünf mit dem Muster überein und vier nicht (95, 929, 9129 und 9919).
+	Stimmt mit dem vorangehenden Element mindestens einmal überein. Ist äquivalent zu {1,}. + ist ein gieriger Quantifizierer, dessen nicht gieriges Äquivalent +? ist. Der reguläre Ausdruck \ba(n)+\w*?\b sucht beispielsweise nach ganzen Wörtern, die mit dem Buchstaben a beginnen, auf den mindestens eine Instanz des Buchstaben n folgt. Im folgenden Beispiel wird dieser reguläre Ausdruck veranschaulicht. Der reguläre Ausdruck liefert die Wörter an, annual, announcement und antique als Ergebnis und erkennt erwartungsgemäß keine Übereinstimmung bei den Wörtern autumn und all.
?	Stimmt mit dem vorangehenden Element nicht oder einmal überein. Ist äquivalent zu {0,1}. ? ist ein gieriger Quantifizierer, dessen nicht gieriges Äquivalent ?? ist. Der reguläre Ausdruck \ban?\b sucht beispielsweise nach ganzen Wörtern, die mit dem Buchstaben a beginnen, auf den keine oder eine Instanz des Buchstaben n folgt. Er sucht also nach den Wörtern a und an. Im folgenden Beispiel wird dieser reguläre Ausdruck veranschaulicht.

***Tabelle 7.4**: Quantifizierer. Diese Tabelle ist aus der MSDN-Entwicklerbibliothek entnommen. Quelle: http://msdn.microsoft.com/de-de/library/3206d374.aspx*

Quantifizierer	Erläuterung
{n}	Stimmt mit dem vorangehenden Element genau n-mal überein. {n} ist ein gieriger Quantifizierer, dessen nicht gieriges Äquivalent {n}? ist. Der reguläre Ausdruck \b\d+\,\d{3}\b sucht beispielsweise nach Übereinstimmungen einer Wortgrenze, auf die mindestens eine Dezimalziffer, drei Dezimalziffern und eine weitere Wortgrenze folgen. Im folgenden Beispiel wird dieser reguläre Ausdruck veranschaulicht.
{n,}	Stimmt mit dem vorangehenden Element mindestens n-mal überein. {n,} ist ein gieriger Quantifizierer, dessen nicht gieriges Äquivalent {n}? ist. Der reguläre Ausdruck \b\d{2,}\b\D+ sucht beispielsweise nach Übereinstimmungen einer Wortgrenze, auf die mindestens zwei Ziffern, eine Wortgrenze und ein weiteres Zeichen folgen, bei dem es sich nicht um eine Ziffer handelt. Im folgenden Beispiel wird dieser reguläre Ausdruck veranschaulicht. Der reguläre Ausdruck liefert kein Ergebnis für den Ausdruck 7 days, da dieser nur eine Dezimalziffer enthält, er erkennt jedoch erfolgreich Übereinstimmungen mit den Ausdrücken 10 weeks und 300 years.
{n,m}	Stimmt mit dem vorangehenden Element mindestens n-mal, jedoch nicht mehr als m-mal überein. {n,m} ist ein gieriger Quantifizierer, dessen nicht gieriges Äquivalent {n,m}? ist. Der reguläre Ausdruck (00\s){2,4} sucht beispielsweise nach zwei bis vier Vorkommen zweier 0-Ziffern, auf die ein Leerzeichen folgt. Im folgenden Beispiel wird dieser reguläre Ausdruck veranschaulicht. Beachten Sie, dass der letzte Abschnitt der Eingabezeichenfolge dieses Muster fünf Mal und nicht maximal vier Mal enthält. Der Anfangsabschnitt dieser Teilzeichenfolge (bis zum Leerzeichen und dem fünften Nullenpaar) stimmt mit dem regulären Ausdrucksmuster jedoch überein.
*?	Stimmt mit dem vorangehenden Element nicht oder mehrmals, jedoch so wenige Male wie möglich überein. Dies ist ein träger Quantifizierer, der das Gegenstück zum gierigen Quantifizierer * darstellt. Der reguläre Ausdruck \b\w*?oo\w*?\b sucht beispielsweise alle Wörter, die die Zeichenfolge oo enthalten. Im folgenden Beispiel wird dieser reguläre Ausdruck veranschaulicht.
+?	Stimmt mit dem vorangehenden Element einmal oder mehrmals, jedoch so wenige Male wie möglich überein. Dies ist ein träger Quantifizierer, der das Gegenstück zum gierigen Quantifizierer + darstellt. Der reguläre Ausdruck \b\w+?\b sucht beispielsweise ein oder mehrere durch Wortgrenzen getrennte Zeichen. Im folgenden Beispiel wird dieser reguläre Ausdruck veranschaulicht.

***Tabelle 7.4**: Quantifizierer. Diese Tabelle ist aus der MSDN-Entwicklerbibliothek entnommen. Quelle: http://msdn.microsoft.com/de-de/library/3206d374.aspx (Forts.)*

Quantifizierer	Erläuterung
??	Stimmt mit dem vorangehenden Element nicht oder mehrmals, jedoch so wenige Male wie möglich überein. Dies ist ein träger Quantifizierer, der das Gegenstück zum gierigen Quantifizierer ? darstellt. Beispielsweise sucht der reguläre Ausdruck ^(\s)*(System.)??Console.Write(Line)??\(?? nach Übereinstimmungen mit den Zeichenfolgen Console.Write oder Console.WriteLine. Die Zeichenfolge kann auch System. vor Console enthalten und eine öffnende Klammer kann auf sie folgen. Die Zeichenfolge muss am Anfang einer Zeile stehen, ihr kann jedoch ein Leerzeichen vorangehen. Im folgenden Beispiel wird dieser reguläre Ausdruck veranschaulicht.
{n}?	Stimmt genau n-mal mit dem vorangehenden Element überein. Dies ist ein träger Quantifizierer, der das Gegenstück zum gierigen Quantifizierer {n}+ darstellt. Der reguläre Ausdruck \b(\w{3,}?\.){2}?\w{3,}?\b sucht beispielsweise nach genau zwei Sätzen von Zeichen, auf die ein Punkt an einer Wortgrenze folgt. Auf diesen folgt dann ein weiterer Satz von Zeichen und eine Wortgrenze. Dieser reguläre Ausdruck soll eine Website-Adresse identifizieren. Im folgenden Beispiel wird der reguläre Ausdruck veranschaulicht. Beachten Sie, dass Übereinstimmungen mit www.microsoft.com und mdsn.microsoft.com erkannt werden, jedoch nicht mit mywebsite oder mycompany.com.
{n,}?	Stimmt mit dem vorangehenden Element mindestens n-mal, jedoch so wenige Male wie möglich überein. Dies ist ein träger Quantifizierer, der das Gegenstück zum gierigen Quantifizierer {n,} darstellt. Beachten Sie zur Veranschaulichung das Beispiel für den {n}?-Quantifizierer. Der reguläre Ausdruck in diesem Beispiel verwendet den {n,}-Quantifizierer, um nach einer Zeichenfolge zu suchen, die mindestens drei Zeichen enthält, auf die ein Punkt folgt.
{n,m}?	Stimmt mit dem vorangehenden Element n-mal bis m-mal, jedoch so wenige Male wie möglich, überein. Dies ist ein träger Quantifizierer, der das Gegenstück zum gierigen Quantifizierer {n,m} darstellt. Der reguläre Ausdruck \b[A-Z](\w*?\s*?){1,10}[.!?] sucht beispielsweise nach Sätzen, die zwischen einem und zehn Wörtern enthalten. Er sucht nach einer Wortgrenze gefolgt von einem Großbuchstaben, auf den ein bis zehn Wiederholungen von keinen oder mehreren Wortzeichen und optional ein Leerzeichen folgen. Die Übereinstimmung wird dann von einem Punkt, einem Ausrufezeichen oder einem Fragezeichen beendet. Im folgenden Beispiel wird dieser reguläre Ausdruck veranschaulicht. Er liefert alle Sätze der Eingabezeichenfolge als Ergebnis mit Ausnahme eines einzigen Satzes, der 18 Wörter enthält.

***Tabelle 7.4**: Quantifizierer. Diese Tabelle ist aus der MSDN-Entwicklerbibliothek entnommen. Quelle: http://msdn.microsoft.com/de-de/library/3206d374.aspx (Forts.)*

Tabelle 7.5 Wichtige Ankerelemente für reguläre Ausdrücke

Sonstige Zeichen	Erläuterung
^	Anfang der Zeichenfolge
$	Ende der Zeichenfolge
\b	Ende eines Worts
\Z	Satzende

Beispiele

Die folgende Tabelle nennt noch weitere Beispiele für reguläre Ausdrücke.

Beispiel	Bedeutung
`\{[0-9a-fA-F]{8}-[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-[0-9|a-f|A-F]{12}\}`	GUID (Global Unique Identifier)
`(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\ .(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)`	IP-Adresse
`href\s*=\s*(?:""(?<1>[^""]*)""|(?<1>\S+))`	HREF=-Parameter in HTML-Seiten

Es gibt im Internet Websites, die reguläre Ausdrücke zur Wiederverwendung sammeln, z.B. *www.regexlib.com/* und *www.regular-expressions.info/*.

7.8 Datum und Uhrzeit

Get-Date

Das Commandlet `Get-Date` liefert eine Instanz der .NET-Klasse `System.DateTime`, die das aktuelle Datum und die aktuelle Uhrzeit enthält.

```
Get-Date
```

Die Anzeige reduziert man wie folgt auf das Datum:

```
Get-Date -displayhint date
```

Die Anzeige reduziert man so auf die Zeit:

```
Get-Date -displayhint time
```

`Get-Date` kann auch dazu genutzt werden, ein spezielles Datum zu erzeugen und dieses in einer Variablen zu speichern:

```
$a = Get-Date "8/1/1972 12:11:10"
```

Die Differenz zwischen dem aktuellen Datum und einem in einer Variablen gespeicherten Datum errechnet man durch den Aufruf der Methode `Subtract()`:

```
(Get-Date).Subtract((Get-Date "8/1/1972 12:11:10"))
```

oder durch die einfache Verwendung des Minuszeichens:

```
(Get-Date) - (Get-Date "8/1/1972 12:11:10")
```

Dies führt zu folgender Ausgabe:

```
Days              : 12662
Hours             : 11
Minutes           : 56
Seconds           : 57
Milliseconds      : 927
Ticks             : 10940398179276185
TotalDays         : 12662,4978926808
TotalHours        : 303899,949424338
TotalMinutes      : 18233996,9654603
TotalSeconds      : 1094039817,92762
TotalMilliseconds : 1094039817927,62
```

Intern verarbeitet die PowerShell Zeiträume als Instanzen der Klasse `System.TimeSpan`. Man kann auch selbst Zeiträume mit dem Commandlet `New-TimeSpan` anlegen und mit diesen rechnen, z.B.:

```
$Dauer = New-TimeSpan -Days 10 -hours 4 -minutes 3 -seconds 50
$jetzt = Get-Date
$zukunft = $jetzt + $Dauer
```



Bei `New-TimeSpan` kann man die Dauer nur in Tagen, Stunden, Minuten und Sekunden angeben. Eine Angabe in Monaten oder Jahren ist nicht möglich.

Die Zeit von einem entfernten System kann man nicht mit dem Commandlet `Get-Date` abfragen, sondern nur unter Zuhilfenahme der WMI-Klasse `Win32_Currenttime`.

```
Get-Wmiobject Win32_currenttime -computername E02
```

System.DateTime

Das Ergebnis der Operation ist dann aber kein .NET-Objekt vom Typ `System.DateTime`, sondern ein .NET-Objekt vom Typ `System.Management.ManagementObject`, das ein WMI-Objekt vom Typ `root\cimv2\Win32_LocalTime` enthält.

Die aktuelle Systemzeit kann man mit `Set-Date` setzen.

7.9 Arrays und assoziative Arrays (Hash-Tabelle)

Datenfelder

Arrays sind Mengen von Variablen, die eine frei wählbare Anzahl von Werten aufnehmen können. Ein Array deklariert man durch die Zuweisung einer durch Kommata getrennten Wertemenge:

```
$a = 01,08,72,13,04,76
```

Das Array kann auch explizit mit `[array]` deklariert werden:

```
[array] $b
$b = 01,08,72,13,04,76
```



Eine Zahlenreihe kann man auch abkürzen: `$b = 1..10` ist gleichbedeutend mit `$b = 1,2,3,4,5,7,8,9,10`.

Um auf die Elemente zuzugreifen, hat man drei Möglichkeiten:

- Zugriff auf ein einzelnes Element mit dem Index in eckigen Klammern (bei 0 beginnend), z.B. ist $b[3] das vierte Element.
- Zugriff auf mehrere Elemente, hier wird der Index durch ein Komma getrennt, z.B. bedeutet `$b[1,3,5]` das zweite, vierte und sechste Element.
- Zugriff auf einen Indexbereich. Der Indexbereich ist durch zwei Punkte zu trennen, z.B. `$a[1..5]`. Dies bedeutet das zweite bis sechste Element.

Der Operator += ergänzt ein Element am Ende eines Arrays (siehe *Abbildung 7.6*). Das Entfernen von Elementen ist nicht möglich (nur das Umkopieren in ein anderes Array).

Möchte man ein Array mit nur einem Element definieren, muss man die Liste mit einem Komma beginnen oder das Array explizit deklarieren:

```
$a = ,"Nur ein Element"
[Array] $a = "Nur ein Element"
```

Zum Auflisten eines Arrays ist `Foreach-Object` nicht zwingend notwendig. Wenn ein Array am Ende der Pipeline steht, wird das Array ausgegeben (siehe *Bild 22.59*). Das Attribut `Count` liefert die Anzahl der Elemente im Array.

```
[array] $b
$b = 1,2,3
$b.Count
```

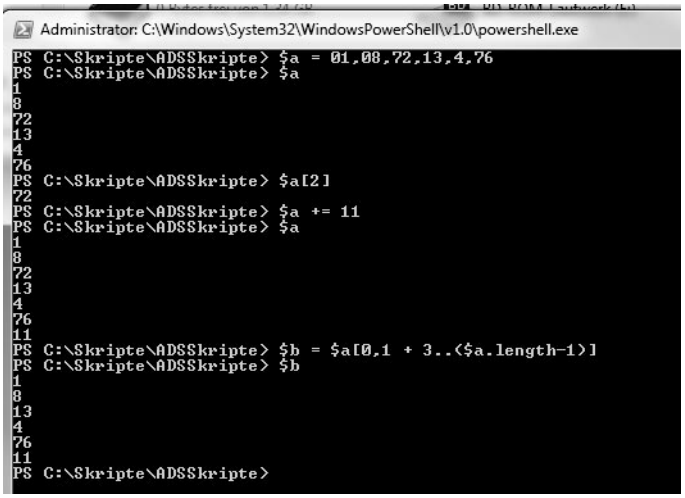


Abbildung 7.6
Arbeit mit Arrays

Zwei Arrays kann man durch den Plus-Operator verbinden:

```
$DomainControllers = "E01", "E02", "E03"
$MemberServers = "E04", "E05", "E06"
$AllServers = $DomainControllers + $MemberServers
$AllServers.Count # Ergebnis: 6 !
```

Mehrdimensionale Arrays sind möglich, indem man die Elemente mit runden Klammern zusammenfasst. In dem folgenden Beispiel entsteht ein zweidimensionales Array. Die Elemente der ersten Dimension enthalten jeweils Arrays mit drei Elementen. Auch hier kann man mit dem Plus-Operator die Menge ergänzen.

```
$DomainControllers = ("E01", "192.168.1.10", "Building 1"), ("E02",
"192.168.1.20", "Building 2"), ("E03", "192.168.1.30", "Building
3")
"Number of Computers: " + $DomainControllers.Count
"IP Address of Computer 2: " + $DomainControllers[1][1] #
192.168.1.20
"Building of Computer 2: " + $DomainControllers[1][2] # Building 3
$DomainControllers += ("E04", "192.168.1.40", "Building 4")
"Building of Computer 4: " + $DomainControllers[3][2] # Building 4
```

Assoziative Arrays (Hash-Tabelle)

Neben den Arrays unterstützt die PowerShell auch benannte (assoziative) Elementmengen in Form sogenannter Hash-Tabellen. In einer Hash-Tabelle werden die Elemente nicht durch die Position, sondern durch einen eindeutigen Bezeichner identifiziert. Dieses Konzept existiert auch in anderen Sprachen und wird dort oft „assoziatives Array" genannt. Das zu Grunde liegende Basiskonzept ist die .NET-Klasse `System.Collections.Hashtable`.

Bei der Definition einer Hash-Tabelle ist das @-Zeichen zu verwenden, gefolgt von der Elementmenge in geschweiften Klammern. Die einzelnen Elemente sind durch Semikola zu trennen. Jedes Element besteht aus einem Elementnamen und einem Elementwert, wobei Elementname und Elementwert durch ein Gleichheitszeichen zu trennen sind. Der Elementname darf nicht in Anführungszeichen stehen. Möchte man den Datentyp explizit angeben, ist `[Hashtable]` zu verwenden.

```
# Implicit Hashtable
$Computers = @{ E01 = "192.168.1.10"; E02 = "192.168.1.20"; E03 =
"192.168.1.30"; }

# Explicit Hashtable
[Hashtable] $Computers = @{ E01 = "192.168.1.10"; E02 =
"192.168.1.20"; E03 = "192.168.1.30"; }
```

Auf eine solche Hash-Tabelle kann man nicht nur wie bei den einfachen Arrays über die Notation mit eckigen Klammern zugreifen, sondern auch direkt über den Punkt-Operator. Dies macht die Arbeit mit Hash-Tabellen sehr elegant:

```
# Get IP Address of Computer E02
$Computers["E02"]
$Computers.E02
```

Man kann die Elemente auch direkt beschreiben.

```
# Change on Element
$Computers.E02 = "192.168.1.21"
```

Sehr komfortabel ist, dass beim Beschreiben eines bisher nicht existierenden Elements das Element neu angelegt wird. Auf diese Weise kann man auch eine Hash-Tabelle schrittweise anlegen, d.h. mit einer leeren Liste starten. Eine leere Hash-Tabelle wird ausgedrückt durch @{ }.

```
# Add a new Element
$Computers.E04 = "192.168.1.40"

# Start with an empty list
$MoreComputers = @{ }
$MoreComputers.E05 = "192.168.1.50"
$MoreComputers.E06 = "192.168.1.60"
$MoreComputers.Count # Result = 2
```

Zwei Hash-Tabellen kann man verbinden wie zwei Arrays. Dies funktioniert aber nur, wenn in beiden Listen zusammen jeder Elementname nur einmal vorkommt. Falls es Duplikate gibt, wird ein Fehler erzeugt und das Ergebnis ist eine leere Menge.

```
# Add two Hashtables
$AllComputers = $Computers + $MoreComputers
$AllComputers.Count # Result = 6
```

Hash-Tabellen kann man nicht nur für echte Listen, sondern auch zur einfachen Definition eigener Datenstrukturen verwenden, z.B. um Informationen über eine Person zu speichern.

```
# Use a Hashtable as a custom data structure
$Author = @{ Name="Dr.Holger Schwichtenberg"; Age=35;
Country="Germany" }
$Author.Name
$Author.Age
$Author.Country
```

7.10 Operatoren

Mathematische Operationen

Die Windows PowerShell unterstützt die elementaren arithmetischen Operatoren +, -, *, / und % (Modulo-Operation alias Divisionsrest). Das Pluszeichen verwendet man sowohl zur Addition von Zahlen als auch zur Verkettung von Zeichenketten. Sogar Mengen (Arrays und Hash-Tabellens) kann man verbinden. Auch der Stern (*) für die Multiplikation hat noch andere Bedeutung: Sowohl eine Zeichenkette als auch ein Array kann man damit multiplizieren. Dadurch werden die Zeichen bzw. Elemente so oft wiederholt wie angegeben. In der Natur einer Hash-Tabelle liegt, dass man die Elemente nicht vervielfachen kann, da dies zu doppelten Elementnamen führen würde, was nicht erlaubt ist.

```
# Multiply a string
$String = "abcdefghijklmnopqrstuvwxyz"
$LongString = $String * 20
"Count: " + $LongString.Length # = 520

# Multiply an Array
$a = 1,2,3,4,5
$b = $a * 10
"Count: " + $b.Count # = 50
```

=

Als Zuweisungsoperator wird das Gleichheitszeichen verwendet. Interessant sind Kreuzzuweisungen, mit denen auf elegante Weise die Inhalte zweier Variablen vertauscht werden können. Normalerweise braucht man dafür eine Zwischenvariable. In der PowerShell kann man aber einfach schreiben: `$x, $y = $y, $x` (siehe *Abbildung 7.7*).

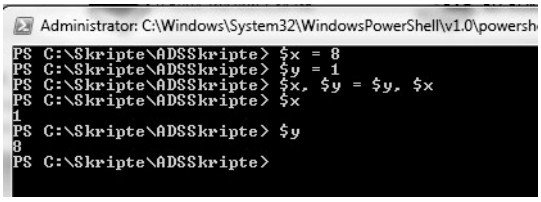


Abbildung 7.7 *Kreuzzuweisung zur Variableninhaltsvertauschung in der PowerShell*

& Ein interessanter Operator ist auch das kaufmännische Und ("&"). Damit kann man eine Zeichenkette als einen Befehl ausführen. Dies ist eine Möglichkeit, dynamischen und selbst modifizierenden Programmcode zu schreiben. Dazu ein Beispiel:

```
$What = "Process"
& ("Get-"+$What)
```

Die obige Befehlsabfolge führt zur Ausführung des Commandlets `"Get-Process "`. Nun könnte man den Inhalt der Variablen `$What` auch aus einer anderen Quelle, z.B. einer Benutzereingabe, bekommen. Alternativ kann man statt des Operators `&` auch das Commandlet `Invoke-Expression` verwenden:

```
$What = "Process"
invoke-expression("Get-"+$What)
```



Unbedingt zu beachten ist, dass dynamische Codeausführung ein Sicherheitsrisiko birgt, wenn man Benutzereingaben direkt in den Befehlen verarbeitet. Man könnte meinen, dass in dem obigen Beispiel das Risiko beschränkt ist, weil immer der „Get"-Befehl ausgeführt wird.

```
$What = "Process; Get-Service"
invoke-expression("Get-"+$What)
```

Neu in 2.0 Bei den PowerShell-Operatoren gibt es in PowerShell 2.0 nur zwei kleine Neuerungen, die Operatoren `-split` und `-join`. Mit `-split` trennt man Zeichenketten anhand eines Trennzeichens, mit `-join` verbindet man Zeichenketten in einer Liste zu einer Zeichenkette mit einem wählbaren Trennzeichen.

Listing 7.10 *Beispiele zum Einsatz der Operatoren -split und -join*

```
$CSV = "www.powershell-doktor.de;Dr. Holger Schwichtenberg;Essen"

$Einzelwerte = $CSV -split ";"
$Name = $Einzelwerte[1]
$Name # liefert "Dr. Holger Schwichtenberg"

$Einzelwerte = $Einzelwerte | Sort-Object
$AlleWerteGetrenntDurchKomma = $Einzelwerte -join ","
$AlleWerteGetrenntDurchKomma # liefert "Dr. Holger
Schwichtenberg,Essen,www.powershell-doktor.de"
```

7.11 Überblick über die Kontrollkonstrukte

Schleifen, Bedingungen und Unterroutinen

Als Kontrollstrukturen kennt die PowerShell-Skriptsprache die folgenden Konstrukte:

- `if (Bedingung) {…}  else {…}`
- `switch ($var) {Wert {…} Wert {…} default {..} } }`
- `for(Initialisierung;Bedingung;Schrittweite) { … }`
- `while (Bedingung) { … }`
- `do { … } while (Bedingung)`
- `do { … } until (Bedingung)`
- `foreach ($var in $menge) {…}`
- `function name {…}`
- `break`
- `continue`
- `return`
- `exit`
- `trap Fehlerklasse { … } else { … }`
- `throw "Fehlertext"`
- `throw Fehlerklasse`



Details zu den Befehlen finden Sie in der Hilfe zur PowerShell. Hier wird zu Gunsten anderer Inhalte auf eine detaillierte Darstellung dieser Grundkonstrukte verzichtet – zumal ihre Funktionsweise anderen Programmiersprachen sehr ähnlich ist. `Throw` und `Trap` werden in *Kapitel 7.16 „Fehlerbehandlung“* separat behandelt.

7.12 Schleifen

Bei Schleifen handelt es sich um Konstrukte zur wiederholten Verarbeitung eines oder mehrerer Befehle. Wenn ein bestimmter Befehl beispielsweise fünfmal aufgerufen werden soll, so könnte man natürlich rein theoretisch den Befehl fünfmal nacheinander in das Skript schreiben. Der PowerShell ist das egal (sie macht brav, was ihr befohlen wird), aber guter Stil wäre das nicht und bei einer großen Anzahl von Wiederholungen wäre das sehr aufwändig. Und diese Vorgehensweise versagt natürlich dann, wenn die Anzahl der Wiederholungen nicht konstant ist, sondern sich aus dem Programmablauf variabel ergibt. Eine Schleife dagegen kann an eine bestimmte Bedingung gebunden werden. Im Fall einer konstanten Menge von Schleifen-

durchläufen setzt man den Befehl in eine Schleife und lässt diese alle Werte zwischen einem unteren Wert (z.B. 1) und einem oberen Wert (z.B. 5) durchlaufen. Dadurch wird der gewünschte Befehl fünfmal aufgerufen. Allerdings sind Schleifen noch viel flexibler und es existieren unterschiedliche Formen für verschiedene Ansprüche.

Die PowerShell kennt folgende Formen von Schleifen:

- zählergesteuerte Schleifen
- bedingungsgesteuerte Schleifen

Zählergesteuerte Schleifen

Im einfachen Fall der zählergesteuerten Schleifen gibt es eine genau definierte Anzahl von Durchläufen und somit wird der enthaltene Code entsprechend oft ausgeführt. Diese Form von Schleifen ist relativ einfach zu handhaben. Man definiert einen Start- und einen Endwert. Alle Werte einschließlich der beiden angegebenen Werte werden durchlaufen. Zusätzlich ist es möglich, eine bestimmte Schrittweite anzugeben, z.B. eine Schrittweite von zwei. Dadurch wird nur jeder zweite Wert der Schleife durchlaufen.

Bedingungsgesteuerte Schleifen

Im Gegensatz dazu gibt es Schleifen, deren Fortsetzung bzw. Abbruch von einer ganz bestimmten Bedingung abhängig ist. Eine genaue Anzahl von Durchläufen ist deshalb nicht von vornherein steuerbar. Eine solche Bedingung könnte beispielsweise eine bestimmte Zahl innerhalb einer Variablen sein, z.B. „Die Schleife wird so lange durchlaufen, bis der Inhalt der Variablen `eingegebeneZahl` kleiner als 100 ist". Somit kann beispielsweise eine Benutzereingabe so lange wiederholt werden, bis der Benutzer einen korrekten Wert eingegeben hat.

Die For-Schleife

Diese Schleife beginnt bei einem bestimmten Startwert und endet bei einem Endwert. Bei jedem Durchlaufen der Schleife wird der Zähler um einen bestimmten Wert hochgezählt, bis die gewünschte Obergrenze erreicht ist. In der Regel wird der Zähler bei jedem Durchlauf um eins erhöht. Allerdings ist es auch möglich, jede andere ganzzahlige Schrittweite zu verwenden.

```
for(Initialisierung;Bedingung;Schrittweite) { … }
```

Die Konfiguration der Schleife steht in runden Klammern, die pro Durchlauf auszuführenden Anweisungen stehen in geschweiften Klammern. Im Konfigurationsteil „Initialisierung" belegt man die Laufvariable mit einem Anfangswert. Die Bedingung vergleicht die Laufvariable mit einem Endwert. Im Teil „Schrittweite" wird die Laufvariable hochgezählt (oder heruntergezählt). Jeweils eine Zahl vor oder zurück kommt man mit `$i++` bzw. `$i--`. Andere Schrittweisen kann man anstelle des zweiten Operators angeben. So zählt `$i+5` um jeweils fünf hoch und `$+-5` um fünf herunter.

Das erste Beispiel gibt die Zahlen von 1 bis 5 aus, unter Verwendung der Laufvariablen i und einer Schrittweite von 1:

```
# Schleife von 1 bis 5
for ($i = 1; $i -lt 6; $i++) { $i }
```



Abbildung 7.8 *Ausgabe der ersten For-Schleife*

Das folgende Beispiel fragt eine Zahl ab und berechnet daraus deren Fakultät. Dies geschieht durch eine Schleife, die von eins bis zur eingegebenen Zahl durchlaufen wird und alle auftretenden Werte dieser Schleife multipliziert.

Listing 7.11 *Fakultätsberechnung mit einer For-Schleife*

```
"Bitte eine Zahl eingeben:"
$Fakultaet = Read-Host
$FakultaetErgebnis = 1
for ($i = 1; $i -lt $Fakultaet; $i++)
{
$FakultaetErgebnis = $FakultaetErgebnis * $i
}
"Die Fakultät von " + $Fakultaet + " ist " + $FakultaetErgebnis
```

Eine `For`-Schleife kann man mit der Anweisung `continue` vorzeitig fortsetzen und mit `break` vorzeitig verlassen. Das folgende Beispiel zeigt, wie man „zu aufwändige" Berechnungen (die hier mehr als 32-Bit-Zahlen benötigen) unterbinden kann.

Listing 7.12 *Fakultätsberechnung mit einer For-Schleife und vorzeitiger Abbruchbedingung*

```
"Bitte eine Zahl eingeben:"
$Fakultaet = Read-Host
$FakultaetErgebnis = 1
$Abbruch = $false
for ($i = 1; $i -lt $Fakultaet; $i++)
{
$FakultaetErgebnis = $FakultaetErgebnis * $i
if ($FakultaetErgebnis -gt [System.Int32]::MaxValue) { $Abbruch =
$true; break; }
}
if ($Abbruch) { "Werteüberlauf!" }
else { "Die Fakultät von " + $Fakultaet + " ist " +
$FakultaetErgebnis }
```

Die Do/While/Until-Schleifenfamilie

Rund um die Schlüsselwörter `Do/While/Until` gibt es drei Arten von Schleifen, die jeweils über Bedingungen gesteuert werden. In Abhängigkeit davon, ob die Bedingung erfüllt ist oder nicht, wird die Schleife abgebrochen oder fortgesetzt.

Die Bedingung kann entweder vor dem ersten Durchlauf der Befehle zum ersten Mal ausgeführt werden (eine sogenannte kopfgesteuerte Schleife) oder nach dem ersten Durchlauf (eine sogenannte fußgeprüfte Schleife).



Im Vergleich zu der For-Schleife hat man bei der `Do/While/Until`-Schleifenfamilie mehr Flexibilität.

Kopfgeprüfte Schleife

Eine kopfgeprüfte Schleife beginnt man mit While. Die Bedingung steht hier vor dem Anweisungsblock, man prüft also eine Bedingung vor dem ersten Durchlaufen ab. Die Bedingung steht in runden Klammern, der Anweisungsblock in geschweiften Klammern. Die Initialisierung erfolgt vor der Schleife. Die Schrittweite steht im Anweisungsblock.

Listing 7.13 *Einfaches Zählen bis fünf mit der While-Schleife*

```
"While:"
$i = 0 # Initialisierung
while($i -lt 5) # Bedingung
{
$i++ # Schrittweite
$i # Aktion
}
```

Fußgeprüfte Schleife

Bei der fußgeprüften Schleife steht die Bedingung erst am Ende. Das hat zur Folge, dass die Schleife mindestens einmal durchlaufen wird, bevor die Bedingung überprüft wird.

Die PowerShell-Skriptsprache kennt hier zwei Varianten, die äquivalent zueinander verwendet werden können.

- Die `Do...While`-Schleife: Tue etwas, solange etwas gilt.
- Die `Do…Until`-Schleife: Tue etwas, bis etwas gilt.

Das folgende Listing zeigt das Zählen von 1 bis 5 mit beiden Schleifenformen. Zu beachten ist der Unterschied bei der Bedingung. Würde man auch bei `Do…Until` statt `$i -eq 5` `$i -lt 5` schreiben, würde die Schleife schon bei der Zahl 4 enden.

Listing 7.14 *Einfaches Zählen bis fünf mit der Do..Schleife*

```
"Do While:"
$i = 0
do
{ $i++
$i
}while($i -lt 5)

"Do Until:"
$i = 0
do
{ $i++
$i
}until($i -eq 5)
```

Die ForEach-Schleife

Die `ForEach`-Schleife ähnelt zwar der `For`-Schleife, allerdings werden bei dieser kein Start- und Endwert benötigt. Diese Schleife verwendet man zum Durchlaufen von Objektmengen. Für jedes Element in einer bestimmten Menge wird die Schleife einmal durchlaufen. Bei einer solchen Menge kann es sich beispielsweise um alle Dateien eines bestimmten Verzeichnisses handeln. Grundsätzlich kann man jede Menge in der PowerShell mit `ForEach` durchlaufen.

Die `ForEach`-Schleife ist kaum geeignet, um damit zu zählen, denn man muss die zu durchlaufenden Zahlen zunächst in einem Array ablegen.

```
# Schleife von 1 bis 5
"Foreach:"
$menge = 1,2,3,4,5
foreach ($i in $menge)
{ $i }
```

Listing 7.15
Einfaches Zählen bis fünf mit der ForEach.Schleife



Das obige Skript könnte man noch etwas optimieren, indem man `$menge = 1,2,3,4,5` mit `$menge = 1...5` abkürzt.

Wirklich interessant ist die `ForEach`-Schleife, wenn man die Anzahl der zu durchlaufenden Objekte zur Entwicklungszeit noch nicht kennt. Jede PowerShell-Pipeline kann mit `ForEach` durchlaufen werden.

```
$dienste = Get-Service -ComputerName F111
foreach ($dienst in $dienste)
{
"{0,-20}: {1}" -f  $dienst.Name ,  $dienst.Status
}
```

Listing 7.16
Durchlaufen des Pipeline-Ergebnisses mit ForEach-Schleife



Es stellt sich die Frage, wie sich das Commandlet `Foreach-Object` und die `ForEach`-Schleife voneinander unterscheiden. Zunächst einmal kann man mit beiden eine jede Objektmenge durchlaufen. Die `ForEach`-Schleife orientiert sich syntaktisch an den Konzepten der prozeduralen Programmierung, während sich das Commandlet `Foreach-Object` in das Pipeline-Konzept einreiht. Unter der Haube gibt es einen wesentlichen Unterschied, den man bei größeren Objektmengen auch bemerkt. Die Pipelines arbeiten asynchron, d.h. `Foreach-Object` beginnt schon mit der Arbeit, sobald das erste Objekt vorliegt. Die `ForEach`-Schleife beginnt hingegen erst, wenn alle Objekte vorliegen.

Wenn man den Pipeline-basierten Befehl

```
Get-ChildItem c:\Windows -Recurse -Filter "*.txt"  |
ForEach-Object { "{0,-20}: {1}" -f  $_.Name , $_.length }
```

mit der skriptbasierten Befehlsfolge

```
$Dateien = Get-ChildItem c:\Windows -Recurse -Filter "*.txt"
foreach ($datei in $Dateien)
{
"{0,-20}: {1}" -f  $datei.Name ,  $dienst.length
}
```

vergleicht, wird man zwei Konsequenzen feststellen:

- Der Pipeline-Befehl beginnt sofort mit der Ausgabe der Dateiliste. Die skriptbasierte Lösung braucht einige Zeit vor der ersten Ausgabe.
- Der Pipeline-Befehl meldet zwischen den Ausgaben Fehler über Pfade, für die es kein Zutrittsrecht gibt. Die skriptbasierte Lösung zeigt erst alle Fehlermeldungen, die ja von Get-ChildItem kommen, und dann erst die Dateiausgabe, die aus der ForEach-Schleife stammt.



Ein Kuriosum ist bei der `ForEach`-Schleife zu beachten: Wenn an Stelle der zu durchlaufenden Objektmenge `$null` steht, wird die Schleife dennoch einmal durchlaufen.

```
"Foreach über eine leere Menge"
foreach ($x in $null) {  "Versuch 1: $x" }
$dienste = (Get-Service x*)
foreach ($x in $dienste) {  "Versuch 2: $x" }
foreach ($x in (Get-Service x*)) {  "Versuch 3: $x" }
```

In den obigen Beispielen wird die Schleife in Versuch 1 und 2 jeweils einmal durchlaufen. `Get-Service x*` liefert auf den meisten Systemen kein Ergebnis, weshalb die Variable `$null` erhält. Daher sind Versuch 1 und 2 aus Sicht der Schleife gleichbedeutend. In Versuch 3 wird die Schleife aber keinmal durchlaufen, denn `Get-Service x*` liefert eine leere Menge und die behandelt `ForEach` tatsächlich so, wie man es auch von `$null` erwarten würde.

7.13 Bedingungen

Die PowerShell-Skriptsprache beherrscht, wie fast alle anderen Programmiersprachen auch, die bedingte Programmausführung. Von einer bedingten Programmausführung spricht man, wenn in Abhängigkeit von bestimmten Bedingungen nicht alle Befehle in einem

Skript ausgeführt, sondern ein oder mehrere Befehle übersprungen werden. Genau genommen heißt dies für das Skript, dass abhängig davon, ob eine bestimmte Bedingung erfüllt ist oder nicht, ein bestimmter Codeabschnitt verarbeitet bzw. nicht verarbeitet wird.

Formen von Bedingungen

Innerhalb der PowerShell-Skriptsprache gibt es zwei grundsätzliche Formen für die bedingte Programmausführung. Durch eine Fallunterscheidung wird es ermöglicht, den Programmablauf abhängig von bestimmten Bedingungen zu steuern:

- `if (Bedingung) {…}  else {…}`
- `switch ($var) {Wert {…} Wert {…} default {..} } }`

If…Else-Bedingung

Das Basiskonstrukt für eine Bedingung ist `If…Else`. Die Bedingung steht in runden Klammern, der Anweisungsblock in geschweiften Klammern. Eine „Else"-Anweisung ist optional.



Man darf weder die geschweiften noch die runden Klammern weglassen.

Das erste Beispiel zeigt drei einfache `If`-Anweisungen ohne `Else`.

Listing 7.17
Drei Bedingungen unter Einsatz von If

```
"Bitte geben Sie eine Zahl ein"
$Wert1 = Read-Host
"Bitte geben Sie noch eine Zahl ein"
$Wert2 = Read-Host

if ($Wert1 -gt $Wert2) { "Der erste Wert ist größer als der
zweite." }
if ($Wert1 -lt $Wert2) { "Der zweite Wert ist kleiner als der
erste." }
if ($Wert1 -eq $Wert2) { "Beide Werte sind gleich groß." }
```

Diese Bedingungen könnte man alternativ mit `Else` schreiben, um dann zwei If-Bedingungen zu verschachteln.

Listing 7.18
Drei Bedingungen unter Einsatz von If und Else

```
if ($Wert1 -gt $Wert2)
{ "Der erste Wert ist größer als der zweite." }
else
{
if ($Wert1 -lt $Wert2) { "Der zweite Wert ist kleiner als der
erste." }
else { "Beide Werte sind gleich groß." }
}
```

Noch etwas prägnanter geht es mit `ElseIf`. Hinter `ElseIf` kann man eine weitere Bedingung angeben.

Listing 7.19 *Drei Bedingungen unter Einsatz von If, ElseIf und Else*

```
if ($Wert1 -gt $Wert2)
{ "Der erste Wert ist größer als der zweite." }
elseif ($Wert1 -lt $Wert2)
 { "Der zweite Wert ist kleiner als der erste." }
else { "Beide Werte sind gleich groß." }
```



Statt für eine Aktion (bisher immer: eine Ausgabe) kann man `If…Else` auch für Wertzuweisungen nutzen. In dem folgenden Beispiel erhält `$Anzeige` abhängig von der Bedingung einen anderen Wert:

```
[DateTime]::Now.Second
$Bedingung = [DateTime]::Now.Second % 2 -eq 0
$Bedingung
$Anzeige = if ($Bedingung) { "gerade" }  else { "ungerade" }
$Anzeige
```

Switch-Bedingung

Mit einer `Switch`-Bedingung kann man auf elegante Weise viele Fälle unterscheiden, weil man übersichtlich Bedingungen und Anweisungsblöcke aneinanderreihen kann.

Listing 7.20 *Drei Bedingungen unter Einsatz von Switch*

```
switch ()
{
 {$Wert1 -gt $Wert2} { "Der erste Wert ist größer als der zweite."
}
 {$Wert1 -lt $Wert2} { "Der erste Wert ist größer als der zweite."
}
 default { "Beide Werte sind gleich groß." }
}
```

Das runde Klammernpaar hinter `Switch` deutet schon an, dass hier noch mehr Potenzial ist. Man kann auch eine Variable mit einer festen Wertemenge vergleichen.

Listing 7.21 *Viele Bedingungen mit Switch*

```
 "Welche Note geben Sie diesem Buch?"
 $note = Read-Host
 switch ($note)
 {
    1 {"sehr gut"}
    2 {"gut"}
    3 {"befriedigend"}
    4 {"ausreichend"}
    5 {"ungenügend"}
    default { "Ungültige Note" }
 }
```

7.14 Unterroutinen (Prozedur/Funktionen)

Um die Übersichtlichkeit eines Skripts zu verbessern und Wiederholungen von Programmcode an mehreren Stellen im Skript zu vermeiden, besteht die Möglichkeit, wiederkehrende Programmzeilen in sogenannte Unterroutinen zu kapseln und anstelle der wiederkehrenden Befehle nur noch die Unterroutinen aufzurufen. Dadurch erhöht sich einerseits die Lesbarkeit; es werden nicht mehr alle Befehle, sondern nur noch eine Funktion aufgerufen. Des Weiteren schlägt sich eine Änderung an der Funktion sofort im gesamten Skript nieder.

Prozedur versus Funktion

Eine Unterroutine, die keinen Wert als Ergebnis ihrer Ausführung zurückliefert, wird oft auch als „Prozedur" bezeichnet. Im Kontrast dazu heißen Unterroutinen mit Rückgabewert „Funktion".

function

Die PowerShell macht keinen großen Unterschied: In beiden Fällen wird die Unterroutine definiert durch:

```
function Name { Anweisungsblock }
```

Es gibt in der PowerShell-Skriptsprache, wie in vielen anderen Sprachen, das Schlüsselwort `return`, um eine Unterroutine vorzeitig zu verlassen. Dabei kann man einen Wert übergeben, der dem Aufruf zurückgesendet werden soll.

```
function Name { Anweisungsblock … return xy }
```



Anders als in vielen anderen Programmiersprachen ist `return` aber nicht die einzige Möglichkeit, einen Rückgabewert zu liefern. Vielmehr wird alles zum Rückgabewert, was nicht explizit mit `Write-Host`, `Write-Debug` oder `Write-Warning` (sowie direkt über die Standardausgabe, z.B. von klassischen Konsolenanwendungen) ausgegeben wird.

Prozeduren

$null

Das erste Beispiel zeigt eine Prozedur. Die Prozedur macht zwei Ausgaben und eine Berechnung. Die Rückgabe an den Aufrufer ist `$null` (repräsentiert „Kein Wert").

```
function Get-AltersProzedur()
{
Write-Host "Dieser Funktion wurde übergeben: $($args[0]) und
$($args[1])"
$Tage = [int] ([System.DateTime]::now -
[DateTime]$Args[1]).TotalDays
Write-Host "$($args[0]) ist $Tage Tage alt!"
}
```

Listing 7.22
Dies ist eine echte Prozedur.

Der Aufruf würde so aussehen:

```
# Normaler Aufruf
Get-AltersProzedur "Max Müller" 08.09.1970
# Test, ob wirklich $null zurückkommt --> Erfüllt!
if ((Get-AltersProzedur "Max Müller" 08.09.1970) -eq $null) {
"$null!" }
```



Beim Aufruf einer PowerShell-Funktion verwendet man keine runden Klammern und kein Komma zur Parametertrennung. Hier ist ein Unterschied zum Aufruf von Methoden in Objekten (siehe *Anhang A „Crashkurs „Objektorientierung““*). Der Aufruf von Funktionen unterscheidet sich nicht vom Aufruf von Commandlets. Dies ist auch das Ziel: Man will Commandlets und Funktionen äquivalent benutzen können. Tatsächlich sind einige Commandlets in Form von Funktionen implementiert. Es ist durchaus guter Stil, Funktionen nach denselben Namenskonventionen wie Commandlets (Verb-Substantiv) aufzubauen.

Der Umgang mit Rückgabewerten

Keine Prozedur

Bei der nachstehenden PowerShell-Unterroutine könnte man auf den ersten Blick geneigt sein zu sagen, dass es sich auch um eine Prozedur ohne Rückgabewert handelt.

Listing 7.23
Dies ist eine Funktion mit zwei Rückgabewerten.

```
function Get-AltersFunktion1()
{
"Dieser Funktion wurde übergeben: $($args[0]) und $($args[1])"
$Tage = [int] ([System.DateTime]::now -
[DateTime]$Args[1]).TotalDays
"$($args[0]) ist $Tage Tage alt!"
}
```

Das ist aber nicht korrekt, denn

```
if ((Get-Altersfunktion1 "Max Müller" 08.09.1970) -eq $null) {
"null!" } else { "nicht null!"}
```

liefert „nicht null!“. Die obige Unterroutine ist eine Funktion, die zwei Zeichenketten zurückliefert. In *Kapitel 8 „Ausgaben“* steht zwar, dass „lose“ Literale und Variablen ausgegeben werden, aber wenn solch ein „loser“ Ausdruck in einer Unterroutine steht, dann wird der Wert des Ausdrucks **zusätzlich** zum Rückgabewert der Methode. Die Ausgabe wird unterdrückt, wenn der Aufrufer den Rückgabewert weiterverarbeitet. Die Ausgabe findet nur statt, wenn der Aufrufer den Rückgabewert ignoriert. Gleiches gilt für Pipelines.

Eine Testfunktion hilft dem Verständnis. Diese Routine erzeugt zwei Ausgaben direkt mit `Write-Host` zu Beginn und zum Ende der Verarbeitung. Dazwischen gibt es vier Zeilen, die eine Ausgabe erzeugen, die zum Rückgabewert wird.

```
function TestFunktion()
{
Write-Host "Beginn der Funktion..."
Get-Service d*
Get-Date | foreach { $_.ToShortDateString() }
2+2
$name="Holger"
$name
Write-Host "Ende der Funktion!"
}
```

Listing 7.24 *Testfunktion für die folgende Tabelle*

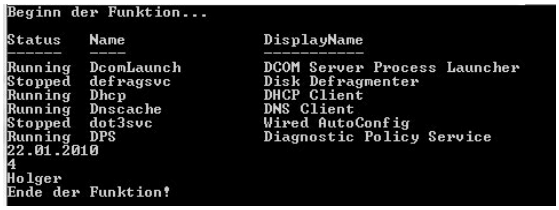
Art des Aufrufs	Ausgabe
TestFunktion	 Erläuterung: Der Aufrufer verwendet die Rückgabewerte nicht. Daher gibt die Funktion sie direkt aus, was man daran erkennt, dass die Werte zwischen „Beginn“ und „Ende“ stehen.
$x = TestFunktion	 Erläuterung: Der Aufrufer verwendet die Rückgabewerte, indem er sie in einer Variablen speichert. Daher gibt die Funktion nichts aus.
TestFunktion | foreach { $_.GetType() }	 Erläuterung: Der Aufrufer sendet die Objekte über die Pipeline an `Foreach-Object`. Nun mag überraschen, dass die Ausgaben zwischen „Beginn“ und „Ende“ stehen, was darauf hindeuten würde, dass die Funktion die Ausgaben macht. Das ist aber nicht so, weil der Aufrufer das Ergebnis ja verarbeitet. Der Grund dafür, dass die Werte zwischen „Beginn“ und „Ende“ stehen, liegt an der asynchronen Verarbeitung der Pipeline. `Foreach-Object` verarbeitet die von der Unterroutine gelieferten Objekte schon, bevor diese überhaupt fertig ist. An der Ausgabe sieht man genau, dass die Unterroutine eine heterogene Menge von Werten liefert.

Tabelle 7.6*: Verschiedene Aufrufformen und deren Ausgabe*

Art des Aufrufs	Ausgabe
TestFunktion | sort | foreach { $_.GetType() }	 Erläuterung: Nun steht die Ausgabe nach dem „Ende“, denn das `Sort-Object` verhindert die asynchrone Verarbeitung.
$x = TestFunktion $x | foreach { $_.GetType() }	 Erläuterung: Die Ausgabe steht nach dem „Ende“, weil der Aufrufer die Werte erst in einer Variablen speichert und dann an die Pipeline sendet. Zwischen zwei Zeilen findet keine asynchrone Verarbeitung statt.

***Tabelle 7.6**: Verschiedene Aufrufformen und deren Ausgabe (Forts.)*



„Return xy“ ist äquivalent zu `xy; return`.

Art der Rückgabewerte

Ein Wert und eine Liste von Werten

Bei der PowerShell sind alle Datentypen als Rückgabewerte erlaubt. Anders als in anderen Programmiersprachen ist eine beliebige Anzahl von Rückgabewerten erlaubt, die in eine Liste (vom Typ Array von Object, in der PowerShell-Notation: `Object[]`) geliefert werden. Eine Unterscheidung gibt es noch, ob ein oder mehrere Werte zurückgegeben werden. Wenn nur ein Wert zurückgegeben wird, dann ist der Rückgabewert keine Liste mit einem Element, sondern genau dieser eine Wert. Dies beweist das folgende Skript. Zu bedenken ist, dass `Count` kein Ergebnis liefert, wenn $x keine Liste ist.

```
function Get-ReturnValues([int] $Anz) # Anz legt die Anzahl der
Rückgabewerte fest
{
for([int] $i=0;$i -lt $Anz;$i++) { $i }
}

$x = (Get-ReturnValues 1)
"Get-ReturnValues 1 --> Typ: {0}, Anzahl {1}" -f $x.GetType(),
$x.Count
$x = (Get-ReturnValues 2)
"Get-ReturnValues 2 --> Typ: {0}, Anzahl {1}" -f $x.GetType(),
$x.Count
$x = (Get-ReturnValues 1000000)
"Get-ReturnValues 1000000 --> Typ: {0}, Anzahl {1}" -f
$x.GetType(), $x.Count
```

Testskript zum Prüfen der Art der Rückgabewerte

```
PS P:\> . 'h:\demo\powershell\1_basiswissen\powershell language\functions_returnvalues.ps1'
Get-ReturnValues 1 --> Typ: System.Int32, Anzahl
Get-ReturnValues 2 --> Typ: System.Object[], Anzahl 2
Get-ReturnValues 1000000 --> Typ: System.Object[], Anzahl 1000000
```

Abbildung 7.9
Ausgabe des obigen Skripts

Parameterübergabe

Parameter für Funktionen

Zwischen Funktion und Prozedur gibt es also in der PowerShell nur einen marginalen Unterschied. Ein interessanterer Unterschied sind die verschiedenen Formen der Parameterübergabe. Im Beispiel `Get-AltersFunktion1()`waren die Parameter weder explizit benannt noch typisiert. Dies hat den Vorteil, dass man leicht beliebig viele Parameter in beliebiger Form übergeben kann. Der Nachteil ist jedoch, dass die PowerShell nicht prüfen kann, ob Anzahl und Art der Parameter stimmen. Folgender Aufruf führt zu einem Abbruch in dem Moment, wo eine Rechenoperation mit dem Datum stattfinden soll:

```
Get-Altersfunktion1 "Max Müller" "unsinn"
```

Es gibt drei andere Formen der Parameterübergabe (siehe auch Listing):

- Benannte Parameter: `Get-Altersfunktion2($Name, $Geb) { … }`
- Benannte und typisierte Parameter im Funktionskopf: `Get-Altersfunktion3([string] $Name, [DateTime] $Geb) { … }`
- Benannte und typisierte Parameter im Funktionsrumpf: `Get-Altersfunktion4{ param([string] $Name, [DateTime] $Geb) … }`

```
# Parameter unbenannt
function Get-Altersfunktion1()
{
"Dieser Funktion wurde übergeben: $($args[0]) und $($args[1])"
$Tage = [int] ([System.DateTime]::now -
[DateTime]$Args[1]).TotalDays
return  "$($args[0]) ist $Tage Tage alt!"
}
```

Listing 7.25
Vergleich der verschiedenen Parameterformen

```
# Parameter benannt
function Get-Altersfunktion2($Name, $Geb)
{
"Dieser Funktion wurde übergeben: $name und $geb"
$Tage = [int] ([System.DateTime]::now - [DateTime]$Geb).TotalDays
return  "$Name ist $Tage Tage alt!"
}

# Parameter benannt und typisiert
function Get-Altersfunktion3([string] $Name, [DateTime] $Geb)
{
"Dieser Funktion wurde übergeben: $name und $geb"
$Tage = [int] ([System.DateTime]::now - $Geb).TotalDays
return  "$Name ist $Tage Tage alt!"
}

# Parameter benannt und typisiert - andere Syntaxform
function Get-Altersfunktion4
{
param([string] $Name, [DateTime] $Geb)
"Dieser Funktion wurde übergeben: $name und $geb"
$Tage = [int] ([System.DateTime]::now - $Geb).TotalDays
return  "$Name ist $Tage Tage alt!"
}
```

Der Aufruf ist in allen vier Fällen gleich. Der Unterschied ist nur, dass im Fall der typisierten Parameter der Fehler schon beim Aufruf der Unterroutine auffällt (`cannot process argument transformation on parameter 'Geb'. Cannot convert value „unsinn" to type „System.DateTime"`), so dass die Routine gar nicht erst gestartet wird.



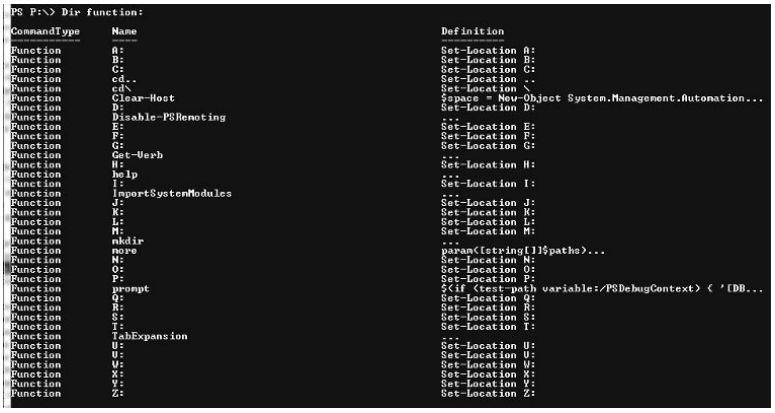
Eine falsche Anzahl von Parametern moniert die PowerShell in keinem Fall. `$args[x]` kann man immer zum Zugriff auf die übergebenen Parameter verwenden.

Dies bedeutet auch, dass es in der PowerShell keine Überladung von Unterroutinen geben kann. Überladung bedeuet in anderen Sprachen, dass es mehrere gleichnamige Unterroutinen geben darf, die sich nur hinsichtlich der Parameterzahl oder Parameterart unterscheiden. Beim Aufruf wird anhand der übergebenen Parameter entschieden, welche Unterroutine aufgerufen wird. In der PowerShell hingegen wird – falls es mehrere gleichnamige gibt – immer die erste verfügbare Unterroutine mit allen Parametern aufgerufen.

Es gibt aber Überladungen in der .NET-Klassenbibliothek und bei Aufrufen von Methoden in .NET-Klassen berücksichtigt die PowerShell dies und ruft die passende Methode auf.

7.15 Eingebaute Funktionen

Wie schon erwähnt sind einige Commandlets als PowerShell-Funktion realisiert (z.B. `Clear-Host`). Die Ausführung des Befehls `dir function:` listet alle Funktionen auf und zeigt, dass auch einige als Kompatibilität zur klassischen Windows-Konsole erhalten gebliebene Anweisungen wie C: und `more` als eingebaute Funktionen der PowerShell realisiert sind.



```
PS P:\> Dir function:

CommandType     Name                                               Definition
-----------     ----                                               ----------
Function        A:                                                 Set-Location A:
Function        B:                                                 Set-Location B:
Function        C:                                                 Set-Location C:
Function        cd..                                               Set-Location ..
Function        cd\                                                Set-Location \
Function        Clear-Host                                         $space = New-Object System.Management.Automation...
Function        D:                                                 Set-Location D:
Function        Disable-PSRemoting                                 ...
Function        E:                                                 Set-Location E:
Function        F:                                                 Set-Location F:
Function        G:                                                 Set-Location G:
Function        Get-Verb                                           ...
Function        H:                                                 Set-Location H:
Function        help                                               ...
Function        I:                                                 Set-Location I:
Function        ImportSystemModules                                ...
Function        J:                                                 Set-Location J:
Function        K:                                                 Set-Location K:
Function        L:                                                 Set-Location L:
Function        M:                                                 Set-Location M:
Function        mkdir                                              ...
Function        more                                               param([string[]]$paths)...
Function        N:                                                 Set-Location N:
Function        O:                                                 Set-Location O:
Function        P:                                                 Set-Location P:
Function        prompt                                             $(if (test-path variable:/PSDebugContext) { '[DB...
Function        Q:                                                 Set-Location Q:
Function        R:                                                 Set-Location R:
Function        S:                                                 Set-Location S:
Function        T:                                                 Set-Location T:
Function        TabExpansion                                       ...
Function        U:                                                 Set-Location U:
Function        V:                                                 Set-Location V:
Function        W:                                                 Set-Location W:
Function        X:                                                 Set-Location X:
Function        Y:                                                 Set-Location Y:
Function        Z:                                                 Set-Location Z:
```

Abbildung 7.10 *Eingebaute Funktionen der PowerShell 2.0*

7.16 Fehlerbehandlung

Die PowerShell 2.0 bietet zwei Arten der Fehlerbehandlung im Skriptcode:

- `Trap` (seit PowerShell 1.0)
- `Try-Catch-Finally` (seit PowerShell 2.0)

Fehlerbehandlung mit Trap

Trap

Die Windows PowerShell unterscheidet zwischen Fehlern, die unbedingt ein Ende der Ausführung des Befehls/des Skripts erfordern (Terminating Error), und Fehlern, die es erlauben, dass die Ausführung beim nächsten Befehl fortgesetzt wird (Non-Terminating Error). Abbrechende Fehler können durch `Trap`-Anweisungen abgefangen werden. Nichtabbrechende Fehler können zu abbrechenden Fehlern gemacht werden.

`Trap` fängt aufgetretene abbrechende Fehler ab und führt den angegebenen Code aus. `$_` enthält dann Informationen zu dem Fehler in Form einer Instanz von `System.Management.Automation.ErrorRecord`. Das Unterobjekt `$_.Exception` ist der eigentliche Fehler in Form einer Instanz einer Klasse, die von `System.Exception` erbt. Über `$_.Exception.GetType().FullName` erhält man den Fehlertyp, durch `$_.Exception.Message` den Fehlertext.

Break oder Continue

`Break` oder `Continue` entscheiden, ob das Skript nach dem Fehler fortgesetzt wird. Das Standardverhalten ist `Continue`. Mit `Exit` kann ein definitives sofortiges Skriptende herbeigeführt werden.

Beispiel

Mit dem folgenden Beispiel können Sie selbst das Fehlerverhalten der PowerShell testen und mit den verschiedenen Reaktionsmöglichkeiten experimentieren. Der Fehler wird aufgelöst durch den Aufruf `Copy-Item` mit falschem Pfad (ein nichtabbrechender Fehler) und „Get-Dir“ (das Commandlet gibt es nicht, ein abbrechender Fehler).

Listing 7.26 Trap5_Copy.ps1[Basiswissen/PowerShell Language/Trap5_Copy.ps1]

```
# Beispiel zum Testen von Fehlerabfangen

trap {
Write-Host ("### ABGEFANGENER FEHLER: " + $_.Exception.Message)
#Write-Error ("Fehler: " + $_.Exception.Message)
#continue
#break
#exit
#throw "test"
}

"Beispiel zum Testen von Fehlerabfangen"
"Erst geht alles gut..."
copy g:\daten\lieferanten c:\temo\Daten
"Dann läuft es nicht mehr so gut (falscher Pfad)"
copy g:\daten\lieferanten k:\daten\lieferanten
"Und dann folgt Unsinn (falsches Commandlet)"
Get-Dir k:\daten\lieferanten
"Ende des Scripts"
```

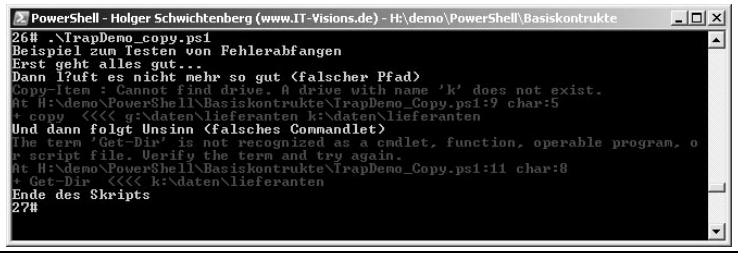
Trap	Reaktion
Nicht vorhanden	PowerShell zeigt Fehlermeldungen für Copy-Item („drive does not exists“) und Get-Dir („not recognized as a cmdlet, function, program oder script file“) und setzt die Ausführung bis zum Ende fort. 

***Tabelle 7.7**: Verhalten der PowerShell im Fehlerfall bei der Verwendung von Trap*

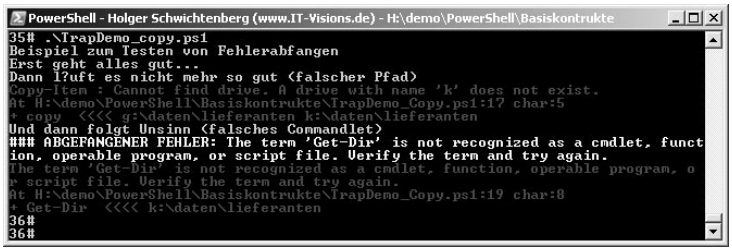
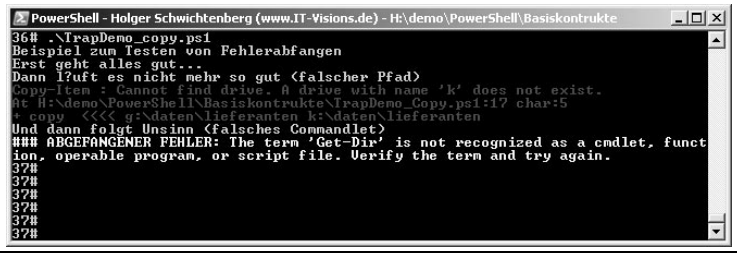
<table>
<tr><th>Trap</th><th>Reaktion</th></tr>
<tr><td>Vorhanden, nur mit Write-Host</td><td>Es erscheint für den abbrechenden Fehler zusätzlich zu der PowerShell-Fehlermeldung auch der eigene Fehlertext aus dem Trap-Block.
 
</td></tr>
<tr><td>Vorhanden, mit continue</td><td>Es erscheint für den abbrechenden Fehler nur der eigene Fehlertext aus dem Trap-Block.
 
</td></tr>
<tr><td>Vorhanden, mit break</td><td>Nach dem abbrechenden Fehler erscheinen der eigene Fehlertext und dann die Fehlermeldung der PowerShell. Danach bricht das Skript ab (d.h., die Ausgabe „Ende des Skripts“ wird nicht mehr erzeugt).
 
</td></tr>
<tr><td>Vorhanden, mit exit</td><td>Nach dem abbrechenden Fehler erscheint der eigene Fehlertext. Dann bricht die Ausführung sofort ab.
 
</td></tr>
</table>

***Tabelle 7.7**: Verhalten der PowerShell im Fehlerfall bei der Verwendung von Trap (Forts.)*

Konfiguration des Fehlerverhaltens

ErrorAction

Die Möglichkeiten werden noch vielfältiger, weil jedes einzelne Commandlet über den Parameter `-ErrorAction` (kurz `-ea`) zudem bestimmen kann, wie mit Fehlern umgegangen werden soll:

- `Stop`: Der Fehler wird ausgegeben und die Ausführung bricht ab. (Alle Non-Terminating-Fehler werden damit zu Terminating-Fehlern.)
- `Continue`: Der Fehler wird ausgegeben und die Ausführung wird fortgesetzt.
- `SilentlyContinue`: Der Fehler wird nicht ausgegeben und die Ausführung wird fortgesetzt.
- `Inquire`: Der Benutzer wird gefragt, ob er die Ausführung trotz des Fehlers fortsetzen möchte.

Alle möglichen Kombinationen von `-ErrorAction` und `Trap` durchzuspielen, würde den Rahmen dieses Buchs sprengen. Daher werden im Folgenden nur ausgewählte Fälle beschrieben.



Die Anwendung von `-ErrorAction` hat nur Auswirkung auf existierende Commandlets. Das in dem Beispiel verwendete, nicht vorhandene Commandlet „`Get-Dir`" kann darauf nicht reagieren.

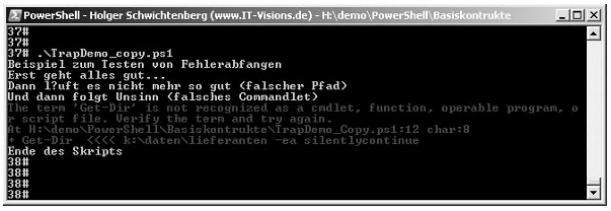
Trap	ErrorAction	Reaktion
Nicht vorhanden	`-ErrorAction silently-continue`	Es erscheint keine Fehlermeldung mehr für den Pfadfehler bei Copy-Item. Weiterhin wird das Problem mit Get-Dir gemeldet. 
Vorhanden, mit continue	`-ErrorAction silently-continue`	Es erscheinen überhaupt keine Standardfehlermeldungen der Power-Shell mehr, sondern nur noch die benutzerdefinierte Meldung aus dem Trap-Block für das nicht existierende Commandlet. 

***Tabelle 7.8**: Verhalten der PowerShell im Fehlerfall bei der Verwendung von Trap und -ErrorAction*

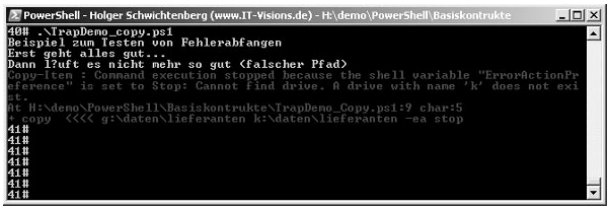
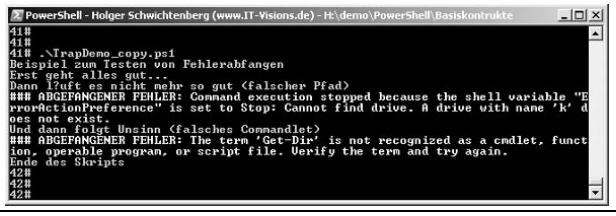
Trap	ErrorAction	Reaktion
Nicht vorhanden	`-ErrorAction stop`	Die Ausführung bricht mit einer PowerShell-Fehlermeldung nach dem ersten nicht ausführbaren Copy-Befehl ab. 
Vorhanden, mit continue	`-ErrorAction stop`	Für beide Fehler erscheint nur der eigene Fehlertext aus dem Trap-Block. 

***Tabelle 7.8**: Verhalten der PowerShell im Fehlerfall bei der Verwendung von Trap und -ErrorAction (Forts.)*

Weitere Möglichkeiten

Das war aber immer noch nicht alles, was die PowerShell in Sachen Fehlerbehandlung zu bieten hat:

Variablen

- Über die globale eingebaute Variable `$ErrorActionPreference` kann man das Standardverhalten für `-ErrorAction` für alle Commandlets setzen. Dies ist in der Standardeinstellung „Continue“.
- `$Error` enthält die gesamte Geschichte der aufgetretenen Fehler in Form von Objekten, die zu Fehlerklassen gehören, z.B. `System.Management.Automation.CommandNotFound Exception`.

Fehlerklassen

- `Trap`-Blöcke können durch Angabe eines Fehlertyps in eckigen Klammern (Fehlerklasse) auf bestimmte Fehlerarten beschränkt werden. Daher kann es mehrere `Trap`-Blöcke in einem Skript geben.

Fehler auslösen

- Mit `Throw` kann man innerhalb und außerhalb von `Trap`-Blöcken beliebige eigene Fehler erzeugen. `Throw` erzeugt einen abbrechenden Fehler der Klasse `System.Management.Automation.RuntimeException`. Man kann aber auch eine andere Fehlerklasse in eckigen Klammern angeben. Die angegebene Klasse muss von `System.Exception` abgeleitet sein.

```
throw "Fehlertext"
throw [System.ApplicationException] "Fehlertext"
```

Fehlerbehandlung mit Try-Catch-Finally

Vergleich mit Trap

Die `Trap`-Anweisung fängt alle Fehler im aktuellen Gültigkeitsbereich ab. Eine `Trap`-Anweisung in einer Funktion gilt für den ganzen Programmcode in der Funktion. Eine `Trap`-Anweisung in globalem Programmcode gilt für den ganzen globalen Code. Es ist unerheblich, wo die Trap-Anweisung steht (am Anfang, in der Mitte, am Ende), sie gilt immer für den gesamten Gültigkeitsbereich.

In PowerShell 2.0 hat Microsoft das aus den .NET-Sprachen bekannte `Try-Catch-Finally` als zweite Alternative der Fehlerbehandlung mit `Trap` eingeführt. `Try` enthält den Codeblock, der fehlschlagen könnte. `Catch` behandelt auftretende Fehler, wobei der erste abbrechende Fehler die Kontrolle in den `Catch`-Block verlagert. `Finally` (als optionaler Block) wird in jedem Fall am Ende ausgeführt.

Hinweis: Während man mit einer `Trap`-Anweisung alle Fehler in dem aktuellen Gültigkeitsbereich abfängt, bezieht sich `Try-Catch` immer nur auf Fehler im `Try-Block`. Die Frage, ob man `Trap` oder `Try-Catch` bevorzugt, ist eine Geschmackssache.

Listing 7.27 *Beispiel für den Einsatz von Try-Catch-Finally*

```
##########################################
# PowerShell-Skript
# (C) Dr. Holger Schwichtenberg
##########################################

# Beispiel zum Testen von Fehlerabfangen

try
{
"Beispiel zum Testen von Fehlerabfangen"
"Erst geht alles gut ..."
copy c:\data\projects c:\temp\Projects -Force
"Dann laeuft es nicht mehr so gut (falscher Pfad) (aber wir machen
weiter:
continue)"
copy c:\data\customers c:\temp\customers -ea continue
Und dann folgt Unsinn: falsches Commandlet (da wird -continue
ignoriert!)"

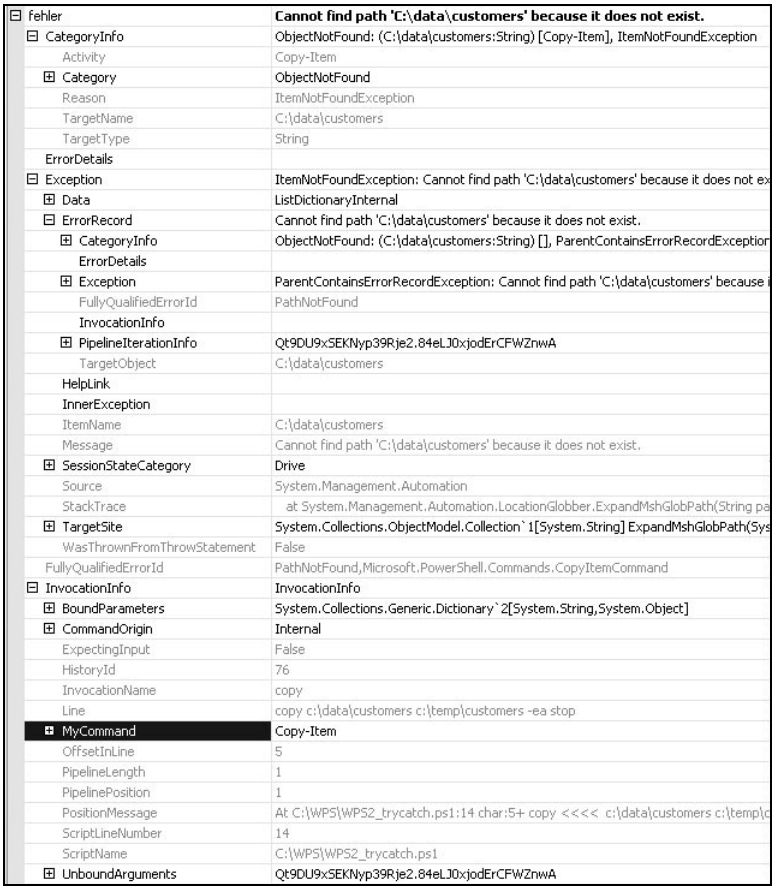
Get-Dir c:\temp\customers -ea continue
"Kopiervorgang abgeschlossen!"
}

catch
{
$fehler = $_
Write-Host ("### ABGEFANGENER FEHLER: " + $_.Exception.Message) -
ForegroundColor yellow
```

```
}
finally
{
"Ende des Skripts"
}
```

Fehlerklasse

Innerhalb des `Catch`-Blocks ist `$_` eine Instanz der Klasse `System.Management.Automation.ErrorRecord`, wobei über das Attribut `Exception` zu der aus dem .NET Framework bekannten Instanz eine .NET-Fehlerklasse kommt. Details über den Umfang der bereitgestellten Fehlerinformationen zeigt die nachstehende Bildschirmabbildung.



⊟ fehler	**Cannot find path 'C:\data\customers' because it does not exist.**
⊟ CategoryInfo	ObjectNotFound: (C:\data\customers:String) [Copy-Item], ItemNotFoundException
Activity	Copy-Item
⊞ Category	ObjectNotFound
Reason	ItemNotFoundException
TargetName	C:\data\customers
TargetType	String
ErrorDetails	
⊟ Exception	ItemNotFoundException: Cannot find path 'C:\data\customers' because it does not ex
⊞ Data	ListDictionaryInternal
⊟ ErrorRecord	Cannot find path 'C:\data\customers' because it does not exist.
⊞ CategoryInfo	ObjectNotFound: (C:\data\customers:String) [], ParentContainsErrorRecordException
ErrorDetails	
⊞ Exception	ParentContainsErrorRecordException: Cannot find path 'C:\data\customers' because i
FullyQualifiedErrorId	PathNotFound
InvocationInfo	
⊞ PipelineIterationInfo	Qt9DU9xSEKNyp39Rje2.84eLJ0xjodErCFWZnwA
TargetObject	C:\data\customers
HelpLink	
InnerException	
ItemName	C:\data\customers
Message	Cannot find path 'C:\data\customers' because it does not exist.
⊞ SessionStateCategory	Drive
Source	System.Management.Automation
StackTrace	at System.Management.Automation.LocationGlobber.ExpandMshGlobPath(String pa
⊞ TargetSite	System.Collections.ObjectModel.Collection`1[System.String] ExpandMshGlobPath(Sys
WasThrownFromThrowStatement	False
FullyQualifiedErrorId	PathNotFound,Microsoft.PowerShell.Commands.CopyItemCommand
⊟ InvocationInfo	InvocationInfo
⊞ BoundParameters	System.Collections.Generic.Dictionary`2[System.String,System.Object]
⊞ CommandOrigin	Internal
ExpectingInput	False
HistoryId	76
InvocationName	copy
Line	copy c:\data\customers c:\temp\customers -ea stop
⊞ MyCommand	Copy-Item
OffsetInLine	5
PipelineLength	1
PipelinePosition	1
PositionMessage	At C:\WPS\WPS2_trycatch.ps1:14 char:5+ copy <<<< c:\data\customers c:\temp\c
ScriptLineNumber	14
ScriptName	C:\WPS\WPS2_trycatch.ps1
⊞ UnboundArguments	Qt9DU9xSEKNyp39Rje2.84eLJ0xjodErCFWZnwA

Abbildung 7.11
Inhalt eines ErrorRecord-Objekts

8 Ausgaben

Dieses Kapitel behandelt die verschiedenen Möglichkeiten zur Erzeugung bzw. Beeinflussung von Ausgaben in der PowerShell.

8.1 Ausgabe-Commandlets

Commandlets mit Verben Out und Format

Ein normales Commandlet sollte keine eigene Bildschirmausgabe erzeugen, sondern allenfalls eine Menge von Objekten in die Pipeline legen. Es ist bestimmten Commandlets vorbehalten, eine Ausgabe zu erzeugen. Beispiele für diese Commandlets sind:

- `Out-Default`: Standardausgabe gemäß der PowerShell-Konfiguration (*DotNetTypes.Format.ps1xml*), siehe ein folgendes Kapitel
- `Out-Host`: wie `Out-Default` mit zusätzlicher Option zur seitenweisen Ausgabe
- `Out-Null`: Die Objekte der Pipeline werden nicht weitergegeben.
- `Out-GridView`: Ausgabe in grafischer Tabelle mit Such- und Filterfunktionen (ab PowerShell 2.0)
- `Format-Wide`: zweispaltige Liste
- `Format-List` (kurz: `fl`): detaillierte Liste
- `Format-Table` (kurz: `ft`): Tabellenausgabe

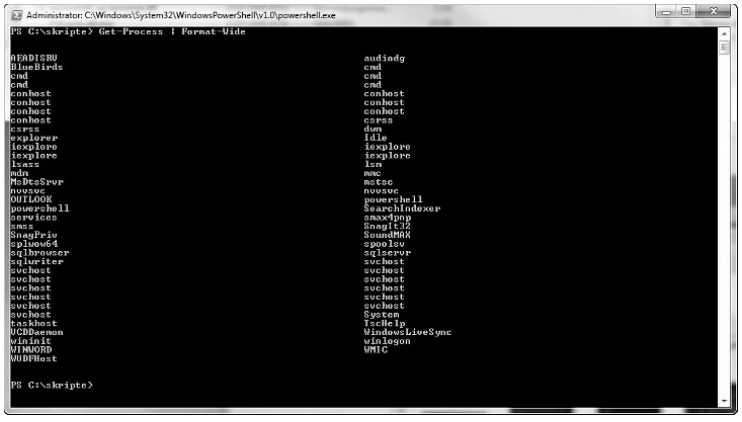


Abbildung 8.1 *Ausgabe mit Format-Wide*

Abbildung 8.2
Ausgabe mit Format-List

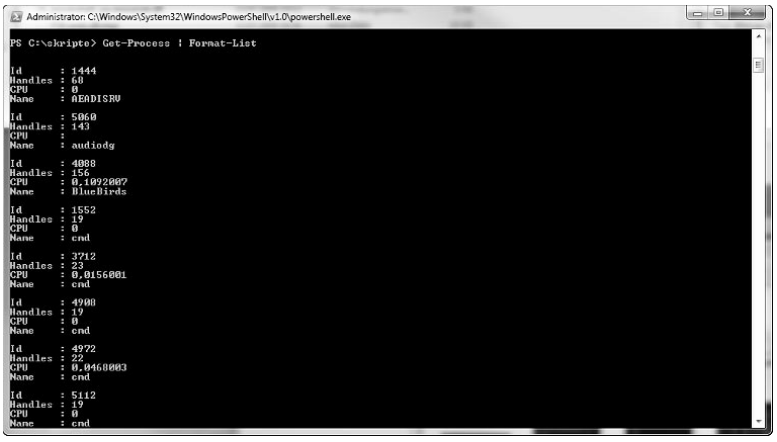
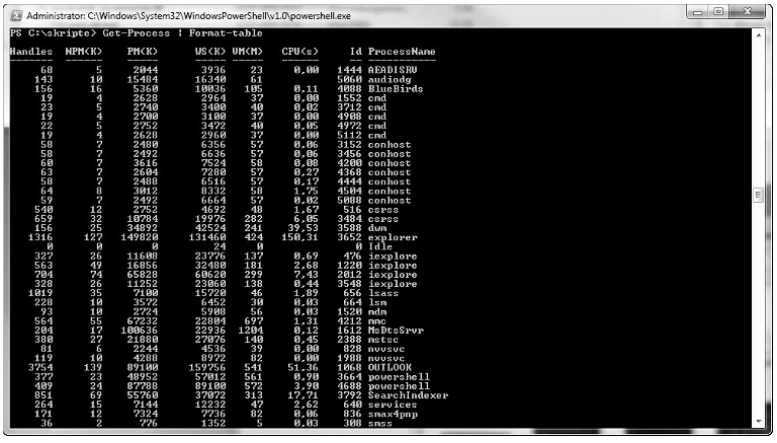


Abbildung 8.3
Ausgabe mit Format-Table





Bei den Ausgabe-Commandlets können Sie jeweils die auszugebenden Attribute mit dem Parameter `-Property` (kurz `-p`) festlegen. Dieser Parameter ist immer der erste erwartete Parameter und daher entfällt der Name in der Regel. Beispiele:

```
Get-Service | Format-Table Name, Status
Get-Service | Format-List Name, Status
```

Bei `Format-Wide` kann man nur ein Attribut nennen:

```
Get-Service | Format-Wide -Property Name
```



In vielen Fällen geben Ausgabe-Commandlets nicht alle Attribute eines Objekts aus. Dies liegt an der Standardausgabe (siehe gleichnamiges Kapitel). Sie können alle Attribute sehen, wenn Sie statt des Parameters `-Property` einen Stern (*) angeben.



***Abbildung 8.4**: Format-Table für Get-Service ohne und mit Stern*

8.2 Out-GridView

Tabellenansicht

Das Commandlet `Out-GridView` gab es schon einmal in der Beta-Version der PowerShell 1.0, es schaffte es dann aber nicht in das endgültige Produkt. Ab PowerShell 2.0 kehrt es zurück. `Out-GridView` zeigt den aktuellen Inhalt der Pipeline in einem separaten Fenster in einer grafischen Tabelle an und bietet dort Filtermöglichkeiten.

Beispiel: `Out-process | Out-GridView`

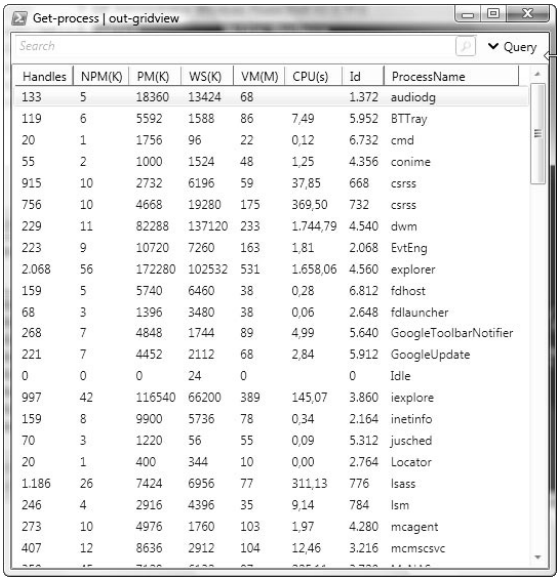
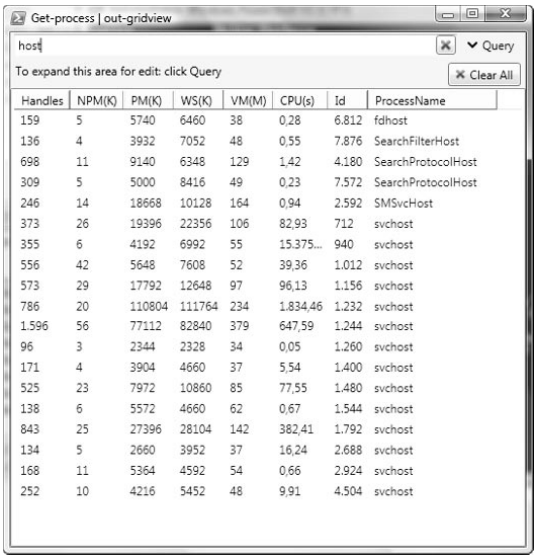


Abbildung 8.5
Anzeige der Prozessliste mit Out-GridView

Abbildung 8.6
Interaktive Suche in der Prozessliste mit Out-GridView

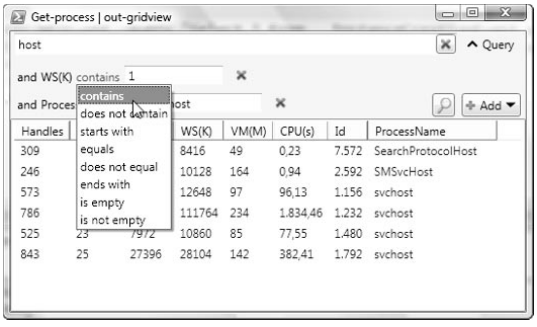


Suchen

`Search` führt eine Volltextsuche über alle Spalten aus. Über die Funktion „Query“ (oben rechts in der Ecke) kann man grundsätzlich auch komplexere Abfragen über einzelne Spalten definieren. Zumindest in der in diesem Test vorliegenden RC-Version gibt es aber nur Filterkriterien für Zeichenketten, nicht aber Vergleichsoperationen wie größer und kleiner für Zahlen.

In der endgültigen Version hat `Out-Gridview` eine Schaltfläche „Kriterien hinzufügen“ bekommen. Aus dem darüber erreichbaren Drop-down-Feld können Sie eine Auswahl aus allen Eigenschaften treffen. In der daraus resultierenden Zeile können die Bedingungen definiert werden.

Abbildung 8.7
Definition einer Abfrage im Out-GridView-Fenster



8.3 Standardausgabe

Standard-ausgabe

Wenn am Ende einer Pipeline keine Ausgabefunktion genannt ist, verwendet die PowerShell automatisch das Commandlet `Out-Default`. `Out-Default` bedient sich bei der Ausgabe einer Standardvorgabe

(„View“), die in der Datei *DotNetTypes.Format.ps1xml* im Installationsordner der PowerShell abgelegt ist. Dort kann man beispielsweise für den Typ `System.Diagnostics.Process` nachlesen, dass die Ausgabe in einer achtspaltigen Tabelle erfolgen soll (siehe *Abbildung 8.8*).

Wenn dort keine Standardvorgabe existiert, verwendet die PowerShell folgende Regeln:

- Alle Attribute (inkl. Noteproperties) werden ausgegeben.
- Die Ausgabe erfolgt mit `Format-Table`, solange es wenig Spalten sind.
- Bei mehr Spalten wird `Format-List` verwendet.



Abbildung 8.8 *Ausschnitt aus der Beschreibung der Standardausgabe für den Typ System.Diagnostics.Process in DotNetTypes.Format.ps1xml*

Views Die in `DotNetTypes.Format.ps1xml` abgelegten Views kann man mit dem Commandlet `Get-ViewDefinition` aus den PSCX ausgeben. Dabei ist zu beachten, dass die Views für .NET-Klassen und WMI-Klassen definiert sind, nicht für Commandlets. Anstelle von `Get-Process` muss man also den Rückgabetyp `System.Diagnostics.Process` angeben.

Beispiel: `Get-ViewDefinition.ps1 System.Diagnostics.Process`

Bei der Ausführung werden Sie feststellen, dass es Views für diese Klasse gibt: „process“, „starttime“ und „priority“ für die Tabellenausgabe und „process“ für das Ausgabeformat „wide“. In den Ausgabe-Commandlets wie `Format-Table` kann man die verschiedenen Views aufrufen. `Get-Process | Format-Table -view priority` liefert eine nach Prioritäten gruppierte Prozessliste.

Abbildung 8.9
Views für die Klasse „Process“

```
PS P:\1_Basiswissen\Scripting> Get-ViewDefinition.ps1 System.Diagnostics.Process


Name        : process
Path        : C:\Windows\System32\WindowsPowerShell\v1.0\DotNetTypes.format.ps1xml
TypeName    : System.Diagnostics.Process
SelectedBy  : System.Diagnostics.Process
GroupBy     :
Style       : Table

Name        : Priority
Path        : C:\Windows\System32\WindowsPowerShell\v1.0\DotNetTypes.format.ps1xml
TypeName    : System.Diagnostics.Process
SelectedBy  : System.Diagnostics.Process
GroupBy     : PriorityClass
Style       : Table

Name        : StartTime
Path        : C:\Windows\System32\WindowsPowerShell\v1.0\DotNetTypes.format.ps1xml
TypeName    : System.Diagnostics.Process
SelectedBy  : System.Diagnostics.Process
GroupBy     :
Style       : Table

Name        : process
Path        : C:\Windows\System32\WindowsPowerShell\v1.0\DotNetTypes.format.ps1xml
TypeName    : System.Diagnostics.Process
SelectedBy  : System.Diagnostics.Process
GroupBy     :
Style       : Wide
```



Die PowerShell gerät in einigen Situationen in Schwierigkeiten, wenn die verschiedenen Ausgabeformen miteinander kombiniert werden. Die folgende Befehlsfolge in einem Skript

```
Get-Service d*
Get-Process d* | ft
```

führt zum Fehler (siehe Bildschirmabbildung).

```
PS P:\> . 'H:\Deno\PowerShell\1_Basiswissen\Ausgaben\Ausgabeproblem.ps1'

Status   Name               DisplayName
------   ----               -----------
Running  DcomLaunch         DCOM Server Process Launcher
Stopped  defragsvc          Disk Defragmenter
Running  Dhcp               DHCP Client
Running  Dnscache           DNS Client
Stopped  dot3svc            Wired AutoConfig
Running  DPS                Diagnostic Policy Service
out-lineoutput : The object of type "Microsoft.PowerShell.Commands.Internal.Format.FormatStartData"
 is not valid or not in the correct sequence. This is likely caused by a user-specified "format-tab
le" command which is conflicting with the default formatting.
    + CategoryInfo          : InvalidData: (:) [out-lineoutput], InvalidOperationException
    + FullyQualifiedErrorId : ConsoleLineOutputOutOfSequencePacket,Microsoft.PowerShell.Commands.O
   utLineOutputCommand
```

***Abbildung 8.10**: Ausgabeproblem*

Die Lösung besteht in diesem Fall darin, `Format-Table` (`ft`) beide Male explizit zu nennen.

```
Get-Service d* | ft
Get-Process d* | ft
```

8.4 Einschränkung der Ausgabe

Die Ausgabebefehle erlauben die Spezifikation der Objektattribute, die ausgegeben werden sollen, z.B.:

```
Get-Process | Format-Table -p id,processname,workingset
```

erzeugt eine Tabelle der Prozesse mit Prozess-ID, Name des Prozesses und Speichernutzung. Attributnamen können dabei auch durch den Platzhalter * abgekürzt werden, z.B.:

```
Get-Process | Format-Table -p id,processn*,working*
```



Die gleiche Ausgabe können Sie auch erzielen, wenn Sie `Select-Object` einsetzen:

```
Get-Process | Select-Object id, processname, workingset | Format-Table
```

8.5 Seitenweise Ausgabe

Out-Host

Viele Ausgaben sind zu lang, um sie auf einer Bildschirmseite darstellen zu können. Manche Ausgaben sind sogar länger als der Puffer des PowerShell-Fensters (z.B. `Get-Command Get-Help`). Die seitenweise Ausgabe erzwingt man mit dem Parameter `-paging` (kurz –p) im `Out-Host`-Commandlet.

Beispiel:

```
Get-Service | Out-Host -p
```

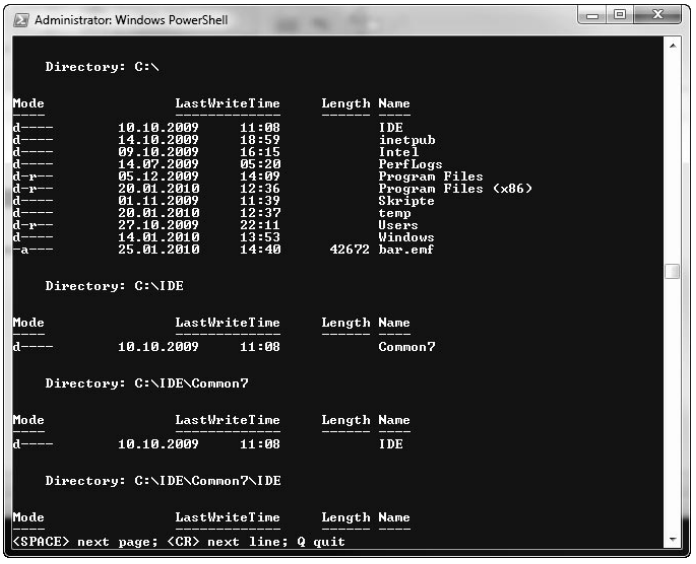


Abbildung 8.11 *Seitenweise Ausgabe mit Out-Host*



Es gibt alternativ auch eine Funktion `more`, die die gleiche Aufgabe zu erfüllen scheint. Allerdings ist die Nutzung im Einzelfall nicht so schön. `more` basiert nämlich auf dem alten `more.com`-Befehl und ist blockierend, d.h., Sie sehen die erste Ausgabeseite erst, wenn das letzte Objekt angekommen ist. `Out-Host -p` arbeitet hingegen asynchron. Versuchen Sie mal:

```
Get-ChildItem c:\ -Recurse | Out-Host -p
```

im Vergleich zu:

```
Get-ChildItem c:\ -Recurse | more
```

8.6 Ausgabe einzelner Werte

Write-Warn, Write-Error

Um ein bestimmtes Literal oder den Inhalt einer Variablen auszugeben, muss man diesen nur an der Konsole eingeben (siehe *Abbildung 8.12*). Auch alle Literale und Variablen, die „lose" in einem Skript stehen, zum Beispiel in der zweiten Zeile dieser Befehlsfolge

```
$name = "Holger"
$name
```

Alternativ kann man für eine Ausgabe von Literalen und Variablen explizit die Commandlets `Write-Host`, `Write-Warn` und `Write-Error` verwenden. `Write-Warn` und `Write-Error` erzeugen die Ausgabe hervorgehoben (im Standard gelb und rot).

Bei `Write-Host` kann man die Farben genau angeben:

```
Write-Host "Hallo Holger" -foregroundcolor red -backgroundcolor
white
```



Abbildung 8.12 *Ausgabe von Konstanten und Variablen*

Verknüpfen vs. Einbetten

Um in einer Ausgabe Literale und Variablen zu mischen, muss man diese entweder mit + verknüpfen:

```
$a + " ist erreichbar unter " + $b + ". Diese Information hat den
Stand: " + $c + "."
```

oder aber die Variablen direkt in die Zeichenkette einbetten. Im Gegensatz zu anderen Sprachen wertet die PowerShell die Zeichenkette aus und sucht dort nach dem Zeichen `$` (Variablenauflösung):

```
"$a ist erreichbar unter $b. Diese Information hat den Stand: $c."
```

Ausgabeoperator

Zur Zusammensetzung von Zeichenketten kann man in der PowerShell auch die in .NET gebräuchlichen Platzhalter und Formatkennzeichner (z.B. `d` = Datum in Langform) verwenden. Dafür ist nach der Zeichenkette der Ausgabeoperator `-f` zu benutzen. Diese Ausgabeoption ist aufgrund der Formatierungsmöglichkeiten die mächtigste:

```
"{0} ist erreichbar unter {1}. Diese Information hat den Stand:
{2:d}." -f $a, $b, $c
```

Das folgende Listing fasst die drei äquivalenten Möglichkeiten zusammen.

Listing 8.1
Formatierte Ausgabe [Ausgabe.ps1]

```
$a = "Holger Schwichtenberg"
$b = "hs@IT-Visions.de"
$c = Get-Date

# Möglichkeit 1
$a + " ist erreichbar unter " + $b + ". Diese Information hat den
Stand: " + $c + "."

# Möglichkeit 2
"$a ist erreichbar unter $b. Diese Information hat den Stand: $c."

# Möglichkeit 3
"{0} ist erreichbar unter {1}. Diese Information hat den Stand:
{2:D}." -f $a, $b, $c
```

Das obige Skript gibt Folgendes aus:

```
Holger Schwichtenberg ist erreichbar unter hs@IT-Visions.de. Diese
Information hat den Stand: 14.09.2006 16:53:13.
Holger Schwichtenberg ist erreichbar unter hs@IT-Visions.de. Diese
Information hat den Stand: 14.09.2006 16:53:13.
Holger Schwichtenberg ist erreichbar unter hs@IT-Visions.de. Diese
Information hat den Stand: Donnerstag, 14. September 2006.
```

8.7 Details zum Ausgabeoperator

Der Ausgabeoperator `-f` ist die mächtigste Form der Zusammensetzung von Zeichenketten aus verschiedenen Werten. Innerhalb der Zeichenkette legt man in geschweiften Klammern Platzhalter fest, wobei die Zählung mit 0 beginnt. Nach dem Operator `-f` führt man dann die einzelne Werte auf.

Beispiel: „Sehr geehrter Herr {0}, wir freuen uns, Sie in {1} zu begrüßen!“ -f $Name, $Ort

Platzhalter dürfen sich wiederholen. Dies bedeutet, dass aus

```
$Name = "Müller"
$Ort = "Essen"
"Sehr geehrter Herr {0}, wir freuen uns, Sie in {1} zu begrüßen!
Herr {0}, Sie werden in {1} vieles erleben!" -f $Name, $Ort
```

folgender Text entsteht

```
Sehr geehrter Herr Müller, wir freuen uns, Sie in Essen zu
begrüßen! Herr Müller, Sie werden in Essen vieles erleben!
```

Im Platzhalter kann man Formatierungen festlegen.

Der folgende Ausdruck würde eigentlich dies ausgegeben:

Ausdruck	Ausgabe
$Name = "Müller" $Geb = [DateTime] "8.9.1970" "Herr {0}, geboren am {1}, ist {2} Tage alt!" -f $Name, $Geb, [INT] ([DateTime]::Now-$Geb).TotalDays	Herr Müller, geboren am 09.08.1970 00:00:00, ist 14411 Tage alt!

Eine andere Formatierung könnte so aussehen:

Ausdruck	Ausgabe
$Name = "Müller" $Geb = [DateTime] "8.9.1970" "Herr {0,-10}, geboren am {1:d}, ist {2:0,000} Tage alt!" -f $Name, $Geb, [INT] ([DateTime]::Now-$Geb).TotalDays	Herr Müller, geboren am 09.08.1970, ist 14.411 Tage alt!

Zur Erläuterung: „:-10“ sorgt für einen Platz von mindestens 10 Zeichen, „:d“ reduziert die Anzeige auf das Datum und „:0,000“ erzeugt das Tausendertrennzeichen.



Bitte beachten Sie: Das Komma ist hier kein Fehler. Die Syntax orientiert sich an amerikanischen Standards. Dort ist der Punkt das Dezimaltrennzeichen und das Komma das Tausendertrennzeichen. Die Ausgabe orientiert sich immer an der aktuellen Systemeinstellung bzw. anderslautenden Vorgaben innerhalb der PowerShell.

Symbol	Beschreibung	Beispiel	Ergebnis
C	Währung	{0:c} –f 10000000	1.000.000,00 ?
d	Dezimalzahl	{0:d} –f 10000000	1000000
e	Wissenschaftlich	{0:e} –f 10000000	1,000000e+006
f	Festkommazahl	{0:f} –f 10000000	1000000,00
g	Generisch	{0:g} –f 10000000	1000000
n	Tausendertrennzeichen	{0:n} –f 10000000	1.000.000,00
x	Hexadezimal	{0:x4} –f 10000000	f4240
d	Kurzes Datumsformat	{0:d} –f [DateTime] "1.22.2010 00:44:15"	22.01.2010
D	Langes Datumsformat	{0:D} –f [DateTime] "1.22.2010 00:44:15"	Freitag, 22. Januar 2010
t	Kurzes Zeitformat	{0:t} –f [DateTime] "1.22.2010 00:44:15"	00:44
T	Langes Zeitformat	{0:T} –f [DateTime] "1.22.2010 00:44:15"	00:44:15
f	Datum und Uhrzeit komplett (kurz)	{0:f} –f [DateTime] "1.22.2010 00:44:15""	Freitag, 22. Januar 2010 00:44
F	Datum und Uhrzeit komplett (lang)	{0:F} –f [DateTime] "1.22.2010 00:44:15""	Freitag, 22. Januar 2010 00:44:15
g	Standarddatum (kurz)	{0:g} –f [DateTime] "1.22.2010 00:44:15"	22.01.2010 00:44
G	Standarddatum (lang)	{0:G} –f [DateTime] "1.22.2010 00:44:15"	22.01.2010 00:44:15
M	Tag des Monats	{0:M} –f [DateTime] "1.22.2010 00:44:15"	22 Januar
Y	Monat und Jahr	{0:Y} –f [DateTime] "1.22.2010 00:44:15"	Januar 2010
R	Datumsformat nach RFC1123	{0:r} –f [DateTime] "1.22.2010 00:44:15"	Fri, 22 Jan 2010 00:44:15 GMT
S	Sortierbares Datumsformat	{0:s} –f [DateTime] "1.22.2010 00:44:15"	2010-01-22T00:44:15
U	Universell sortierbares Datumsformat	{0:u} –f [DateTime] "1.22.2010 00:44:15"	2010-01-22 00:44:15Z
U	Universell sortierbares GMT-Datumsformat	{0:U} –f [DateTime] "1.22.2010 00:44:15"	Donnerstag, 21. Januar 2010 23:44:15

***Tabelle 8.1**: Vordefinierte Formatierungen*

Symbol	Beschreibung	Aufruf	Ergebnis
0	0-Platzhalter	{0:00.0000} –f 10000000	1000000,0000
#	Zahl-Platzhalter	{0:(#).##} –f 10000000	(1000000)
.	Dezimalpunkt	{0:0.0} –f 10000000	1000000,0
,	Tausendertrennzeichen	{0:0,0} –f 10000000	1.000.000
,.	Ganzzahliges Vielfaches von 1.000	{0:0,.} –f 10000000	1000
%	Prozentwert	{0:0%} –f 10000000	100000000%
e	Exponenten-Platzhalter	{0:00e+0} –f 10000000	10e+5
dd	Tag	{0:dd} –f [DateTime] "1.22.2010 00:44:15"	22
ddd	Tagname (kurz)	{0:ddd} –f [DateTime] "1.22.2010 00:44:15"	Fr
dddd	Tagname (lang)	{0:dddd} –f [DateTime] "1.22.2010 00:44:15"	Freitag
gg	Zeitalter	{0:gg} –f [DateTime] "1.22.2010 00:44:15"	n. Chr.
hh	Stunde 2stellig	{0:hh} –f [DateTime] "1.22.2010 00:44:15"	12
HH	Stunde 2stellig (24-Stunden)	{0:HH} –f [DateTime] "1.22.2010 00:44:15"	00
mm	Minute	{0:mm} –f [DateTime] "1.22.2010 00:44:15"	44
MM	Monat	{0:MM} –f [DateTime] "1.22.2010 00:44:15"	01
MMM	Monatsname (Kürzel)	{0:MMM} –f [DateTime] "1.22.2010 00:44:15"	Jan
MMMM	Monatsname (ausgeschrieben)	{0:MMMM} –f [DateTime] "1.22.2010 00:44:15"	Januar
ss	Sekunde	{0:ss} –f [DateTime] "1.22.2010 00:44:15"	15
yy	Jahr 2stellig	{0:yy} –f [DateTime] "1.22.2010 00:44:15"	10
yyyy	Jahr 4stellig	{0:YY} –f [DateTime] "1.22.2010 00:44:15"	2010
zz	Zeitzone (kurz)	{0:zz} –f [DateTime] "1.22.2010 00:44:15"	+01
zzz	Zeitzone (lang)	{0:zzz} –f [DateTime] "1.22.2010 00:44:15"	+01:00

***Tabelle 8.2**: Individualformatierungen*

8.8 Benutzerdefinierte Tabellenformatierung

Format-Table

Bei `Format-Table` kann man mit Hash-Tabellen angeben, wie die Spalten formatiert werden sollen. Für eine jede Spalte kann man eine Hash-Tabelle mit folgenden Angaben angeben:

- `Label`: Spaltenüberschrift
- `Expression`: beliebiger PowerShell-Ausdruck, der die Werte liefert, auch Berechnung
- `Width`: Spaltenbreite
- `Format`: Formatierungsbefehl

Es folgt ein Beispiel einer Tabelle mit drei Spalten:

- 5 Zeichen für die Prozessnummer in der ersten Spalte
- 20 Zeichen für den Prozessnamen
- 11 Zeichen für die Speichernutzung, wobei die Angabe in Megabyte erfolgt und mit maximal einer Nachkommastelle

```
Get-Process | sort workingset64 -desc | ft @{Label="Nr";
Expression={$_.ID}; Width=5}, @{Label="Name"; Expression={$_
.Processname}; Width=20 }, @{Label="Speicher MB"; Expression={$_
.WorkingSet64 / 1MB}; Width=11; Format="{0:00000.0}" }
```

```
PS P:\> . 'H:\Demo\PowerShell\1_Basiswissen\Pipelining\CustomFormatTable.ps1'

   Nr Name                 Speicher MB
   -- ----                 -----------
  888 explorer                 00477,8
  984 svchost                  00179,7
 6332 WINWORD                  00142,1
 2868 OUTLOOK                  00133,3
 1724 powershell               00086,3
 3976 SearchIndexer            00078,1
 7072 powershell               00073,4
 1548 sqlservr                 00066,3
 4696 iexplore                 00056,7
 6776 PowerShellPlus           00049,9
 2664 dwm                      00047,3
 1664 mstsc                    00027,5
 4168 WindowsLiveSync          00024,2
 4244 SnagIt32                 00020,3
 1012 svchost                  00019,9
 4352 iexplore                 00019,7
  600 csrss                    00019,4
 8172 audiodg                  00015,8
 5468 iexplore                 00015,6
 1100 iexplore                 00013,7
 5964 WINWORD                  00013,6
```

Abbildung 8.13
Ausgabe des obigen Befehls

8.9 Ausgabe von Methodenergebnissen und Unterobjekten in Pipelines

`GetType()`

Manchmal bekommt man Daten nicht aus direkten Eigenschaften (Attributen), sondern aus Methoden eines Objekts oder aus Attributen eines Unterobjekts. Ein häufiger Fall ist `GetType()`s, das Typinformationen über die .NET-Klasse in Form eines `System.Type`-Objekts lie-

fert. Nun kann man in den Ausgabe-Commandlets aber eigentlich nur Attribute angeben.

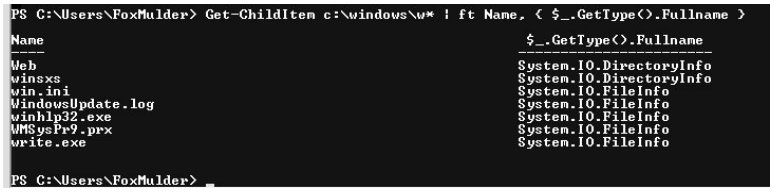
Die folgende Syntax ist nicht erlaubt:

```
Get-ChildItem c:\windows\w* | ft Name, GetType()
```

Stattdessen muss man etwas umständlicher den Aufruf von `GetType()` in einen Ausdruckblock verpacken und schreiben:

```
Get-ChildItem c:\windows\w* | ft Name, { $_.GetType() }
```

Abbildung 8.14
Aufruf von GetType() innerhalb von Format-Table (ft)



Ein weiteres Beispiel ist der Zugriff auf die Stundenzahl (Attribut Hours) im Attribut `TotalProcessorTime` der `Process`-Klasse.

Falsch:

```
Get-Process | ft ProcessName, .TotalProcessorTime.Hours
```

Richtig:

```
Get-Process | ft ProcessName, { $_.TotalProcessorTime.Hours }
```

8.10 Ausgabe von Methodenergebnissen und Unterobjekten in Zeichenketten

Wenn man eine Zeichenkette konstruieren möchte, dann ist die Syntax etwas anders. Man muss den Ausdruck in `$( … )` einbetten.

Gegeben sei folgendes Beispiel:

Listing 8.2
Anwendung von $(..)

```
# Der Prozess, der am meisten Speicher braucht
$p = Get-Process | Sort-Object -Property WorkingSet64 -Descending |
Select-Object -First 1
# Ausgabe Versuch 1
"Der Prozess $p.name braucht am meisten Speicher, nämlich [INT]
($p.WorkingSet64 / 1MB) !"
# Ausgabe Versuch 2
"Der Prozess {$p.name} braucht am meisten Speicher, nämlich {[INT]
($p.WorkingSet64 / 1MB)} !"
# Ausgabe Versuch 3
"Der Prozess $($p.name) braucht am meisten Speicher, nämlich
$([INT] ($p.WorkingSet64 / 1MB)) MB !"
```

Hier liefert nur der dritte Versuch das gewünschte Ergebnis. Bei Versuch 1 und 2 wird der Ausdruck ausgegeben statt ausgewertet.



Die abweichende Schreibweise ist notwendig, da `{..}` in Zeichenketten schon Platzhalter für den Formatoperator darstellt. In folgendem Fall (Einsatz von `-f`) ist `$(…)` nicht notwendig:

```
# Ausgabe Versuch 4
"Der Prozess {0} braucht am meisten Speicher, nämlich {1} MB !"
-f  $p.name, [INT] ($p.WorkingSet64 / 1MB)
```

8.11 Unterdrückung der Ausgabe

Out-Null, void

Die Existenz der Standardausgabe sorgt dafür, dass alle Rückgabewerte von Commandlet-Pipelines auch ausgegeben werden. Dies ist nicht immer erwünscht.

Es gibt vier Alternativen, die Ausgabe zu unterdrücken:

- Am Ende der Pipeline wird `Out-Null` verwendet:
  ```
  Commandlet | Commandlet | Out-Null
  ```
- Das Ergebnis der Pipeline wird einer Variablen zugewiesen:
  ```
  $a = Commandlet | Commandlet
  ```
- Das Ergebnis der Pipeline wird auf den Typ `[void]` konvertiert:
  ```
  [void] (Commandlet | Commandlet)
  ```
- Das Ergebnis der Pipeline wird an `$null` zugewiesen:
  ```
  $null = Commandlet | Commandlet
  ```

8.12 Weitere Ausgabefunktionen

Die folgende Liste zeigt weitere Ausgabemöglichkeiten:

Out-Printer, Out-File

- Ausgaben zum Drucker sendet man mit dem Commandlet `Out-Printer`. Mit `Out-File` schreibt man den Inhalt in eine Datei.
- Ausgabe der Prozessliste auf den Standarddrucker:
  ```
  Get-Process | Out-Printer
  ```
- Ausgabe der Prozessliste auf einen bestimmten Drucker:
  ```
  Get-Process | Out-Printer "HP LaserJet on E02"
  ```
- Ausgabe der Prozessliste in eine Textdatei (mit Überschreiben des bisherigen Inhalts):
  ```
  Get-Process | Out-File "c:\temp\prozessliste.txt"
  ```
- Ausgabe der Prozessliste in eine Textdatei (Anhängen an bisherigen Inhalt):
  ```
  Get-Process | Out-File "c:\temp\prozessliste.txt" -Append
  ```

9 Eingabefunktionen

Texteingaben vom Benutzer kann man durch `Read-Host` entgegennehmen.

Read-Host

```
PS C:\Documents\hs> $name = read-host "Bitte Benutzernamen
eingeben"
Bitte Benutzernamen eingeben: HS
PS C:\Documents\hs> $kennwort = read-host -assecurestring "Bitte
Kennwort eingeben"
Bitte Kennwort eingeben: ****
```

Ein einfaches Eingabefeld stellt die bereits aus Visual Basic/VBScript bekannte Funktion `InputBox()` dar. Diese Funktion existiert auch noch im .NET Framework in der Klasse `Microsoft.VisualBasic.Interaction`. Zur Nutzung der Funktion muss erst die *Assembly Microsoft.VisualBasic.dll* geladen werden.

Grafischer Eingabedialog

```
[System.Reflection.Assembly]::LoadWithPartialName("Microsoft.Visual
Basic")
$eingabe = [Microsoft.Visual Basic.Interaction]::InputBox("Bitte
geben Sie Ihren Namen ein!")
"Hallo $Eingabe!"
```

Listing 9.1 *Einfache grafische Dateneingabe in der PowerShell[Aufbauwissen/DOTNET/InputBox1.ps1]*

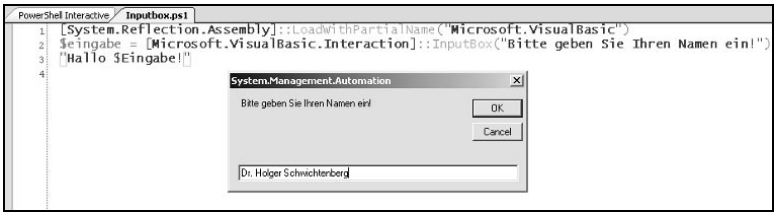


Abbildung 9.1 *Start des PowerShell-Skripts aus der PowerShellIDE heraus*

Dialogfenster

Für Dialogfenster kann man auf .NET-Klassen zurückgreifen. Das folgende Skript bittet den Anwender über ein Dialogfenster um eine Entscheidung (Ja/Nein):

MessageBox

Listing 9.2 *Nutzung der Klasse MessageBox in der PowerShell[Aufbauwissen/DOTNET/MessageBox.ps1]*

```
[System.Reflection.Assembly]::LoadWithPartialName("System.windows.
forms")
[System.Console]::Beep(100, 50)
[System.Windows.Forms.MessageBox]::Show("Gleich kommt eine
Frage","Vorwarnung", [System.Windows.Forms.MessageBoxButtons]::OK)

$antwort =
[System.Windows.Forms.MessageBox]::Show("Nachricht","Ueberschrift",
[System.Windows.Forms.MessageBoxButtons]::YesNo)
if ($antwort -eq "Yes")
{ "Sie haben zugestimmt!" }
else
{ "Sie haben abgelehnt!" }
```

Authentifizierungsdialog

Einen Windows-Authentifizierungsdialog (siehe *Abbildung 9.2*) öffnet die PowerShell mit `Get-Credential`. Das Ergebnis ist eine Instanz von `System.Management.Automation.PSCredential` mit dem Benutzernamen im Klartext in `UserName` und dem Kennwort verschlüsselt in `Password`.

Abbildung 9.2 *Einsatz von Get-Credential*





Sie bekommen auch dann ein `PSCredential`-Objekt, wenn die Eingabe falsch war. Zur konkreten Nutzung der Daten zur Authentifizierung sind andere Verfahren notwendig. Insbesondere werden Commandlets und Klassen benötigt, welche die verschlüsselten Kennwörter verarbeiten können.

10 Das PowerShell-Navigationsmodell

Get-PSDrive

Neben dem Objekt-Pipelining wartet die PowerShell noch mit einem interessanten Administrationskonzept auf: dem einheitlichen Navigationsparadigma für alle Arten von Datenmengen. Beim Aufruf des Befehls `Get-PSDrive` zeigen sich nicht nur die erwarteten Laufwerke, sondern auch Umgebungsvariablen (`env`), die Registrierungsdatenbank (`HKCU`, `HKLM`), der Windows-Zertifikatsspeicher (`cert`) die PowerShell-Aliase (`Alias`), PowerShell-Variablen (`Variable`) und PowerShell-Funktionen (`Function`). Die PowerShell fasst auch diese Daten als Laufwerke auf. Konsequenterweise muss man beim Aufruf auch einen Doppelpunkt verwenden: `Get-ChildItem Alias:` listet genau wie `Get-Alias` alle definierten Aliase auf.

10.1 Navigation in der Registrierungsdatenbank

In der Registrierungsdatenbank kann der Administrator somit mit den gleichen Befehlen wie im Dateisystem arbeiten. Beispiele für gültige Registrierungsdatenbankbefehle sind:

- Navigation zu *HKEY_LOCAL_MACHINE/Software*:

  ```
  cd hklm:\software
  ```

  Kurzform für:

  ```
  Set-Location hklm:\software
  ```

- Auflisten der Unterschlüssel des aktuellen Schlüssels:

  ```
  Dir
  ```

  Kurzform für:

  ```
  Get-ChildItem
  ```

- Erzeugen eines Unterschlüssels mit Namen „IT-Visions“:

  ```
  md IT-Visions
  ```

- Erzeugen eines Unterschlüssels mit einem Standardwert:

  ```
  New-Item -Name "Website" -Value "www.IT-Visions.de" -type
  String
  ```

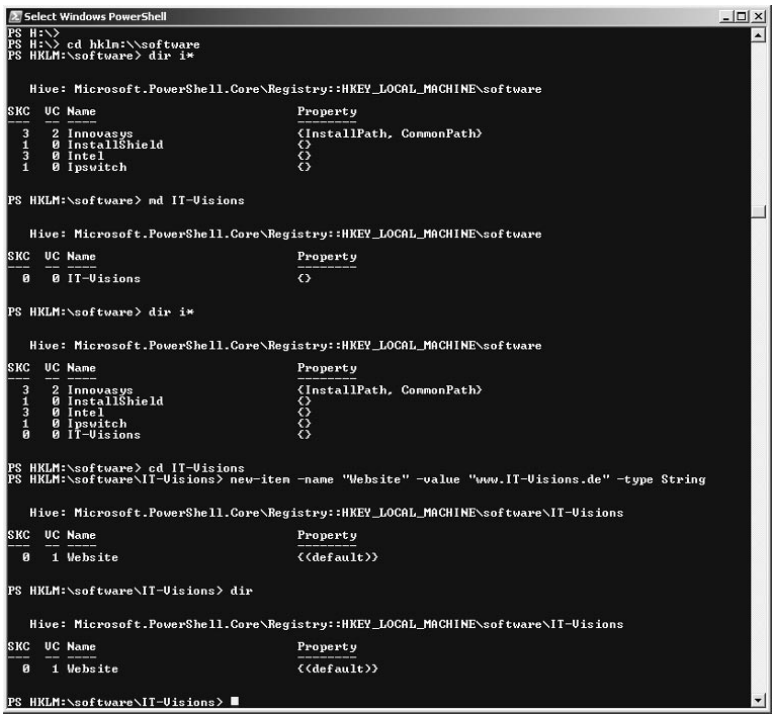


Abbildung 10.1 Navigation in und Manipulation der Registrierungsdatenbank

10.2 Provider und Laufwerke

Provider

`Get-PSDrive` zeigt an, dass es verschiedene „Laufwerk"-Provider gibt. Die Festplatten gehören zum Provider „FileSystem" (FS). Microsoft nennt die Provider „Navigation Provider" oder „Commandlet Provider" und will alle Datenmengen, egal ob flach oder hierarchisch, mit den gleichen Basisverben (`Get`, `Set`, `New`, `Remove` etc.) behandeln. Sowohl die Menge der Provider als auch die Menge der Laufwerke ist erweiterbar.

In der PowerShell 2.0 sind folgende Laufwerke enthalten:

- Windows-Dateisystem (`A`, `B`, `C`, `D`, `E` etc.)
- Windows-Registrierungsdatenbank (`HKCU`, `HKLM`)
- Windows-Umgebungsvariablen (`env`)
- Windows-Zertifikatsspeicher (`cert`)
- Funktionen der PowerShell (`function`)
- Variablen der PowerShell (`variable`)
- Aliase der PowerShell (`alias`)
- WSMan Webdienstkonfiguration und Verwaltung (wsman)

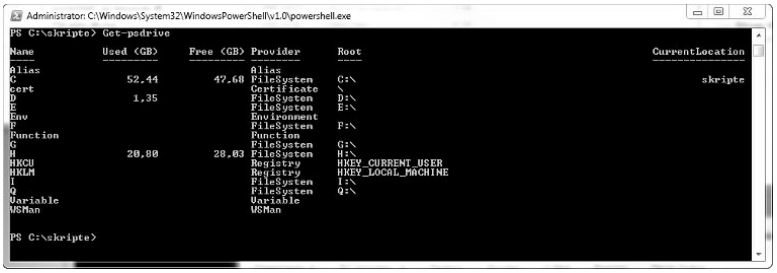


Abbildung 10.2
Aus Sicht der PowerShell sind Laufwerke auch die Umgebungsvariablen, die Aliase und die Registrierungsdatenbankeinträge.

Auch das Active Directory kann man diesem Navigationsparadigma unterwerfen. In den frühen Beta-Versionen der PowerShell war ein Provider dafür auch enthalten; er hat es aber nicht in die endgültige Version geschafft. Der Active Directory-Provider ist jetzt aber als Modul über die Remote Server Administration Tools (RSAT) bzw. als Bestandteil der Serverrolle Active Directory Directory Services (AD DS) enthalten bzw. für PowerShell 1.0 als Teil der PowerShell Community Extensions (PCSX) erhältlich.

Die installierten Provider sieht man mit `Get-PSProvider`.



Provider	Quelle	Laufwerke
Alias	PowerShell 1.0/2.0	Alias
Environment	PowerShell 1.0/2.0	Env
Filesystem	PowerShell 1.0/2.0	A, B, C, D etc.
Function	PowerShell 1.0/2.0	Function
Registry	PowerShell 1.0/2.0	HKLM, HKCU
Variable	PowerShell 1.0/2.0	Variable
Certificate	PowerShell 1.0/2.0	cert
RSS-Feedstore	PCSX 1.1.1 [CODEPLEX01]	Feed
Assemblycache	PCSX 1.1.1 [CODEPLEX01]	Gac
Directoryservices	PCSX 1.1.1 [CODEPLEX01]	(NT 4.0-kompatbiler Name der Domäne)
Active Directory	Active Directory-PowerShell-Modul in Windows Server 2008 R2 bzw. im Microsoft Remote Server Administration Tools (RSAT) für Windows 7	AD:

***Tabelle 10.1**: Verfügbare PowerShell-Provider*

Provider	Quelle	Laufwerke
Internet Information Server	PowerShell-Modul »WebAdministration« in Windows Server 2008 R2 bzw. im Microsoft Remote Server Administration Tools (RSAT) für Windows 7	IIS :
Microsoft SQL Server	Microsoft SQL Server 2008	SQLSERVER:
Windows SharePoint Services oder SharePoint Portal Server	PowerShell SharePoint Provider [CODEPLEX02].	(Beliebiger Name)

***Tabelle 10.1**: Verfügbare PowerShell-Provider (Forts.)*

10.3 Navigationsbefehle

Zur Navigation stehen folgende Befehle zur Verfügung:

***Tabelle 10.2** Navigationsbefehle*

Commandlet	Aliase	Beschreibung
`Get-ChildItem`	`dir, ls`	Auflisten der Elemente
`Get-Cwd`	`cd, pwd`	Wechsel des Standorts
`Get-Content`	`type, cat`	Abruf eines Elementinhalts
`New-Item`	`mkdir`	Erstellen eines Elements (Ast oder Blatt)
`Get-Location`		Abrufen des aktuellen Standorts
`Set-Location`	`Cd`	Festlegung des aktuellen Standorts

10.4 Pfadangaben

Pfadangaben in der PowerShell unterstützen verschiedene Platzhalter, insbesondere:

- Ein Punkt (`.`) steht für den aktuellen Ordner.
- Zwei Punkte (`..`) stehen für den übergeordneten Ordner.
- Die Tilde (~) steht für das Profilverzeichnis des aktuellen Benutzers (siehe *Abbildung 10.3*).
- Eine eckige Klammer steht für eines der Zeichen in der Klammer.

Beispiel: Der folgende Befehl listet alle Dateien aus dem Windows-Verzeichnis auf, die mit a, b, c oder w beginnen: `Get-ChildItem c:\windows\[abcw]*.*`.

Alternativ kann man dies auch schreiben: `Get-ChildItem c:\windows\[a-cw]*.*`.

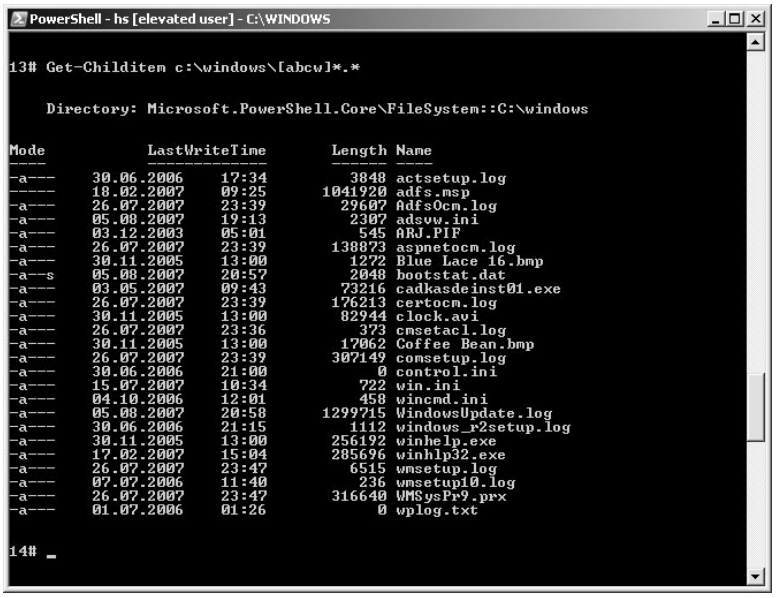


Abbildung 10.3
Einsatz von Platzhaltern

Es gibt mehrere Commandlets, die Unterstützung bei der Navigation in PowerShell-Laufwerken bieten.

Test-Path

`Test-Path` prüft, ob es einen Pfad gibt. Das Ergebnis ist `True` oder `False` (`System.Boolean`).

```
Test-Path c:\temp
Test-Path HKLM:\software\IT-Visions
```

Resolve-Path

`Resolve-Path` löst Platzhalter in Pfadangaben auf und gibt den resultierenden Pfad als ein Objekt vom Typ `System.Management.Automation.PathInfo` zurück.

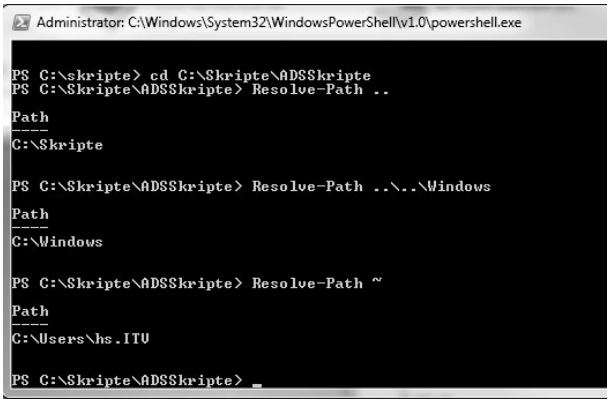


Abbildung 10.4
Einsatz von Resolve-Path

Convert-Path Viele Commandlets geben Pfadangaben des Typs `System.Management.Automation.PathInfo` aus. Um dies in eine einfache Zeichenkette (die allerdings dann providerspezifisch ist) umzusetzen, steht das Commandlet `Convert-Path` zur Verfügung.

10.5 Beispiel

Das nachstehende Listing zeigt ein Skript, das eine Hierarchie von Schlüsseln in der Registrierungsdatenbank ablegt. Dabei wird absichtlich die einfache Addition von Zahlen in eine Unterroutine gekapselt, um die Rückgabe von Werten an den Aufrufer mit der `return`-Anweisung zu zeigen. Literale und Ausdrücke, die ohne Commandlet im Skript enthalten sind, werden an der Konsole ausgegeben.

Listing 10.1 *Ein PowerShell-Skript zur Manipulation der Registrierungsdatenbank [Einsatzgebiete/Registry/Registry_Commandlets.ps1]*

```
########################################
# PowerShell-Skript
# Das Skript legt eine Schlüsselhierarchie in der
Registrierungsdatenbank an.
# (C) Dr. Holger Schwichtenberg
########################################

# === Unterroutine, die eine Addition ausführt
function Addition
{
return $args[0] + $args[1]
}

# === Unterroutine, die einen Schlüssel in der
Registrierungsdatenbank anlegt
function CreateEntry
{
"Eintrag anlegen..."

New-Item -Name ("Eintrag #{0}" -f $args[0]) -value $args[1] -type
String

}

# === Hauptroutine
"PowerShell-Registrierungsdatenbank-Skript (C) Dr. Holger
Schwichtenberg 2006"

# Navigation in die Registrierungsdatenbank
cd hklm:\software

# Prüfe, ob Eintrag \software\IT-Visions vorhanden
$b = Get-Item IT-Visions
if ($b.childName -eq "IT-Visions")
```

```
{ # Lösche vorhandenen Eintrag mit allen Unterschlüsseln
"Schluessel existiert bereits, loesche..."
cd hklm:\software
del IT-Visions -force -recurse
}
# Erzeuge neuen Eintrag "IT-Visions"
"Erzeuge IT-Visions..."
md IT-Visions
cd IT-Visions

# Lege Unterschlüssel an
for($a=1;$a -lt 5;$a++)
{
$ergebnis =  Addition $a $a
CreateEntry $a $ergebnis
}
```

10.6 Eigene Laufwerke definieren

New-PSDrive

Das Navigationsmodell der PowerShell erlaubt die Definition eigener Laufwerke, die dann als Abkürzung verwendet werden können.

Der folgende Befehl definiert ein neues Laufwerk „Skripte:“ als Alias für einen Dateisystempfad:

```
New-PSDrive -Name Skripte -PSProvider FileSystem -Root "h:\Skripte\
ps\"
```

Danach kann man mit

```
Dir Skripte:
```

auf den Pfad zugreifen.



Das neu definierte Laufwerk funktioniert nur innerhalb der PowerShell und ist nicht in sonstigen Windows-Anwendungen verfügbar. Genau genommen funktioniert das neue Laufwerk sogar nur innerhalb der aktuellen Instanz der PowerShell. Zwei PowerShell-Fenster teilen sich nicht solche Deklarationen!

Auch für die Registrierungsdatenbank kann man solche Abkürzungen definieren:

```
New-PSDrive -Name Software -PSProvider Registry -Root HKLM:\
SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall
```

Die Anzahl der Laufwerke ist im Standard auf 4096 beschränkt. Dies kann durch die Variable `$MaximumDriveCount` geändert werden.

11 PowerShell-Werkzeuge

Dieses Kapitel bespricht die von Microsoft gelieferte PowerShell-Konsole sowie nützliche Werkzeuge anderer Anbieter. Microsoft liefert erst mit der Version 2 einen Editor für PowerShell-Skripte.

11.1 PowerShell-Konsole

Eingabefenster

Die PowerShell-Konsole basiert auf der normalen Windows-Konsole („Eingabeaufforderung"). Sie bietet etwas mehr Eingabeunterstützung als das Kommandozeilenfenster, vom Komfort eines Visual Studio ist die PowerShell-Konsole aber weit entfernt.

Funktionsumfang der Konsole

Die PowerShell-Konsole bietet folgende Funktionen:

- Größe und Aussehen des Fensters können über die Eigenschaften gesteuert werden (siehe *Abbildung 11.1*).
- Die Zwischenablage steht genauso umständlich wie zuvor nur über das Menü zur Verfügung (siehe *Abbildung 11.2*) bzw. den sogenannten „Quick Edit Mode". Die Tastenkombinationen [STRG]+[C] / [X] / [V] funktionieren nicht.
- Befehls- und Pfadeingaben sowie Objektattribute können mit der [↹]-Taste vervollständigt werden.
- Ein Rücksprung zu den letzten 64 Befehlen (Anzahl änderbar) ist möglich.
- Die letzten Befehle werden durch die Taste [F7] angezeigt.
- Aufruf des letzten Befehls komplett durch die Taste [F3] bzw. zeichenweise durch [F1]
- Der Abbruch eines laufenden Befehls ist mit [STRG]+[C] möglich.

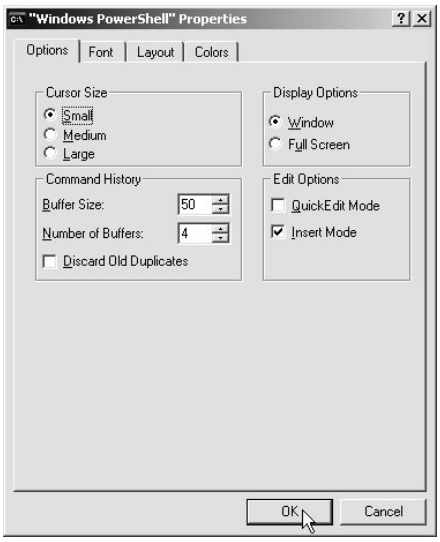


Abbildung 11.1 *Fenstereigenschaften für das PowerShell-Konsolenfenster*

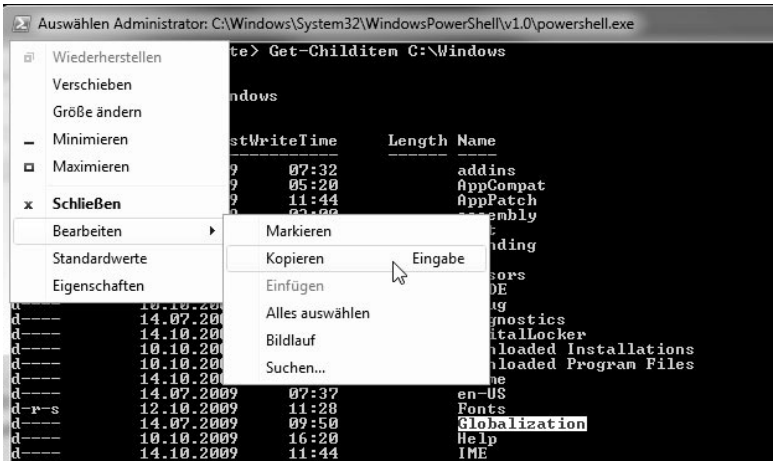


Abbildung 11.2 *Nutzung der Zwischenablage im PowerShell-Konsolenfenster*

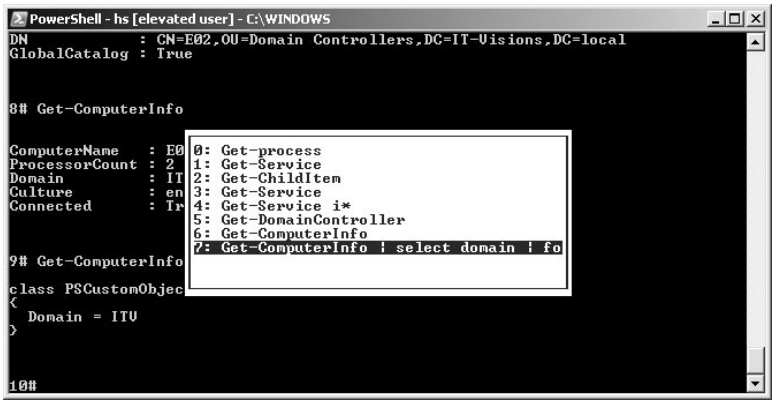


Abbildung 11.3 *Anzeige der Befehlsgeschichte mit F7*

Tabulatorvervollständigung

„Tab Completion“

Die PowerShell kennt eine bereits im klassischen Kommandozeilenfenster verfügbare Funktion auch für Commandlets, Parameter und Objektattribute. Im DOS-Kommandozeilenfenster kann man nach Eingabe eines oder mehrerer Buchstaben die erreichbaren Dateien und Unterverzeichnisse mit der Tabulator-Taste ⇆ durchlaufen (in der Entwicklersprache „Tab Completion“ genannt). In der PowerShell funktioniert das auch bei den Commandlets, deren Parameter und den Attributen von Objekten in der Pipeline (siehe *Abbildung 11.4* ff.).

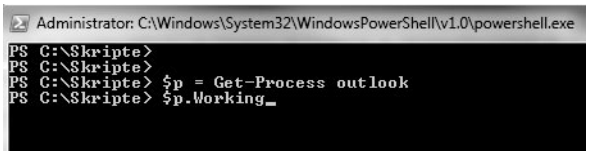


Abbildung 11.4
Eingabe des Wortanfangs

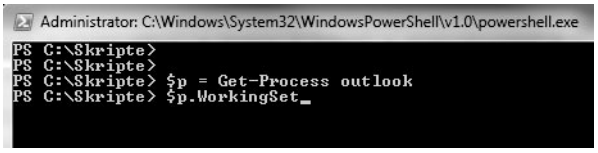


Abbildung 11.5
Nach dem Drücken der Tabulator-Taste erscheint die erste Alternative.

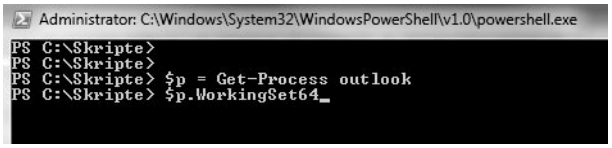


Abbildung 11.6
Nach nochmaligem Drücken erscheint die zweite Alternative.

Kommandomodus versus Interpretermodus

Normalerweise führt die Konsole alle Befehle nach dem Drücken der Taste ↵ sofort aus. Wenn man allerdings einen unvollständigen Befehl eingibt (z.B. einen Befehl, der auf dem Pipeline-Symbol | endet), dann geht die Konsole in den sogenannten Interpretermodus, bei dem die Befehle nicht mehr sofort ausgeführt werden. Der Interpretermodus wird durch die Eingabeaufforderung >> (siehe *Abbildung 11.7*) angezeigt. Der Interpretermodus gilt so lange, bis man eine leere Eingabe macht. Dann wird der Befehl ausgeführt.



Abbildung 11.7
Die Konsole ist im Interpretermodus.



Abbildung 11.8 *Der Interpretermodus wurde durch eine leere Eingabe wieder verlassen.*

Benutzerkontensteuerung in Windows Vista/ Windows 7/Windows Server 2008 (inkl. R2)

Die Windows PowerShell unterliegt wie alle anderen Anwendungen auch der Benutzerkontensteuerung in den neueren Windows-Versionen und wird daher unter eingeschränkten Rechten gestartet. Um die PowerShell mit vollen Rechten zu starten, können Sie im Kontextmenü die Funktion ALS ADMINISTRATOR AUSFÜHREN wählen. Danach werden Windows Vista und Windows Server 2008 R2 um die Bestätigung der Rechteerhöhung bitten (siehe *Abbildung 11.9*). Windows 7 und Windows Server 2008 R2 fragen bei den Standardeinstellungen der Benutzerkontensteuerung nicht nach.



Abbildung 11.9 *Zustimmung zur Rechteerhöhung der PowerShell (hier in Windows Vista)*

Anders als die normale Windows Shell zeigt die PowerShell danach nicht durch einen Eintrag in der Titelleiste an, dass sie nun unter administrativen Rechten läuft. Dies funktioniert erst ab Windows 7 und Windows Server 2008 R2.

Um in der Titelleiste der PowerShell-Konsole den Rechtestatus anzuzeigen und gegebenenfalls weitere Anpassungen der Anzeige vorzunehmen, wie man dies in der folgenden Bildschirmabbildung sieht, kann man sich ein PowerShell-Profilskript schreiben. Die Erstellung eines solchen Skripts und das Skript zur Anzeige des Rechtestatus in der Titelleiste finden Sie im *Abschnitt 27.5 „Profileinstellungen für die PowerShell-Konsole“*.

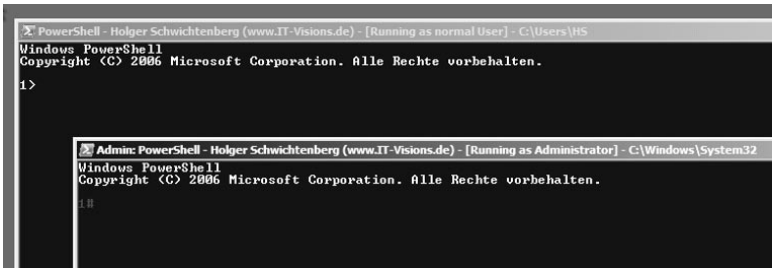


Abbildung 11.10 *Zwei PowerShell-Instanzen mit verschiedenen Rechten*

Um bei einer laufenden Konsole ohne Titelanzeige festzustellen, welche Rechte diese besitzt, können Sie das in Windows mitgelieferte Kommandozeilenwerkzeug *whoami.exe* mit der Option `/all` verwenden.

11.2 PowerShell Integrated Scripting Environment (ISE)

ISE

Die PowerShell 1.0 bot als einziges Werkzeug zum Eingeben und Ausführen von PowerShell-Befehlen und -Skripten die klassische Windows-Konsole. Ein komfortabler Editor fehlt und rief Drittanbieter auf den Plan, z.B. PowerShellPlus und PowerGUI. Das PowerShell Integrated Scripting Environment (ISE), in den ersten Alpha-Versionen der PowerShell 2.0 noch „Graphical PowerShell“ genannt, ist nun eine Windows-Anwendung, die einen Editor und eine Ausführungsumgebung für die PowerShell bereitstellt.

Bereiche

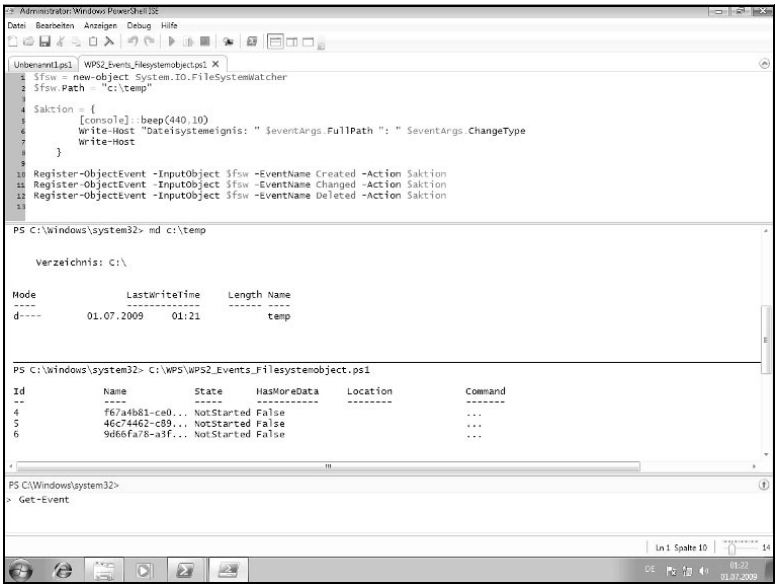
Die ISE bietet drei Bereiche:

- Eingabebereich für Skripte (Skripteditor mit Syntaxfarbhervorhebung und Tabulatorvervollständigung): Hier eingegebene Befehle werden erst ausgeführt, wenn das Skript explizit gestartet wird.
- Eingabebereich für Einzelbefehle (interaktiver Bereich mit Syntaxfarbhervorhebung und Tabulatorvervollständigung): Hier eingegebene Befehle werden nach Drücken der Eingabe-Taste sofort ausgeführt.
- Ausgabebereich, der von beiden Eingabebereichen verwendet wird

Zum Start der ISE ruft man entweder das Symbol im Startmenü, die ausführbare Datei *PowerShell_ise.exe* oder in der PowerShell den Alias „ise“ auf.

Im Standard startet ISE mit dem Ausgabebereich in der Mitte. Man kann aber durch das Menü ANZEIGEN Einfluss auf die Anordnung nehmen, z.B. den Skripteditor ausblenden und den interaktiven Eingabebereich nach oben setzen. Eine flexible Fensteranordnung wie in Visual Studio oder dem SQL Server Management Studio ist hier aber nicht möglich. Alternativ können Sie die Schaltflächen unterhalb der Menüzeile benutzen.

Abbildung 11.11
Die ISE mit den drei Bereichen



Unterschiede zur normalen PowerShell-Konsole

Es gibt ein paar Unterschiede zwischen ISE und der normalen PowerShell-Konsole.

Interaktive klassische Konsolenanwendungen wie ftp.exe werden durch die ISE nicht unterstützt. Kein Problem gibt es dagegen mit interaktiven PowerShell-Commandlets und -Skripten.

Das Blättern in Ausgaben mit „more“ funktioniert nicht.

Man kann die Farbe der Konsole nicht über die .NET-Klasse [`System.Console`] setzen.

Nicht möglich ist also:

```
[console]::BackgroundColor  = 'red'
```

Richtig ist stattdessen die Verwendung des Objektmodells der ISE, das man über `$psISE` erreicht:

```
$psISE.Options.ScriptPaneBackgroundColor = "red";
```

Über die Angabe „white“ stellen Sie die Ausgangslage wieder her.

Auch der Zugang über `$host.UI.RawUI` ist eingeschränkt. Möglich ist hierüber aber z.B. die Veränderung des Titels:

```
$host.UI.RawUI.WindowTitle = "Holgers PowerShell IDE".
```

Die ISE hat ein eigenes Startprofilskript: `Microsoft.PowerShellISE_profile.ps1`.

Eventuell existiert diese Datei auf Ihrem System noch nicht. Geben Sie zu ihrer Erstellung in der ISE `notepad $profile` ein. Notepad wird Sie fragen, ob diese Datei erstellt werden soll.

In Windows PowerShell ISE können die folgenden Profile erstellt und verwendet werden. Jedes Profil wird in einem eigenen spezifischen Pfad gespeichert.

Profiltyp	Profilpfad
Aktueller Benutzer, PowerShell ISE	$profile.CurrentUserCurrentHost oder $profile
Alle Benutzer, PowerShell-ISE	$profile.AllUsersCurrentHost
Aktueller Benutzer, alle Hosts	$profile.CurrentUserAllHosts
Alle Benutzer, alle Hosts	$profile.AllUsersAllHosts

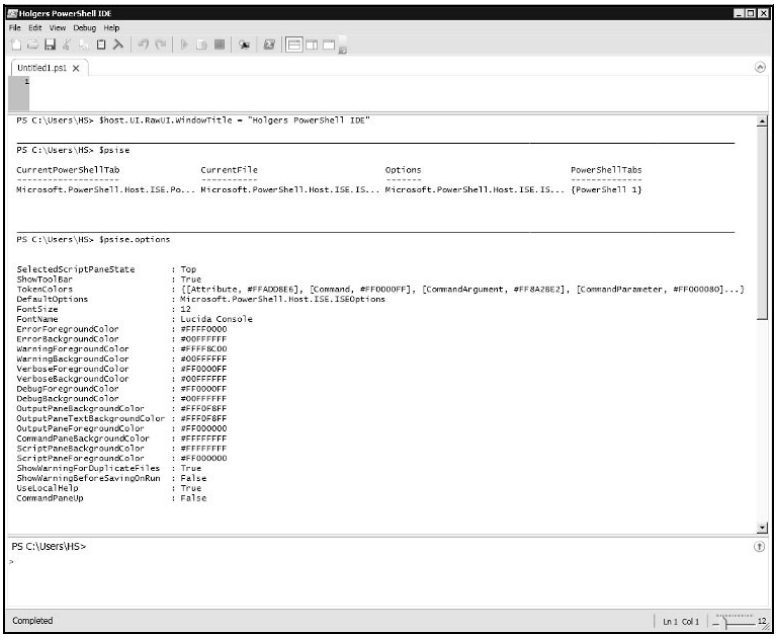


Abbildung 11.12 *Inhalt der eingebauten Variablen $psISE*

11.3 PowerShellPlus

Host und Editor PowerShellPlus ist die kommerzielle Weiterentwicklung der früheren PowerShell IDE. PowerShellPlus besteht aus einer verbesserten PowerShell-Konsole (PowerShellPlus Host), die direkt IntelliSense unterstützt, und einem damit verbundenen Editor (PowerShellPlus Editor).

PowerShellPlus	
Hersteller:	Idera
Preis:	145 USD
URL:	*http://www.idera.com/Products/PowerShell/*
Enthalten auf Buch-CD?	Ja (Testversion)

Funktionen Bemerkenswerte Funktionen sind:

- Die Konsole ist eine Weiterentwicklung der normalen Windows-Konsole und versteht daher alle Befehle, welche die von Microsoft gelieferte PowerShell-Konsole auch versteht.
- Die Konsole unterstützt im Gegensatz zur normalen Windows-Konsole Kopieren und Einfügen mit [STRG]+[C] und [STRG]+[V].
- Integration zwischen Editor und Konsole: Die Konsole und der Editor werden in zwei getrennten Fenstern angezeigt, beim Start eines Skripts wird das Skript jedoch in der Konsole angezeigt. Der schnelle Wechsel ist mit [STRG]+[W] möglich.
- IntelliSense sowohl in der Konsole als auch im Editor für Commandlet-Namen, Commandlet-Parameter, Variablennamen, Pfadangaben, .NET-Klassenname .NET-Klassenmitglieder
- Codeeditor mit Syntaxhervorhebung
- Debugging mit Einzelschrittmodus
- Verwendung und Verwalten von wiederverwendbaren Programmcodeausschnitten (Code-Snippets)
- Mitschneiden von Eingaben in der Konsole, die über Tastenkürzel wieder aufgerufen werden können
- Anzeige aller aktuellen Variablen und Details zu ihren Inhalten
- Transparente Darstellung des Konsolenfensters möglich
- Direkter Zugriff auf PowerShell-Profilskripte

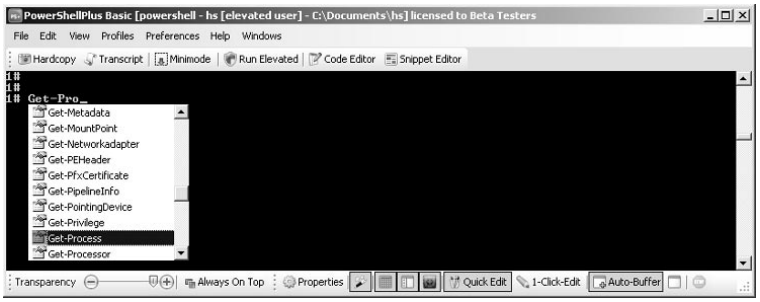


Abbildung 11.13
IntelliSense für Commandlet-Namen

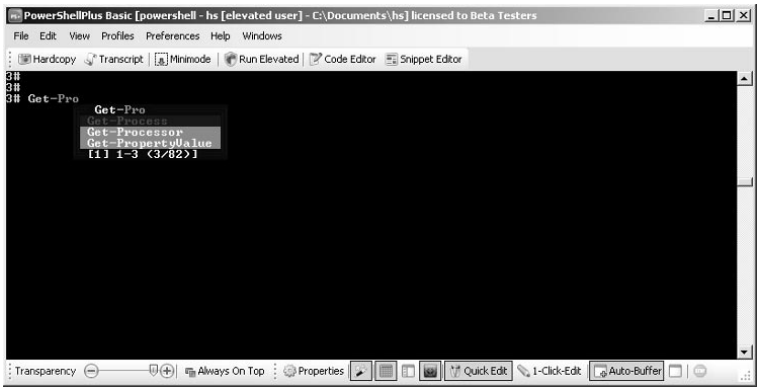


Abbildung 11.14
Eine Variante der IntelliSense für Commandlet-Namen

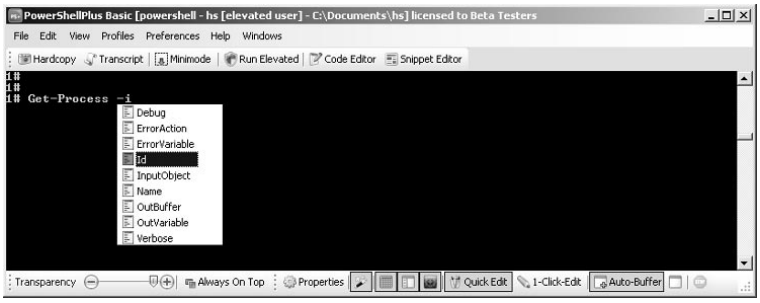


Abbildung 11.15
IntelliSense für Commandlet-Parameter

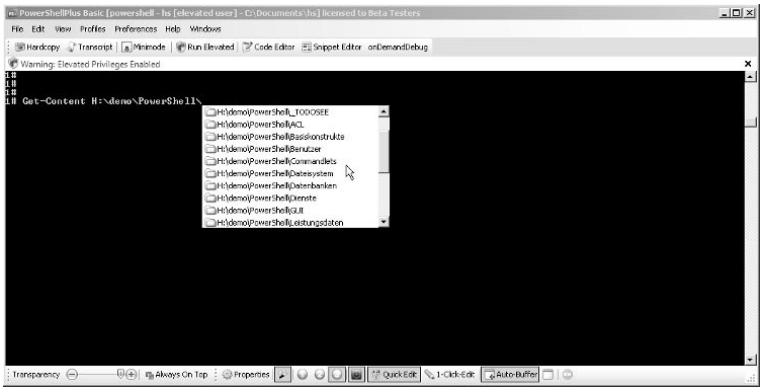


Abbildung 11.16
IntelliSense für Pfadangaben

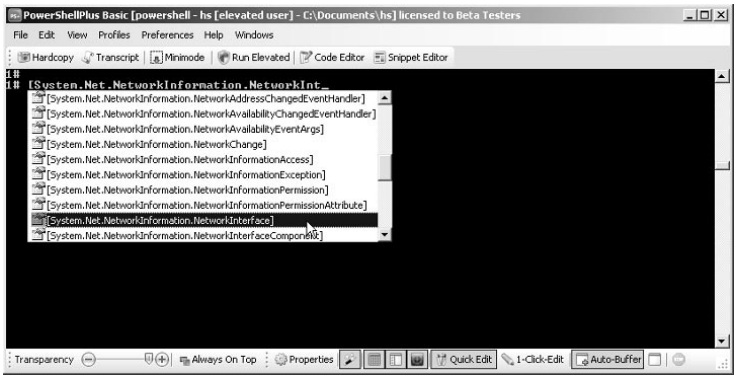


Abbildung 11.17 *IntelliSense für .NET-Klassennamen*

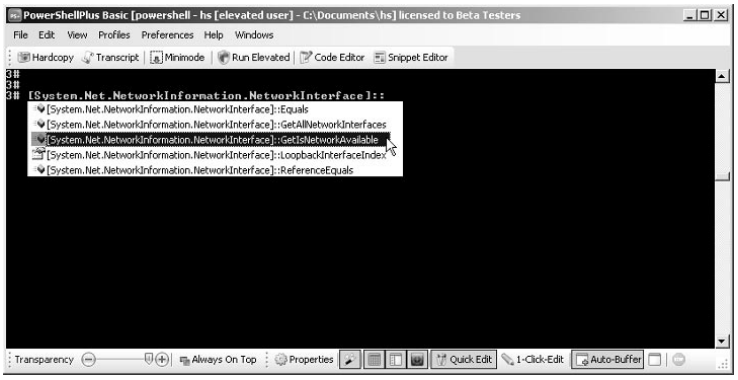


Abbildung 11.18 *IntelliSense für .NET-Klassenmitglieder*

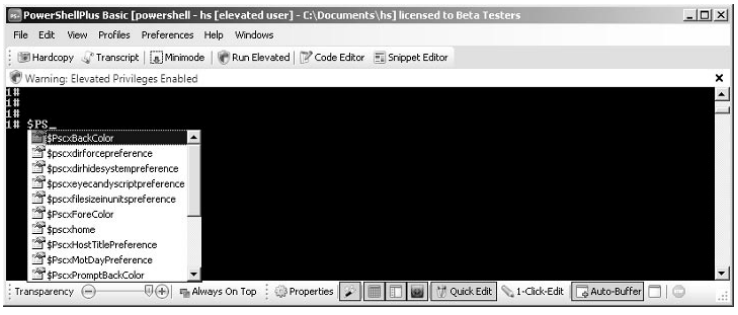


Abbildung 11.19 *IntelliSense für Variablennamen*

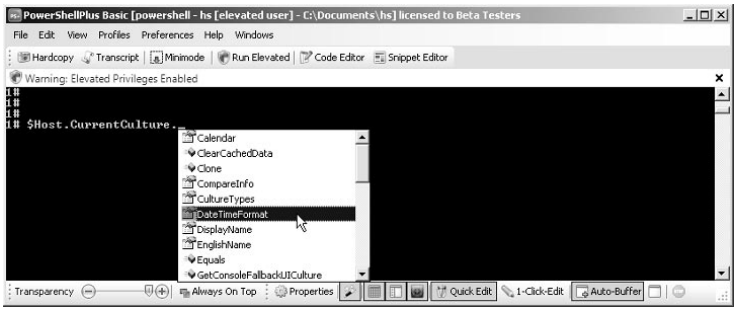


Abbildung 11.20 *IntelliSense für Mitglieder der Variablen*

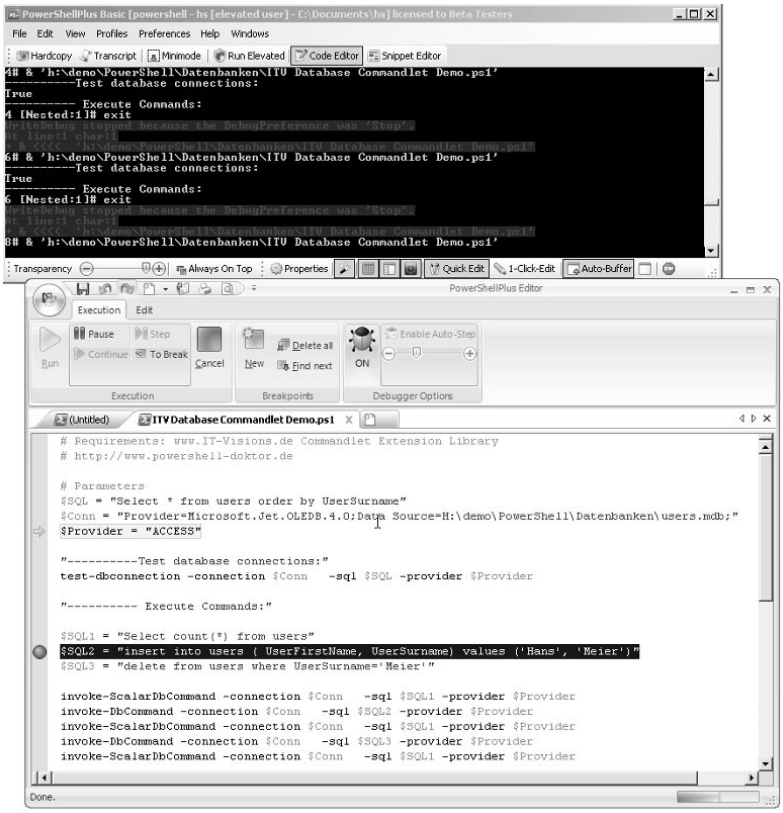


Abbildung 11.21
Debugging mit Einzelschrittmodus



Das Debugging hat im PowerShellPlus-Editor nicht nur eine Funktion zur Fehlersuche, sondern auch für die IntelliSense. Da ein Commandlet nicht deklariert, welche Objekte es in die Pipeline gibt, und die Ausgabe eines Commandlets kontextabhängig sein kann, kann der Editor nicht wissen, welche Möglichkeiten zur Verfügung stehen, solange der Nutzer das Skript nicht bis zu der besagten Stelle hat laufen lassen.

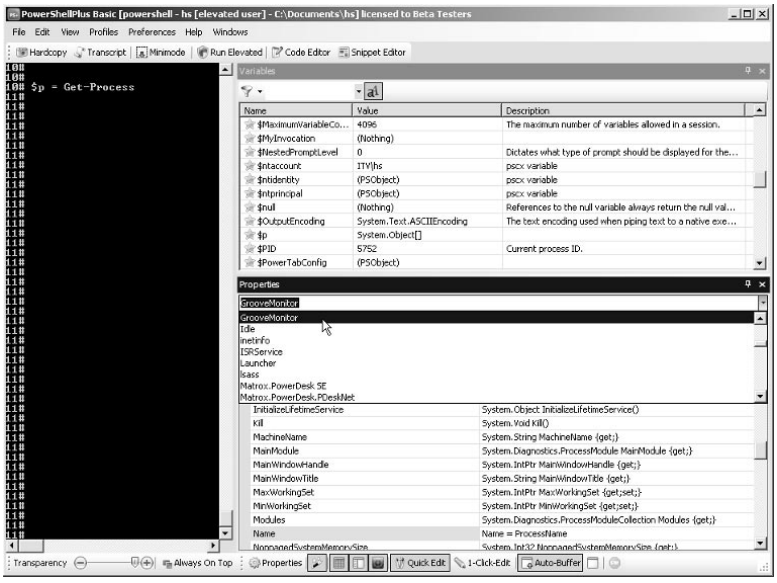


Abbildung 11.22 *Anzeige aller aktuellen Variablen und Details zu ihren Inhalten*

11.4 PoshConsole

PoshConsole ist eine grafische Konsole für die PowerShell, die mit Hilfe der neuen .NET-GUI-Technik „Windows Presentation Foundation" geschrieben ist. In der PoshConsole kann man direkt Grafiken ausgeben (siehe *Abbildung 11.23*).

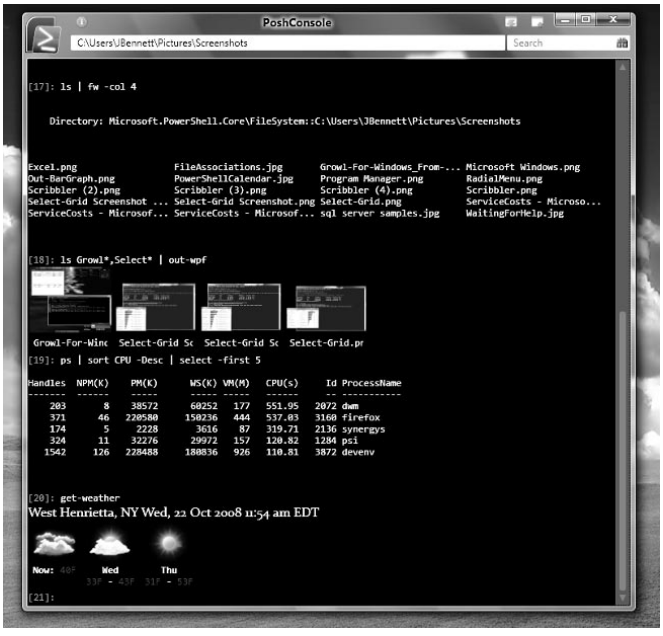


Abbildung 11.23 *Grafische Möglichkeiten der PoshConsole (Quelle: http://poshconsole.codeplex.com/)*

PoshConsole	
Hersteller:	Joel Bennett
Preis:	Kostenfrei
URL:	*http://poshconsole.codeplex.com/*
Enthalten auf Buch-CD?	Nein

11.5 PowerGUI

PowerGUI ist neben PowerShellPlus und PrimalScript der dritte bedeutende Editor für PowerShell.

PowerGUI	
Hersteller:	Quest Software
Preis:	Kostenfrei
URL:	*http://www.powergui.org*
Enthalten auf Buch-CD?	Nein

11.6 PowerShell Analyzer

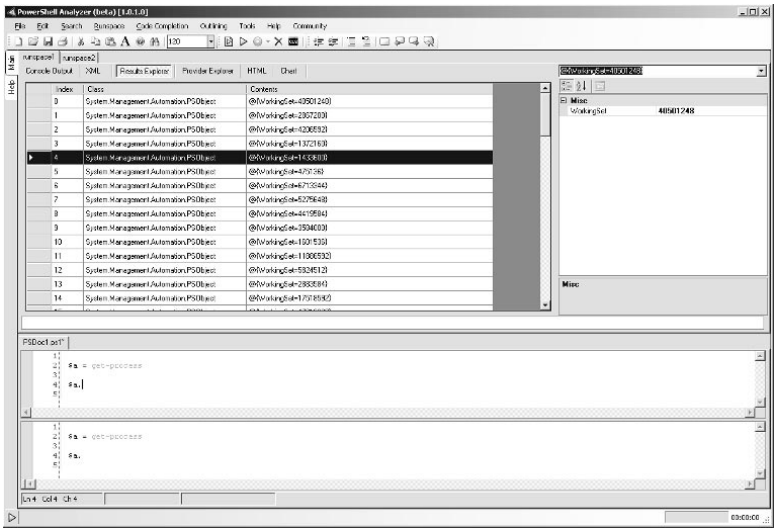
Der PowerShell Analyzer von Karl Prosser bietet gegenüber PowerShellPlus noch zwei zusätzliche Funktionen:

- Mehrere getrennte Ablaufumgebungen (sogenannte Runspaces)
- Visualisierung der Objekte in der Pipeline in einer Tabelle (siehe *Abbildung 11.24*) oder einem Diagramm

Allerdings fehlen auch zwei wichtige Funktionen: die IntelliSense für Klassen und Klassenmitglieder (siehe *Abbildung 11.24*) sowie ein Debugger.

PowerShell Analyzer	
Hersteller:	Shell Tools, LLC
Preis:	Kostenfrei
URL:	*http://www.powershellanalyzer.com*
Enthalten auf Buch-CD?	Nein

Abbildung 11.24
PowerShell Analyzer





Sowohl bei PowerShell Analyzer und PowerShellPlus als auch bei PowerGUI ist zu beachten, dass die Werkzeuge ein eigenes Hosting der PowerShell implementieren. Dies bedeutet, dass die Entwicklungsumgebungen zwar den gleichen Funktionsumfang wie die PowerShell-Konsole haben, jedoch mit dieser keinen gemeinsamen Deklarationsbereich besitzen. Definitionen von Aliases, Laufwerken und neuen skriptbasierten Commandlets betreffen also nur die jeweils aktuelle Ablaufumgebung.

11.7 PrimalScript

Der Universaleditor PrimalScript enthält ab Version 4.1 Unterstützung für die Erfassung von PowerShell-Skripten.

PrimalScript	
Hersteller:	Sapien
Preis:	299 Dollar
URL:	*http://www.primalscript.com/*
Enthalten auf Buch-CD?	Ja, Demo-Version von PrimalScript 2009

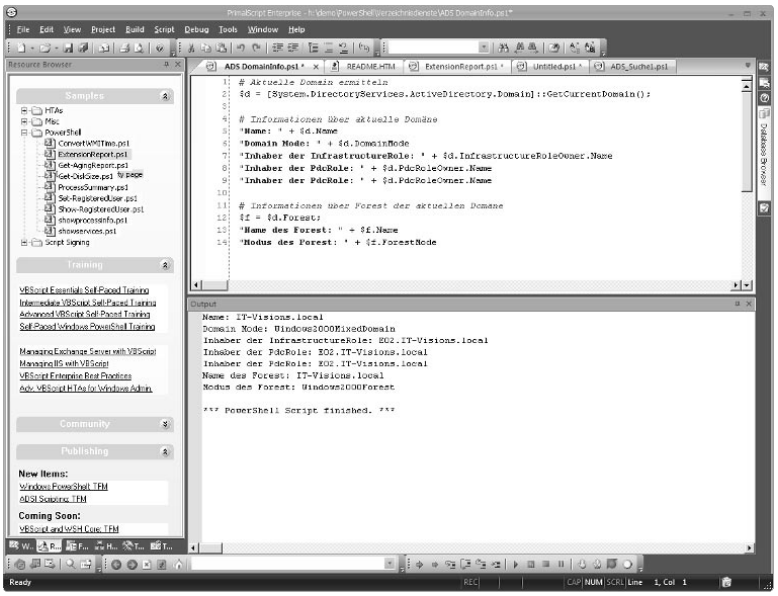


Abbildung 11.25
Ausgabe eines PowerShell-Skripts in PrimalScript

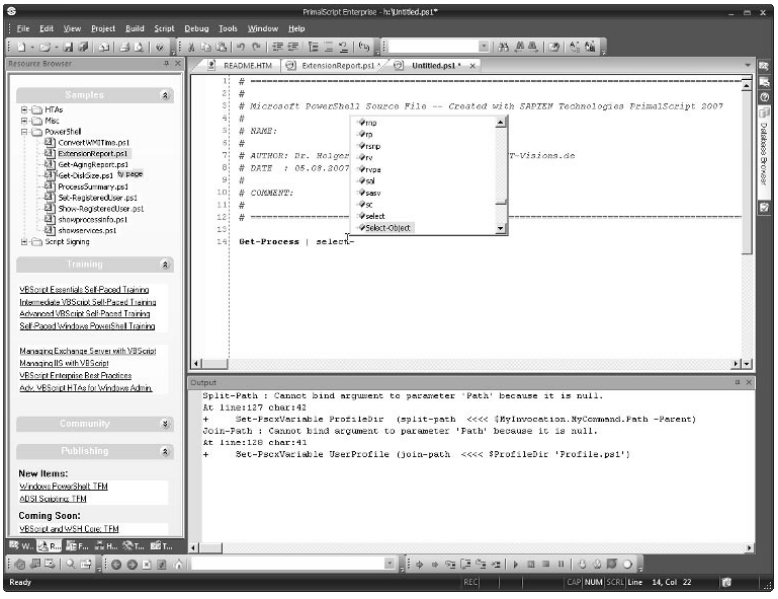


Abbildung 11.26
IntelliSense für Commandlets

Abbildung 11.27
IntelliSense für Parameter

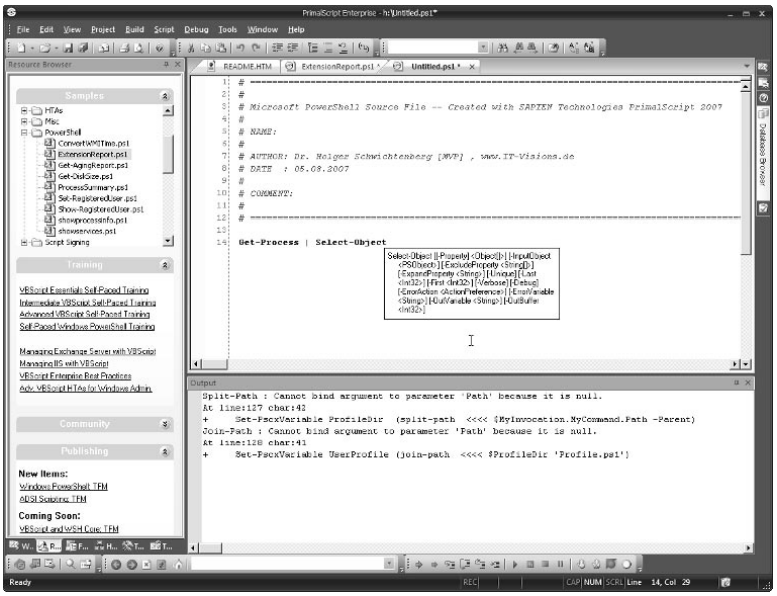
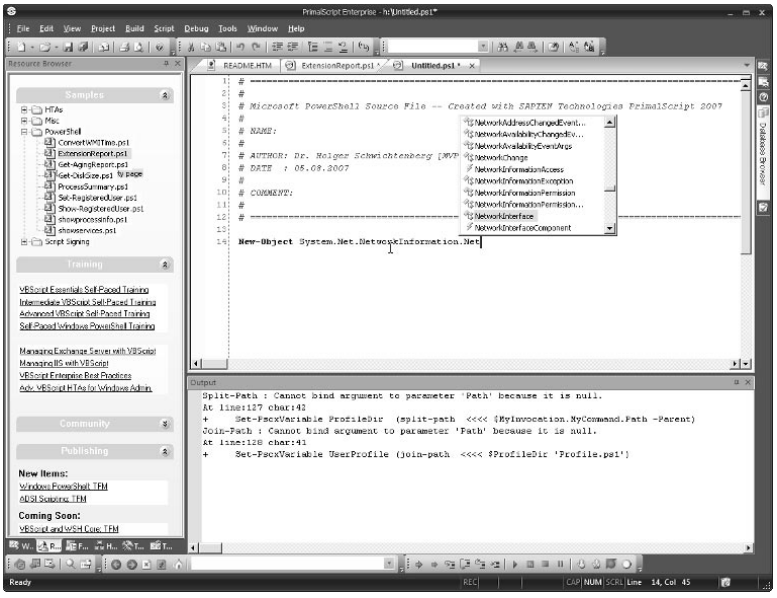


Abbildung 11.28
IntelliSense für Klassennamen



11.8 PowerShell Help

PowerShell Help ist ein einfaches Werkzeug zur Anzeige von Hilfetexten zu Commandlets.

PowerShell Help	
Hersteller:	Sapien
Preis:	Kostenlos
URL:	*http://www.primaltools.com/downloads/communitytools/*
Enthalten auf Buch-CD?	Ja

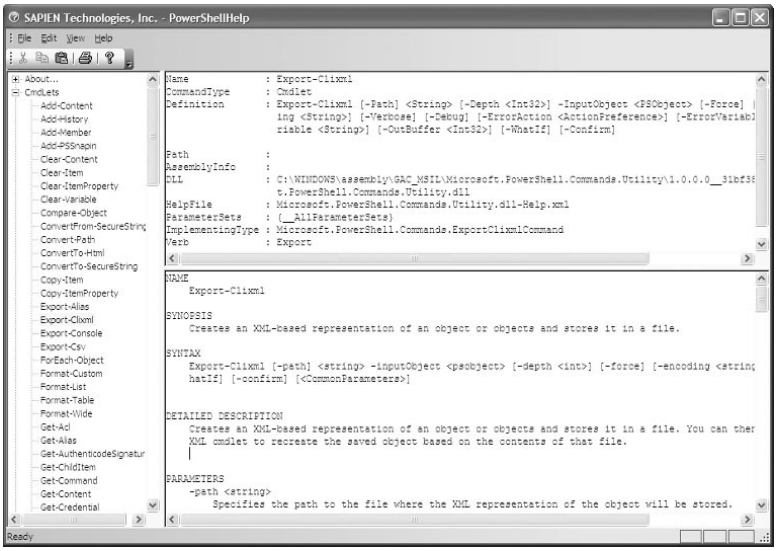


Abbildung 11.29 *PowerShell Help für PowerShell 1.0*

11.9 PowerShell Help Reader

PowerShell Help ist ein weiteres Werkzeug zur Anzeige von Hilfetexten zu Commandlets.

PowerShell Help Reader	
Hersteller:	PowerShellTools
Preis:	29,47 $
URL:	*http://powershelltools.com*
Enthalten auf Buch-CD?	Ja, Demo-Version

11.10 PowerTab

PowerTab erweitert die Fähigkeiten der PowerShell-Konsole, dem Benutzer Vorschläge für mögliche Befehle durch Drücken der Tabulator-Taste zu machen. Insbesondere liefert PowerTab Vorschläge für Mitglieder von .NET-Klassen.

PowerTab	
Hersteller:	Marc van Orsouw (der sich als „MoW" abkürzt)
Preis:	Kostenlos
URL:	*http://thepowershellguy.com/blogs/posh/pages/powertab.aspx*
Enthalten auf Buch-CD?	Ja

11.11 VS Command Shell

VS Command Shell erweitert Visual Studio um ein Befehlsfenster, in dem man sowohl Windows-Konsolenbefehle (Befehle für `cmd.exe`) als auch PowerShell-Befehle ausführen kann.



Das Add-In läuft nur in den kommerziellen Versionen von Visual Studio, nicht in den Express-Varianten, da diese keine Erweiterungen unterstützen.

VS Command Shell	
Hersteller:	Microsoft/Open-Source-Community-Projekt
Preis:	Kostenlos
URL:	*http://www.codeplex.com/VSCmdShell*
Enthalten auf Buch-CD?	Ja

11.12 PowerShell Remoting

PowerShell Remoting erlaubt die Verbindung mit einer entfernten Instanz der PowerShell 1.0 (d.h. einer Instanz auf einem anderen Computer), um dort Skripte auszuführen. Damit wird eine Begrenzung der PowerShell 1.0 überwunden, die Commandlets und Skripte nur lokal ausführen kann. Fernausführung ist in der PowerShell 1.0 sonst nur über WMI möglich.

Hinweis: In PowerShell 2.0 ist diese Erweiterung nicht mehr notwendig, da Fernausführung zum Standardumfang der PowerShell 2.0 gehört.

VS Command Shell	
Hersteller:	Microsoft/Open-Source-Community-Projekt
Preis:	Kostenlos
URL:	*http://www.codeplex.com/powershellremoting*
Enthalten auf Buch-CD?	Ja

11.13 Vergleich der Skripteditoren

Die folgende Tabelle, die die vier wichtigsten Skripteditoren für PowerShell vergleicht, zeigt, dass PrimalScript inzwischen im Skripteditor einen ähnlichen Funktionsumfang wie PowerShellPlus anbietet. Der zentrale Unterschied ist, dass PowerShellPlus zusätzlich eine interaktive Konsole mit IntelliSense-Eingabeunterstützung bietet, während PrimalScript nur ein Skripteditor ist.

	ISE 1.0	**PowerShell Plus 3.0**	**PrimalScript 2009**	**PowerGUI 2.0**
Website	**Keine**	**www.idera.com/Products/PowerShell/PowerShellPlus**	**www.primalscript.com/**	**www.powergui.org**
Preis	**Kostenlos**	**145 US-Dollar**	**299 US-Dollar**	**Kostenlos**
Konsole für interaktive Eingabe	Ja	Ja	Nein	Nein
Syntaxfarbhervorhebung	Ja	Ja	Ja	Ja
Skripteditor	Ja	Ja	Ja	Ja
IntelliSense für Commandlets	Nur Tabulatorvervollständigung	Ja	Ja	Ja
IntelliSense für Alias	Nur Tabulatorvervollständigung	Ja	Ja	Ja
IntelliSense für Parameter	Nur Tabulatorvervollständigung	Ja	Ja	Ja.

***Tabelle 11.1**: Vergleich der wichtigsten PowerShell-Editoren*

	ISE 1.0	PowerShell Plus 3.0	PrimalScript 2009	PowerGUI 2.0
Website	**Keine**	**www.idera.com/Products/PowerShell/PowerShellPlus**	**www.primalscript.com/**	**www.powergui.org**
Preis	**Kostenlos**	**145 US-Dollar**	**299 US-Dollar**	**Kostenlos**
IntelliSense für Klassennamen (nach New-Object)	Nein	Ja	Ja	Ja
IntelliSense für Klassennamen (in eckigen Klammern)	Nein	Ja	Ja	Ja
IntelliSense für .NET-Klassenmitglieder	Nein	Ja	Ja (außer Mitglieder statischer Klassen)	Nein
IntelliSense für Variablennamen	Nein	Ja	Ja	Ja
IntelliSense für Mitglieder der Variablen	Nein	Ja, erfordert aber tw. vorherige Ausführung des Skripts	Ja	Nein
IntelliSense für Pfadangaben	Nur Tabulatorvervollständigung	Ja	Nein	Ja
Variablenbrowser	Nein	Ja	Nein	Ja
Debugging	Ja	Ja	Ja	Ja
Skripte signieren	Nein	Ja	Nein	Nein
Bearbeitung anderer Codearten	Nein	C#, XML, VBScript, Batch, VB.NET, HTML	WSH, ActionScript, AWK, AutoIt, Batch, HTA, Kixtart, LotusScript, Perl, Python, Rebol, REXX, Ruby, SQL, Tcl, WinBatch, ASP, HTML, JSP, PHP, XML, XLST, XSD, C#, C++, VB, ColdFusion u.a.	Nein

***Tabelle 11.1**: Vergleich der wichtigsten PowerShell-Editoren (Forts.)*



Teil 2

PowerShell-Aufbauwissen

Inhalt dieses Teils sind die erweiterten Möglichkeiten der PowerShell, insbesondere der Zugriff auf Klassenbibliotheken (.NET, COM und WMI) sowie die Einbindung von Commandlet-Erweiterungen (PowerShell Snap-Ins).

12 Verwendung von .NET-Klassen

Mit der Windows PowerShell kann man jede .NET-Klasse aus der .NET Framework-Klassenbibliothek „FCL“ oder aus anderen .NET-Bibliotheken verwenden.

New-Object

12.1 Erzeugen von Instanzen

Mit dem Commandlet `New-Object` kann der Administrator jede beliebige .NET-Klasse (oder eine COM-Klasse, siehe nächstes Kapitel) instanzieren und sich von den Methoden dieser Klassen Objekte liefern lassen.



Ein fundamentaler Unterschied zwischen `Get-WmiObject` und `New-Object` ist, dass `Get-WmiObject` alle bestehenden Instanzen einer WMI-Klasse holt (z.B. alle Prozesse), während `New-Object` eine neue Instanz erzeugt. Die Semantik von `Get-WmiObject` ist für COM- und .NET-Objekte nicht möglich, da es dort jeweils kein zentrales Verzeichnis für Instanzen gibt. In WMI gibt es dafür das WMI-Repository. Wie man in COM- und .NET-Klassen eine Liste aller Instanzen erhält, ist vom Aufbau der jeweiligen Klassen abhängig und kann daher in der PowerShell nicht allgemein ausgedrückt werden.

```
PS C:\Skripte\ADSSkripte> New-Object -com scripting.filesystemobject | get-member

   TypeName: System.__ComObject#{2a0b9d10-4b87-11d3-a97a-00104b365c9f}

Name                 MemberType Definition
----                 ---------- ----------
BuildPath            Method     string BuildPath (string, string)
CopyFile             Method     void CopyFile (string, string, bool)
CopyFolder           Method     void CopyFolder (string, string, bool)
CreateFolder         Method     IFolder CreateFolder (string)
CreateTextFile       Method     ITextStream CreateTextFile (string, bool, bool)
DeleteFile           Method     void DeleteFile (string, bool)
DeleteFolder         Method     void DeleteFolder (string, bool)
DriveExists          Method     bool DriveExists (string)
FileExists           Method     bool FileExists (string)
FolderExists         Method     bool FolderExists (string)
GetAbsolutePathName  Method     string GetAbsolutePathName (string)
GetBaseName          Method     string GetBaseName (string)
GetDrive             Method     IDrive GetDrive (string)
GetDriveName         Method     string GetDriveName (string)
GetExtensionName     Method     string GetExtensionName (string)
GetFile              Method     IFile GetFile (string)
GetFileName          Method     string GetFileName (string)
GetFileVersion       Method     string GetFileVersion (string)
GetFolder            Method     IFolder GetFolder (string)
GetParentFolderName  Method     string GetParentFolderName (string)
GetSpecialFolder     Method     IFolder GetSpecialFolder (SpecialFolderConst)
GetStandardStream    Method     ITextStream GetStandardStream (StandardStreamTypes, bool)
GetTempName          Method     string GetTempName ()
MoveFile             Method     void MoveFile (string, string)
MoveFolder           Method     void MoveFolder (string, string)
OpenTextFile         Method     ITextStream OpenTextFile (string, IOMode, bool, Tristate)
Drives               Property   IDriveCollection Drives () {get}

PS C:\Skripte\ADSSkripte> _
```

Abbildung 12.1
Instanziierung eines COM-Objekts in der PowerShell



Die PowerShell bietet eine besondere Behandlung für WMI (`System.Management`), ADSI (`System.DirectoryServices`) und ADO.NET (`System.Data`). Objekte aus diesen Bibliotheken werden über Objektadapter dem Nutzer vereinfacht dargestellt. Collaboration Data Objects (CDO) zum Zugriff auf Microsoft Exchange werden aber in PowerShell 1.0 noch nicht besonders unterstützt.

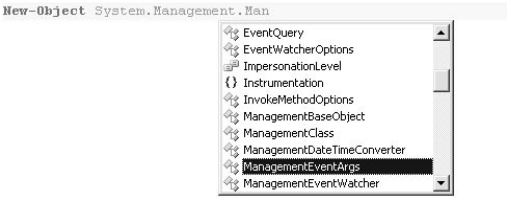


Abbildung 12.2 *PowerShellPlus bietet Eingabeunterstützung für .NET-Klassennamen nach New-Object.*

12.2 Parameterbehaftete Konstruktoren

Konstruktoren

Ein Konstruktor ist der Programmcode, der in der Klasse beim Instanziieren des Codes aufgerufen wird. .NET-Klassen können in den Konstruktoren Parameter erwarten. Diese kann man mit oder ohne runde Klammern nach dem Klassennamen angeben:

```
$o = New-Object System.Directoryservices.DirectoryEntry("LDAP://
SERVERF112")
```

oder

```
$o = New-Object System.Directoryservices.DirectoryEntry "LDAP://
SERVERF112"
```

12.3 Nutzung von Attributen und Methoden

Attribute und Methoden eines mit `New-Object` erzeugten .NET-Objekts nutzt man genauso wie Attribute und Methoden eines .NET-Objekts, das von einem Commandlet über die Punkt-Notation erzeugt wurde (Trennung von Objekt und Mitglied durch einen „.“).

Beispiel: WebClient

Das Beispiel zeigt das Herunterladen einer HTML-Seite von einer HTTP-Adresse mit Hilfe der Methode `DownloadString()` und des Attributs `ResponseHeaders` in der Klasse `System.Net.WebClient`.

Listing 12.1 *Herunterladen einer HTML-Seite von einer HTTP-Adresse*

```
$wc = (new-object System.Net.WebClient)
$wc.DownloadString("http://www.powershell-doktor.de")
"Zusätzliche Header-Informationen:"
$wc.ResponseHeaders.ToString()
```

Ob der Konstruktor einen Parameter erwartet oder nicht, macht nur bei `New-Object` einen Unterschied. Die weitere Verwendung über die Punkt-Notation ist gleich.

Beispiel: DirectoryEntry

```
$o = New-Object System.Directoryservices.DirectoryEntry("LDAP://
SERVERF112")
"Beschreibungstext: " + $o.description
"Ganzer Name: " + $o.distinguishedName
```

Listing 12.2 *Informationen von einem Active Directory-Domänencontroller*

Das obige Skript liefert zum Beispiel folgende Ausgabe:

```
Beschreibungstext: www.IT-Visions.de Windows Network
Ganzer Name: DC=IT-Visions,DC=local
```

12.4 Statische Mitglieder in .NET-Objekten/statische .NET-Klassen

Statische Mitglieder und Klassen

.NET-Klassen besitzen das Konzept der statischen Mitglieder (Klassenmitglieder), die man aufrufen kann, ohne eine Instanz zu erzeugen. Einige dieser Klassen sind auch statische Klassen, d.h., sie haben nur statische Mitglieder. Solche Klassen besitzen keinen Konstruktor. Folglich kann man das Commandlet `New-Object` auf statische Klassen nicht anwenden.

```
# Das geht nicht:
#(New-Object System.Console).Beep(100,50)
```

Für diesen Fall gibt es in der PowerShell ein anderes Konstrukt, bei dem man den .NET-Klassennamen in eckige Klammern setzt und dann den Namen des Mitglieds mit zwei Doppelpunkten abtrennt. Der folgende Befehl nutzt die statische Methode `Beep()` in der statischen Klasse `System.Console` zur Ausgabe eines Tons:

```
[System.Console]::Beep(100, 50)
```

Auch auf statische Mitglieder einer Klasse kann man über Variablen zugreifen. Dabei wird der Variablen nicht mit `New-Object` eine Instanz der Klasse zugewiesen, sondern die Klasse selbst über die Schreibweise in eckigen Klammern.

```
PS C:\Users\HS> $env = [System.Environment]
PS C:\Users\HS> $env
IsPublic IsSerial Name           BaseType
-------- -------- ----           --------
True     False    Environment    System.Object

PS C:\Users\HS> $env::MachineName
F174
```

12.5 Laden von Assemblies

Assembly:: LoadWithPartialName()

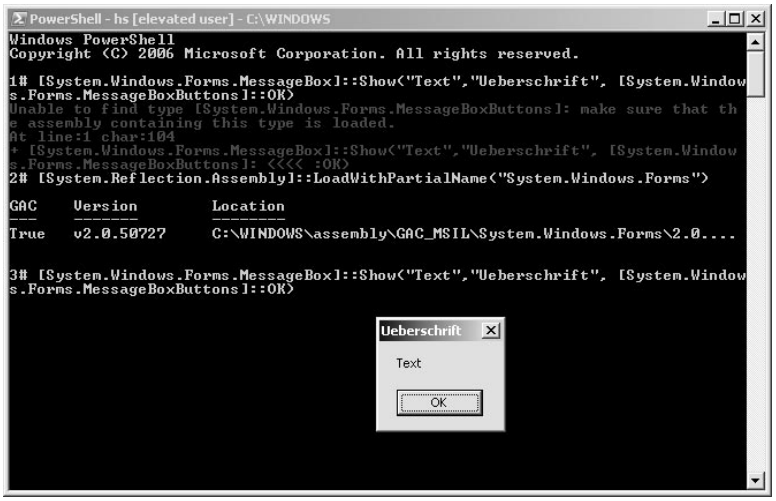
.NET-Klassen können über `New-Object` und die Notation in eckigen Klammern nur genutzt werden, wenn die Softwarekomponente (Assembly), in der sie sich befinden, auch geladen ist. Die PowerShell lädt einige Assemblies automatisch. In anderen Fällen muss man das Laden der Assembly über die Klasse `System.Reflection.Assembly` erst anstoßen. Um ein Dialogfenster auszugeben, muss man daher erst die *System.Windows.Forms.dll* laden. Da sich diese Assembly im sogenannten Global Assembly Cache (GAC) von .NET befindet, muss man keinen Pfad dahin angeben.

```
[System.Reflection.Assembly]::LoadWithPartialName("System.Windows.
Forms")
[System.Windows.Forms.MessageBox]::Show("Text","Ueberschrift",
[System.Windows.Forms.MessageBoxButtons]::OK)
```



Das Laden einer Assembly gilt bis zum Beenden der Instanz der PowerShell-Konsole. Der Befehl ist daher ein guter Kandidat für die *profile.ps1*-Datei.

Abbildung 12.3 *Die Nutzung der Klasse MessageBox ist erst erfolgreich, wenn die zugehörige Assembly geladen wurde.*





Anstelle der Notation mit eckigen Klammern kann man auch den eingebauten PowerShell-Typ `[Type]` verwenden, der auf Basis einer Zeichenkette ein .NET-Typobjekt erzeugt. Damit kann man das obige Beispiel auch so schreiben:

```
([Type]
"System.Reflection.Assembly")::LoadWithPartialName("System.
windows.forms")
[System.Windows.Forms.MessageBox]::Show("Text","Ueberschrift",
[System.Windows.Forms.MessageBoxButtons]::OK)
```

Eine weitere Vereinfachung ist möglich über das Commandlet `Resolve-Assembly` aus den PSCX, das eine Assembly findet und optional lädt, wenn `-import` angegeben ist:

```
resolve-assembly system.windows.forms -import
[System.Windows.Forms.MessageBox]::Show("Text","Ueberschrift",
[System.Windows.Forms.MessageBoxButtons]::OK)
```

12.6 Objektanalyse

Get-Member

Mit Hilfe des Commandlets `Get-Member`, das im ersten Teil des Buchs schon zur Analyse von Pipeline-Inhalten verwendet wurde, kann man übrigens auch den Inhalt einer Variablen analysieren, die eine Objektinstanz erhält. Zu beachten ist dabei nur, dass das Objekt entweder über die Pipeline an `Get-Member` zu schicken ist (also `$Variable | Get-Member`) oder aber der Parametername `-InputObject` zu verwenden ist (`Get-Member -InputObject $Variable`). Nicht nur `Get-Member`, sondern den meisten Commandlets ist es egal, ob sich in der Pipeline eine Objektmenge oder ein einzelnes Objekt befindet.



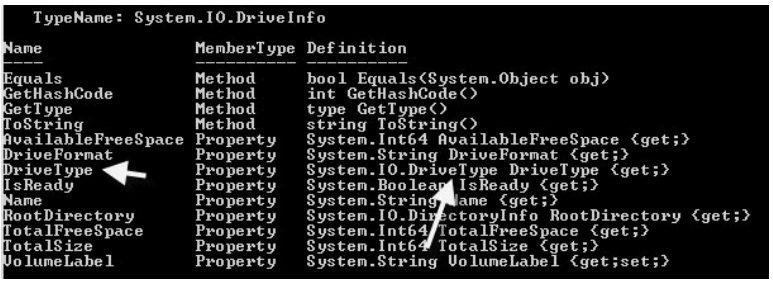
Die statischen Mitglieder einer Klasse kann man ganz allgemein erforschen mit `Get-Member`. Dazu sendet man den Klassennamen in eckigen Klammern an `Get-Member` mit dem Zusatz `-static`:

```
[System.Net.WebClient] | gm -static
```

12.7 Auflistungen (Enumerationen)

Eine Auflistung (Enumeration) ist eine .NET-Klasse, die eine Menge statischer Werte enthält, die andere Klassen nutzen. Zum Beispiel besitzt die Klasse `System.IO.DriveInfo` ein Attribut mit Namen `DriveType`, das den Datentyp `System.IO.DriveType` besitzt.

Abbildung 12.4
Mitglieder der Klasse System.IO.DriveInfo



System.IO.DriveInfo ist eine Aufzählung (Enumerationsklasse), die folgende Werte als möglich definiert:

- Unknown
- NoRootDirectory
- Removable
- Fixed
- Network
- CDRom
- Ram

Intern sind diesen Werte Zahlen zugeordnet, aber im Programmcode und in der PowerShell kann man diese Werte als sprechenden Namen verwenden.

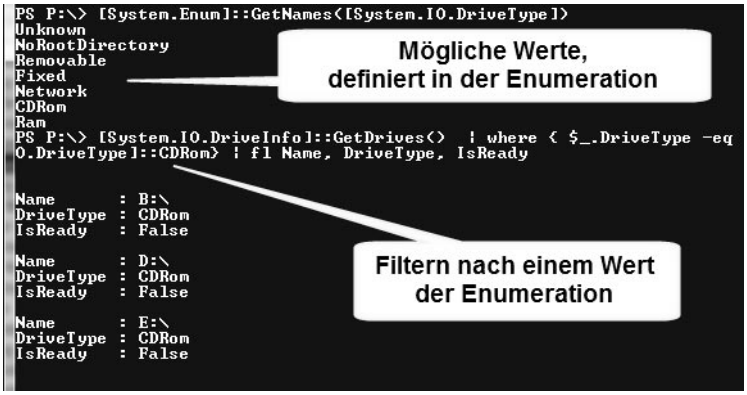
Der folgende Befehl filtert also alle CD-ROM-Laufwerke:

```
System.IO.DriveInfo]::GetDrives()  | where { $_.DriveType -eq
O.DriveType]::CDRom} | fl Name, DriveType, IsReady
```



Die Liste der möglichen Werte erhält man über die statische Methode `GetNames()` der Klasse `System.Enum`, der man den Namen einer Enumerationsklasse übergeben kann, z.B. `[System.Enum]::GetNames([System.IO.DriveType])`.

Abbildung 12.5
Arbeit mit Enumerationen



12.8 Verknüpfen von Aufzählungswerten

Enumerationen

Bei der Benutzung einiger .NET-Klassen, z.B. `FileSystemRights` für Rechte im Dateisystem, muss man verschiedene Flags durch ein binäres Oder miteinander verknüpfen. Wenn man bei jedem Flag den Namen der Aufzählung, in der das Flag definiert ist, wiederholen müsste, würden die Fingerkuppen strapaziert.

Die PowerShell ist in der Lage, aus einer Zeichenkette mit Kommatrennung die entsprechenden Flagwerte in der Enumeration zu suchen und miteinander durch ein binäres Oder zu verknüpfen. Anstelle von

```
$Rights= [System.Security.AccessControl.FileSystemRights]::Read `
-bor
[System.Security.AccessControl.FileSystemRights]::ReadExtendedAttri
butes `
-bor
[System.Security.AccessControl.FileSystemRights]::ReadAttributes `
-bor
[System.Security.AccessControl.FileSystemRights]::ReadPermissions
```

kann man also verkürzt schreiben:

```
$Rights = [System.Security.AccessControl.FileSystemRights]
"ReadData, ReadExtendedAttributes, ReadAttributes, ReadPermissions"
```

13 Verwendung von COM-Klassen

Die Windows PowerShell kann nicht nur .NET-Klassen, sondern auch Klassen aus dem Vorgänger von .NET, dem Component Object Model (COM), verwenden. COM ist auch in Windows 7 immer noch stärker verbreitet als .NET.

Die Handhabung von COM-Objekten unterscheidet sich in folgenden Punkten von der von .NET-Objekten:

- Es gibt keine Konstruktoren mit Parametern.
- Es gibt keine statischen Klassenmitglieder.
- Die Meta-Informationen, die Get-Member liefert, sind nicht so vollständig und so akkurat wie bei .NET.

13.1 Erzeugen von Instanzen

New-Object

Bei der Instanziierung von COM-Klassen kommt ebenfalls das Commandlet `New-Object` zum Einsatz. Dem Namen der COM-Klasse ist aber der Parameter `–comobject` (kurz: `-com`) voranzustellen. Als Name ist der Programmatic Identifier (ProgID) anzugeben. Die COM-Klasse muss auf dem lokalen System in der Registrierungsdatenbank verzeichnet sein. `New-Object` entspricht `CreateObject()` in Visual Basic/VBScript.

13.2 Nutzung von Attributen und Methoden

Punkt-Notation

Die Nutzung von Attributen und Methoden erfolgt wie bei .NET-Klassen über die Punkt-Notation. Fälle, in denen zwei Doppelpunkte („::“) zu verwenden wären, gibt es nicht.

Das folgende Beispiel zeigt den Aufruf der Methode `GetTempName()` aus der COM-Klasse `Scripting.FileSystemObject`. Diese Methode liefert einen Namen für eine temporäre Datei.

```
$fso = New-Object -com "scripting.filesystemobject"
$fso.GetTempName()
```

Listing 13.1 *[COM Create-Object.ps1]*

Mit dem zweiten Skript öffnet man den Internet Explorer mit einer bestimmten Seite:

Listing 13.2 *[COM Create-Object.ps1]*

```
$ie = New-Object -com "InternetExplorer.Application"
$ie.Navigate("http://www.powershell-doktor.de")
$ie.visible = $true
```

13.3 Holen bestehender Instanzen

GetObject()

In der COM-Welt gibt es darüber hinaus eine globale Liste, in der sich laufende Instanzen bzw. im Dateisystem existierende Objekte verzeichnen können. Diese Running Objects Table spricht man in vielen Programmiersprachen über `GetObject()` an. Ein direktes Äquivalent für das `GetObject()`gibt es in der PowerShell nicht. Hier gibt es nur die Möglichkeit, die Assembly für Visual Basic .NET zu laden und die dortige Methode `GetObject()` zu nutzen.

Das folgende Beispiel zeigt ein Word-Dokument (.doc) in Microsoft Word auf dem Bildschirm an und schreibt einen Text in das Dokument:

Listing 13.3 *[COM Get-Object.ps1]*

```
$doc = [microsoft.visualbasic.interaction]::GetObject("C:\temp\
ManuScript.doc")
$doc.application.visible = $true
$doc.application.selection.typetext("Erfolgreicher Start von
Word!")
```

14 Zugriff auf die Windows Management Instrumentation (WMI)

Das Commandlet `Get-WmiObject` sowie die eingebauten PowerShell-Typen `[WMI]`, `[WMICLASS]` und `[WMISEARCHER]` eröffnen die Welt der Windows Management Instrumentation (WMI), die in den modernen Windows-Betriebssystemen fast jeden Baustein des Betriebssystems objektorientiert anbietet.

Get-Wmiobject

14.1 Einführung in WMI

Dieses Unterkapitel bietet eine Einführung in WMI für Leser, die bisher nicht mit WMI vertraut sind. Wenn Sie WMI schon aus dem Windows Script Host (WSH) oder anderen Programmierumgebungen kennen, können Sie dieses Unterkapitel übergehen.

Was ist WMI?

Die *Windows Management Instrumentation (WMI)* ist ein Windows-Systembaustein zum Zugriff auf System- und Netzwerkinformationen. Mit WMI kann man:

- Systeminformationen über einzelne Systembausteine oder ganze Mengen von Systembausteinen auslesen,
- Systeminformationen verändern,
- Aktionen ausführen,
- sich über Veränderungen im System informieren lassen.

WMI ist die Microsoft-Implementierung des *Web Based Enterprise Management (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-Werkzeugen häufig noch die Bezeichnung WBEM.

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 Managementarchitekturen wie dem OSI Management Framework X.700 (Common Management Information Protocol – CMIP) und dem Simple Network Management Protocol (zu CMIP und SNMP siehe [STA93]).



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 Werkzeug noch Anwendung.

WMI ist fester Bestandteil von Windows schon seit Windows ME. In Windows Vista ist WMI 6.0, in Windows 7 WMI 6.1 enthalten.

WMI-Funktionsumfang

Funktionen

Die Informationsfülle, die WMI auf modernen Betriebssystemen (XP, 2003, Vista, 2008, Windows 7) liefert, ist riesig. Beispiele für Bereiche, aus denen WMI Informationen liefert, zeigt die folgende Tabelle.

Grund-konfiguration	• BIOS • Boot-Konfiguration • Installiertes Betriebssystem (z.B. Betriebssystemname, Build-Version, Installationsdatum, Datum und Uhrzeit des letzten Boot-Vorgangs) • Umgebungsvariablen • Performance-Monitor-Daten • SNMP-Daten • Eingerichtete Zeitzonen • Drucker und Druckerwarteschlangen • Auslagerungsdateien • Datum und Uhrzeit • Clustering

***Tabelle 14.1**: Überblick über WMI-Informationen*

Hard- und Software	• Installierte Software • Installierte Updates und Hotfixes • Installierte Hardware (z.B. Netzwerkkarten, Grafikkarten) einschließlich Treiber und deren Zuordnung zu LoadOrderGroups, belegter Ressourcen (IRQ, Port, DMA), Konfiguration (z.B. Druckereinstellungen) • Installierte COM-Komponenten einschließlich Zuordnung zu Komponentenkategorien und DCOM-Einstellungen • Laufende Prozesse • Geplante Vorgänge (Zeitplandienst) • Programmgruppen im Startmenü • Windows-Systemdienste
Sicherheit	• Benutzerkonten (inklusive deren Gruppenzuordnung, Desktop-Einstellungen und Ereignisprotokolleinträge) • Security Configuration Editor (SCE)
Dateisystem und Datenspeicher	• Ordner und Dateien des Dateisystems • Netzlaufwerksverbindungen • Dateisicherheit, Freigabesicherheit • Registrierungsdatenbank • Ereignisprotokoll • ODBC-Einstellungen • Disk Quotas • Ausführung von CHKDSK • Distributed File System (DFS)
Netzwerk	• IP-Routing • Ausführung eines Ping • Netzwerkverbindungen und Sitzungen • Terminal Services • Active Directory • DNS-Server • Network Load Balancing (NLB) • Microsoft Exchange Server • Internet Information Server (IIS) • ASP.NET • Windows Communication Foundation (WCF)

***Tabelle 14.1**: Überblick über WMI-Informationen (Forts.)*



Einige Schwierigkeiten mit WMI resultieren daraus, dass WMI mit jeder Windows-Version (stark) erweitert wurde, die erweiterten WMI-Klassen und -Funktionen aber nicht als Add-on für ältere Windows-Versionen bereitgestellt werden. Viele Skripte laufen daher nur auf den jeweils neuesten Windows-Versionen, auf älteren Plattformen erhält man nichtssagende Fehlermeldungen wie „Automatisierungsfehler".

WMI-Klassen und WMI-Objekte

WMI ist ein objektorientiertes Konzept, bei dem alle Informationen in Form von strukturierten Objekten bereitgestellt werden, die Instanzen von Klassen sind. Eine Klasse beschreibt eine Informationsart (z.B. Datei), ein Objekt enthält die Informationen eines konkreten Vorkommens dieser Informationsart (z.B. *c:\Daten\Computerliste.txt*).

WMI-Klassen besitzen Informationen in Attributen und erlauben die Ausführung von Aktionen durch Methoden. Klassen können von anderen Klassen erben.

WMI 6.1 in Windows 7 Ultimate enthält nach der Grundinstallation 9905 WMI-Klassen, die – aus Gründen der Übersichtlichkeit – in 58 sogenannte Namensräume gegliedert sind. Ältere Windows-Versionen besitzen deutlich weniger WMI-Klassen.

Durch die Installation von Zusatzdiensten (z.B. Internet Information Services) und von Zusatzprodukten (z.B. Microsoft Office) kommen Dutzende weiterer Klassen hinzu, da heute viele Produkte einen WMI-Provider mitliefern.



Hier auch nur einige dieser Klassen zu nennen, würde nur Platz verschwenden. Sie werden einige WMI-Klassen in Beispielen in den folgenden Kapiteln kennenlernen.

Arten von WMI-Klassen

WMI-Klassen beginnen meistens mit der Vorsilbe „Win32" oder „CIM". *CIM-Klassen* sind eine sehr allgemeine, betriebssystemunabhängige Beschreibung von Ressourcen. *Win32-Klassen* sind eine konkrete, in der Windows-Umgebung implementierte Repräsentation von Ressourcen. Die meisten Win32-Klassen sind von CIM-Klassen abgeleitet und erweitern den Standard. Einige Ressourcen in einem Windows-System können auch direkt durch CIM-Klassen abgebildet werden.

Es gibt drei Arten von Klassen in WMI:

- Abstrakte Klassen, von denen es keinen Instanzen geben kann und die nur der Vererbung dienen
- Statische Klassen: Instanzen dieser Klassen werden im WMI-Repository gespeichert.
- Dynamische Klassen: Instanzen dieser Klassen werden dynamisch von einem WMI-Provider geliefert.

Systemklassen

WMI definiert eine Reihe von Systemklassen, die der Verwaltung von WMI selbst und insbesondere dem Ereignissystem dienen. Die Systemklassen sind in jedem Provider implementiert; sie sind daran erkennbar, dass der Name mit einem doppelten Unterstrich beginnt. Beispiele für Systemklassen sind:

- `__EventConsumer`
- `__Namespace`
- `__Event`
- `__InstanceDeletionEvent`

Schlüsselattribute

Keys

Schlüsselattribute (*Keys*) sind besondere 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 sich auch ein Key in WMI aus mehreren Attributen zusammensetzen. 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.

Einige WMI-Klassen besitzen keine Schlüsselattribute (z.B. */root/default:__cimomidentification* und */root/cimv2:NetDiagnostics*). Diese Klassen sind sogenannte Singleton-Klassen. Eine Singleton-Klasse ist in der objektorientierten Programmierung eine Klasse, von der es nur eine Instanz geben kann. In WMI wird eine Singleton-Klasse mit dem Qualifier (vgl. nächster Abschnitt) `Singleton = True` gekennzeichnet. An Stelle eines Paars aus Schlüsselattributname und Schlüsselwert wird zur Identifizierung des einen Objekts das Zeichen `@` verwendet.

Systemattribute

Systemattribute

Alle WMI-Klassen und alle Instanzen dieser Klassen besitzen eine Reihe von Systemattributen. Die Namen dieser Systemattribute beginnen mit einem doppelten Unterstrich (__).

Tabelle 14.2 WMI-Systemeigenschaften

Attribut	Erläuterung
__Class	Name der Klasse, zu der das WMI-Objekt gehört
__Derivation	Eine Zeichenkettenliste, welche die Vererbungshierarchie wiedergibt. Der erste Eintrag ist die direkte Oberklasse.
__Dynasty	Name der obersten Klasse der Vererbungshierarchie. Bei der obersten Klasse steht hier keine leere Zeichenkette, sondern der gleiche String wie bei __Class.
__Genus	1 = Eintrag ist eine Klasse. 2 = Eintrag ist ein WMI-Objekt, d.h. die Instanz einer WMI-Klasse.
__Namespace	WMI-Namensraum, in dem die Klasse registriert ist
__Path	Vollständiger WMI-Pfad einschließlich Server und WMI-Namensraum
__Property_Count	Anzahl der Attribute der Klasse. Dabei werden diese Systemattribute nicht mitgezählt.
__Relpath	WMI-Pfad ohne Server und WMI-Namensraum
__Server	Name des Servers
__Superclass	Name der direkten Oberklasse

Datentypen

Datentypen

WMI definiert sechzehn Standarddatentypen (siehe *Tabelle 14.3*).

Tabelle 14.3 CIM-Standarddatentypen

Datentyp
wbemCimtypeSint8
wbemCimtypeUint8
wbemCimtypeSint16
wbemCimtypeUint16
wbemCimtypeSint32
wbemCimtypeUint32
wbemCimtypeSint64
wbemCimtypeUint64
wbemCimtypeReal32
wbemCimtypeReal64
wbemCimtypeBoolean

Datentyp
wbemCimtypeString
wbemCimtypeDatetime
wbemCimtypeReference
wbemCimtypeChar16
wbemCimtypeObject

Datum und Uhrzeit

Datum und Uhrzeit werden als Zeichenkette der Form `yyyy mmddHHMMSS.mmmmmmsUUU` gespeichert, wobei neben dem selbst erklärenden Kürzel anzumerken ist, dass `mmmmmm` die Anzahl der Millisekunden ist und `UUU` die Anzahl der Minuten, um welche die lokale Zeit von der Universal Coordinated Time (UTC) abweicht. Das `s` ist das Vorzeichen. In Deutschland steht daher für `UUU` der Wert *+060*.

WMI kennt auch ein eigenes Format für Zeitintervalle: `dddddddd HHMMSS.mmmmmm:000`. Auch ein Zeitintervall wird als Zeichenkette abgelegt. Dabei repräsentiert `dddddddd` die Anzahl der Tage. Die Zeichenkette endet immer auf `:000`.

Qualifizierer (Qualifier)

Qualifizierer (engl. Qualifier) sind Zusatzinformationen, die in WMI eine Klasse, ein Objekt, ein Attribut, eine Methode oder einen Parameter näher 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.

An einem Qualifier erkennt man auch die Art einer Klasse: Die Existenz der Qualifier `abstract` und `dynamic` weist auf die entsprechenden Typen hin. Ist keiner dieser Qualifier vorhanden, ist die Klasse statisch. Eine Assoziationsklasse hat einen `association`-Qualifier.

Weitere interessante Qualifier:

- `Provider` zeigt für eine Klasse den Namen des WMI-Providers an, der die Klasse bereitstellt.
- `EnumPrivileges` legt eine Liste von Privilegien fest, die gesetzt sein müssen, um diese Klasse zu nutzen.
- `Singleton` kennzeichnet eine WMI-Klasse, von der es nur eine Instanz geben kann.

Objektassoziationen

Assoziationen

WMI-Objekte können durch Assoziationen miteinander verbunden sein (z.B. ein Verzeichnis enthält eine Datei). Eine Objektassoziation ist selbst eine Instanz einer WMI-Klasse. Ein Beispiel für eine Assoziation ist `CIM_DirectoryContainsFile`. Diese Klasse stellt eine Assoziation zwischen `CIM_Directory` und `CIM_DataFile` dar.

WMI-Namensräume (Namespaces)

Namensräume

Ein WMI-Namensraum ist ein Instrument zur Gruppierung von WMI-Klassen und deren Instanzen in logische Einheiten. Ein Namensraum ist der Startpunkt sowohl für eine Vererbungshierarchie von WMI-Klassen als auch für eine Containment-Hierarchie von WMI-Objekten. Jeder Namensraum hat also sein eigenes Schema. Klassennamen in zwei Namensräumen dürfen theoretisch gleich sein, sollten es aber nicht, da in zukünftigen WMI-Versionen geplant ist, namensraumübergreifende Operationen zu ermöglichen. Es sind keine Objektassoziationen zwischen verschiedenen Namensräumen möglich. Um gleiche Klassennamen zu vermeiden, gibt es die Konvention, dass dem Klassennamen stets der Namensraumname vorangestellt werden soll. Ein Namensraum ist selbst eine Klasse, die direkt oder indirekt von der Systemklasse `__Namespace` geerbt hat.

Namensräume können hierarchisch angeordnet werden, so dass eine Namensraumhierarchie entsteht. Diese Hierarchie dient aber nur der Übersichtlichkeit; sie impliziert keine Vererbung von Klassen. Ein bestimmter Namensraum wird über einen hierarchischen Pfad der Form *Wurzelnamensraum\Namensraum1\Namensraum2\...* usw. angesprochen.

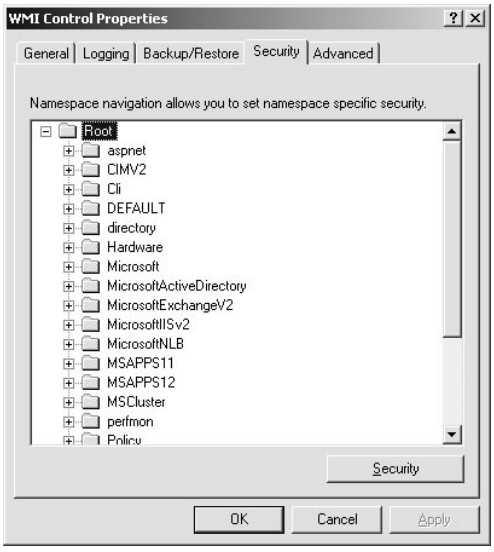


Abbildung 14.1 *WMI-Namensräume in Windows Vista Business (Grundinstallation)*



Namensraum versus Provider Namensräume sind nicht gleichzusetzen mit Providern. Ein Provider kann mehrere Namensräume realisieren, ebenso wie ein Namensraum Klassen aus mehreren Providern enthalten kann.

Lokalisierung

WMI erlaubt Lokalisierung (d.h. landesspezifische Anpassung) von Schemainformationen und die Speicherung mehrerer lokalisierter Versionen eines Namensraums innerhalb des WMI-Repository. WMI speichert dazu die sprachneutralen Teile der Klassendefinition getrennt von den landesspezifischen Teilen. Landesspezifische Informationen sind insbesondere die Hilfeinformationen zu den Klassen und Eigenschaften.

LocaleID

Die landesspezifischen Informationen werden in Unter-Namensräumen gespeichert. Jedes Land hat eine *LocaleID*.

- *ms_407* steht für Deutsch.
- *ms_409* steht für amerikanisches Englisch.

Der Namensraum *\root\CIMV2\ms_407* ist also der *\root\CIMV2*-Namensraum mit deutschen Hilfeinformationen, *root\CIMV2\ms_409* der gleiche mit englischen Texten.

Beim Zugriff auf den übergeordneten Namensraum *root\CIMV2* ist die Sprache abhängig von Ihren Computereinstellungen.

In einigen Werkzeugen werden diese Unter-Namensräume durch eine Weltkugel angezeigt, in anderen Werkzeugen gibt es keinen Unterschied zu den normalen Namensräumen.

Statische Methoden

WMI-Klassen können sogenannte 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.

WMI-Pfade

Sowohl einzelne WMI-Objekte als auch WMI-Klassen werden durch WMI-Pfade adressiert.

Ein WMI-Pfad ist folgendermaßen aufgebaut:

- Aufbau eines WMI-Pfads für eine WMI-Klasse

```
\\Computer\Namensraum:Klasse
```

- Aufbau eines WMI-Pfads für ein WMI-Objekt

```
\\Computer\Namensraum:Klasse.Schluessel='wert'
```

Dabei bedeuten die Elemente Folgendes:

- `Computer` steht für den Namen des anzusprechenden Computers. Ein Punkt in Anführungszeichen (`"."`) steht dabei für den lokalen Computer, auf dem das Skript läuft.
- `Namensraum` ist der Namensraum, in dem die anzusprechende Klasse registriert wurde. Diese Angabe ist optional, wenn der Standardnamensraum angesprochen werden soll.
- `Klasse` ist der Name der Klasse, die angesprochen werden soll.
- Mit `Schluessel='wert'` wird optional festgelegt, welche Instanz der Klasse angesprochen werden soll. Dabei ist `Schluessel` der Name des Schlüsselattributs der Klasse und `Wert` der Wert dieses Schlüsselattributs in der gesuchten Instanz.



Es gibt WMI-Klassen, die einen Schlüssel besitzen, der aus mehreren Attributen besteht. In diesem Fall sind die Schlüsselattribute durch ein Komma zu trennen, z.B.:

```
\\F111\root\cimv2:Win32_UserAccount.Domain="FBI",Name="hs"
```

Die folgende Tabelle zeigt Beispiele für den Zugriff auf WMI-Objekte am Beispiel der Klasse `Win32_LogicalDisk`, die ein Laufwerk repräsentiert. Die Klasse liegt im Namensraum `\root\cimv2`. Wie Sie der Tabelle entnehmen können, sind viele Bestandteile der Pfadangabe optional.

Tabelle 14.4 *Beispiele für WMI-Zugriffe*

WMI-Pfad	Beschreibung
`\\.\root\cimv2:` `Win32_LogicalDisk.DeviceID='D:'`	Die Instanz der Klasse `Win32_LogicalDisk` aus dem Namensraum `\root\cimv2`: mit dem Namen „D:“ auf dem lokalen Computer
`\\ServerSERVERF112\root\cimv2:` `Win32_LogicalDisk.DeviceID='D:'`	Die Instanz der Klasse `Win32_LogicalDisk` aus dem Namensraum `\root\cimv2`: mit dem Namen „D:“ auf dem Computer *Server_SERVERF112*
`Win32_LogicalDisk.DeviceID='D:'`	Die Instanz der Klasse `Win32_LogicalDisk` aus dem Standard-Namensraum mit dem Namen „D:“ auf dem lokalen Computer
`\\ServerSERVERF112\root\cimv2:` `Win32_LogicalDisk`	Alle Instanzen der Klasse `Win32_LogicalDisk` auf dem Computer *Server_SERVERF112*
`\\ServerSERVERF112\root\cimv2:` `Win32_UserAccount.Domain="FBI",Name="hs"`	Das Benutzerkonto "FBI\hs" auf dem Rechner *Server_SERVERF112*



Der sogenannte Standardnamensraum, wo die Klasse gesucht wird, wenn kein Namensraum explizit genannt wurde, ist in der Registrierungsdatenbank festgelegt (*HKEY_LOCAL_MACH INE \Software\ Microsoft\WBEM\Scripting\ Default Namespace*). Die Standardeinstellung ist `\root\cimv2` und kann über die MMC-Konsole „WMI-Steuerung" verändert werden. Man sollte diese Einstellung aber nicht ändern.

WMI-Schema

Schema

Das WMI-Schema (alias CIM-Schema) definiert die Klassen mit ihren Attributen und Methoden, die Vererbungshierarchie und die Objektassoziationen. Zur Laufzeit existieren von diesen Klassen Instanzen (WMI-Objekte). Managed Objects können durch Assoziationen miteinander verbunden sein, so dass eine Containment-Hierarchie entsteht. WMI-Klassen erlauben Vererbung (Einfachvererbung), so dass die Klassen auch in einer Vererbungshierarchie zueinander stehen können.

Genau genommen besitzt WMI mehrere Schemata, da jeder WMI-Namensraum ein eigenes Schema hat. Ein WMI-Schema ist oft sehr umfangreich und enthält auch Hilfeinformationen, wahlweise in verschiedenen Sprachen. Das Schema für Windows heißt „Win32 Schema Version 2" und ist eine Microsoft-Erweiterung des CIM-Schemas Version 2.



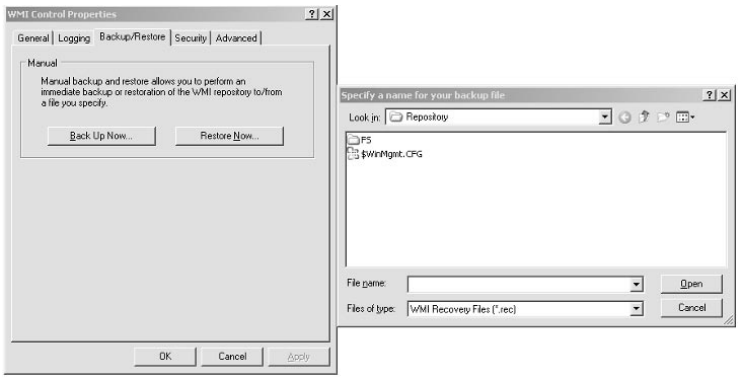
Auch der Aufbau des Schemas ist Teil des Schemas. Dieser Teil des Schemas wird als *Metamodell* bezeichnet.

WMI-Repository

Das WMI-Repository (alias CIM-Repository) ist der Datenspeicher/die Datenbank von WMI. Im Repository werden das Schema und auch Daten über die Instanzen gespeichert. WMI erzeugt einige Instanzen ad hoc, wenn diese angefragt werden, weil es keinen Sinn machen würde, diese zwischenzuspeichern (z.B. Daten über die laufenden Prozesse). Andere Daten werden im Repository abgelegt, weil diese sich selten oder nie ändern (z.B. Hardwaredaten).

Das WMI-Repository liegt in einem gleichnamigen Verzeichnis unterhalb von *%SystemRoot%\System32\WBEM\Repository*. Mit der MMC-Konsole „WMI-Steuerung" kann man diese Datenbank sichern und wiederherstellen.

Tabelle 14.5
Sichern und Wiederherstellen des CIM-Repositorys in WMI



WMI-Systemdienst

Das WMI-Repository wird durch die ausführbare Datei *WinMgmt.exe* betrieben. *WinMgmt.exe* läuft unter allen NT-basierten Windows-Systemen als Systemdienst unter dem Namen „WinMgmt (Windows-Verwaltungsinstrumentation)“. Auf Windows 9x/Windows ME wird *WinMgmt.exe* beim ersten WMI-Aufruf als normaler Prozess gestartet, wenn ein Aufruf erfolgt.



WMI kann Fernzugriffe auf andere Computer realisieren, wenn auf dem anderen Computer auch WMI installiert ist, der WMI-Dienst dort läuft und der Zugang zu dem anderen Computer über DCOM-Protokoll möglich ist. Seit Windows Server 2003 Release 2 ist alternativ auch eine Kommunikation über HTTP/SOAP möglich (Windows Remote Management – WinRM).

WMI-Provider

Provider

WinMgmt.exe stellt nur das Repository bereit, nicht aber die WMI-Klassen und WMI-Objekte. Diese sind in getrennten DLLs realisiert. WMI ist nicht in einer einzigen DLL realisiert. Für unterschiedliche Systembausteine gibt es unterschiedliche sogenannte WMI-Provider. Für jeden WMI-Provider gibt es eine DLL.

WMI-Provider	Erläuterung
Directory Services Provider	Zugriff auf die ADSI-Informationen aus Verzeichnisdiensten
Event Log Provider	Zugriff auf die Windows-Ereignisprotokolle (nur NT-Produktfamilie)

***Tabelle 14.6**: Ausgewählte WMI-Provider*

WMI-Provider	Erläuterung
Exchange Queue Provider, Exchange Routing TableProvider, ExchangeClusterProvider	Zugriff auf Microsoft Exchange Server ab Version 2000
Microsoft Windows Installer Provider	Zugriff auf Software, die durch den Windows Installer (MSI) installiert wurde
Performance Counters Provider	Zugriff auf rohe Leistungsindikatordaten
Performance Monitor Provider	Zugriff auf Leistungsdaten, wie sie der Windows-Leistungsmonitor sieht
Power Management Event Provider	Ereignisse aus dem Bereich Power Management
Registry Event Provider	Ereignisse bei Änderungen in der Registrierungsdatenbank
Registry Provider	Zugriff auf die Registrierungsdatenbank
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
Windows Installer Provider	Softwareinstallation/-deinstallation
WDM Provider	Zugriff auf Gerätetreiber via Windows Driver Model (WDM)
Win32 Provider	Zugriff auf das Win32-Subsystem

***Tabelle 14.6**: Ausgewählte WMI-Provider (Forts.)*



Nicht alle WMI-Provider werden automatisch registriert und im CIM Repository eingetragen. In der MSDN-Entwicklerbibliothek finden Sie die Informationen darüber, wie die einzelnen Provider registriert werden.

Installierte Treiber

Wie die folgende Tabelle zeigt, sind die vorinstallierten Treiber abhängig von Betriebssystem und installierten Komponenten und Zusatzsoftware. Die Namen sind manchmal sehr lang und manchmal auch leider sehr kurz. Beispiel: *OffProv11* heißt der Provider für Microsoft Office 2003 (Version 11.0). Der IIS-Provider (*IIS__PROVIDER*) ist erst ab IIS Version 6.0 verfügbar. Die beiden folgenden Tabellen zeigen einen Vergleich der Provider für drei typische Systemkonfigurationen.

Tabelle 14.7 *Vergleich der Provider in Windows XP, Windows Server 2003, Windows Vista und Windows 2008 R2*

Kürzel	Installationen
XP SP2	Windows XP Tablet PC Edition Service Pack 2 Microsoft Office 2003 Internet Information Server 5.1 Internet Explorer 6.0 .NET Framework 1.0 .NET Framework 2.0 .NET Framework 3.0 Visual Studio 2005 Team Suite Microsoft SQL Server 2005 Express
2003 R2	Windows Server 2003 Release 2 Microsoft Office 2003 Internet Information Server 6.0 Internet Explorer 6.0 .NET Framework 1.1 .NET Framework 3.0 Visual Studio 2005 Team Suite Microsoft SQL Server 2005 Express
Vista Business	Windows Vista Business .NET Framework 3.0 Internet Information Server 7.0 Internet Explorer 7.0
Windows Server 2008 R2	Windows Server 2008 Release 2 Alle Rollen außer HyperV Internet Explorer 8.0

Provider	Windows XP SP2	Windows Server 2003 R2	Windows Vista Business	Windows Server 2008 R2
AAGProvider				X
ACT_EventProvider		X		
ACTBroker		X		
ACTControllerMethod Provider		X		
ActiveScriptEventConsumer	X	X	X	X
ApplianceManager		X		
BcdProv			X	X
BitsClientProv				X
BitsLightweightServerProv				X
BrokerMethodProv		X		
CIMWin32	X	X	X	X

Tabelle 14.8*: Vergleich der Provider in Windows XP, Windows Server 2003, Windows Vista und Windows Server 2008 R2*

Provider	Windows XP SP2	Windows Server 2003 R2	Windows Vista Business	Windows Server 2008 R2
Cimwin32A	X	X		X
ClearClientConfigAlert Consumer		X		
Cluster Event Provider		X		
CmdTriggerConsumer	X	X		
CommandLineEvent Consumer	X	X	X	X
DecoupledEventProvider	X	X	X	X
DFSProvider		X	X	X
DfsrConfigProv			X	X
DfsrMonitorProv			X	X
DskQuotaProvider	X	X	X	X
EventTraceProv	X	X		
EventViewerConsumer	X	X		
FsrmEventProvider				X
HiPerfCooker_v1	X	X	X	X
ieinfo5	X	X		
IIS__PROVIDER		X	X	X
IPMIPrv		X		X
LogFileEventConsumer	X	X	X	X
Microsoft WMI Forwarding Consumer Provider	X			
Microsoft WMI Forwarding Event Provider	X			
Microsoft WMI Template Association Provider	X			
Microsoft WMI Template Event Provider	X			
Microsoft WMI Template Provider	X			
Microsoft WMI Transient Event Provider	X			
Microsoft WMI Transient Provider	X			

***Tabelle 14.8**: Vergleich der Provider in Windows XP, Windows Server 2003, Windows Vista und Windows Server 2008 R2 (Forts.)*

Provider	Windows XP SP2	Windows Server 2003 R2	Windows Vista Business	Windows Server 2008 R2
Microsoft WMI Transient Reboot Event Provider	X			
Microsoft WMI Updating Consumer Assoc Provider	X			
Microsoft WMI Updating Consumer Event Provider	X			
Microsoft WMI Updating Consumer Provider	X			
Microsoft_SA_AlertEmail ConsumerProvider		X		
Microsoft_SA_DiskProvider		X		
Microsoft|DSLDAPClass AssociationsProvider|V1.0	X	X	X	
Microsoft|DSLDAPClass-AssociationsProvider|V1.0				X
Microsoft|DSLDAP-ClassProvider|V1.0				X
Microsoft|DSLDAP-ClassProvider|V1.0	X	X	X	
Microsoft|DSLDAPInstance Provider|V1.0	X	X	X	
Microsoft|DSLDAPInstance Provider|V1.0				X
Microsoft|NLB_Provider| V1.0		X		
Microsoft|ServerCompo-nentProvider|V1.0				X
MS_CLUSTER_CLASS_ PROVIDER		X		
MS_CLUSTER_PROVIDER		X		
MS_InstalledGameProv			X	
MS_NT_DNS_PROVIDER				X
MS_NT_EVENTLOG_EVENT_ PROVIDER	X	X	X	
MS_NT_EVENTLOG_EVENT_ PROVIDER				X

***Tabelle 14.8**: Vergleich der Provider in Windows XP, Windows Server 2003, Windows Vista und Windows Server 2008 R2 (Forts.)*

Provider	Windows XP SP2	Windows Server 2003 R2	Windows Vista Business	Windows Server 2008 R2
MS_NT_EVENTLOG_ PROVIDER	X	X	X	X
MS_Power_Management_ Event_Provider	X	X	X	
MS_Power_Management_ Event_Provider				X
MS_Shutdown_Event_ Provider	X	X	X	X
MS_SNMP_CLASS_ PROVIDER				X
MS_SNMP_ ENCAPSULATED_ EVENT_PROVIDER				X
MS_SNMP_INSTANCE_ PROVIDER				X
MS_SNMP_REFERENT_ EVENT_PROVIDER				X
MS_VIEW_INSTANCE_ PROVIDER		X	X	X
Msft_ProviderSubSystem	X	X	X	X
MSIProv	X	X	X	X
MSiSCSIInitiatorProvider			X	X
MSiSNSServerProvider				X
MSNFS_PROVIDER				X
MSSQL_Management-Provider	X	X		
MSVDS__PROVIDER		X	X	X
MSVSS__PROVIDER		X	X	X
NamedJobObjectActgInfo-Prov	X	X	X	X
NamedJobObjectLimit-SettingProv				X
NamedJobObjectLimit-SettingProv	X	X	X	
NamedJobObjectProv	X	X	X	X
NamedJobObjectSecLimit SettingProv	X	X	X	

***Tabelle 14.8**: Vergleich der Provider in Windows XP, Windows Server 2003, Windows Vista und Windows Server 2008 R2 (Forts.)*

Provider	Windows XP SP2	Windows Server 2003 R2	Windows Vista Business	Windows Server 2008 R2
NamedJobObjectSecLimitSettingProv				X
Nap_ClientProvider			X	X
NetDiagProv	X			
NetFrameworkv1Provider	X			
NlbsNicProv		X		
NT5_GenericPerfProvider_V1	X	X		
NTEventLogEventConsumer	X	X	X	X
OffProv11	X	X		
PolicSOM	X	X	X	X
PolicStatus	X	X	X	
PowerMeterProvider				X
PowerPolicyProvider				X
ProfileAssociationProviderCimV2				X
ProfileAssociationProviderInterop				X
ProviderSubSystem	X	X	X	X
RegistryEventProvider	X	X	X	X
RegPropProv	X	X	X	X
RegProv	X	X	X	X
ReliabilityMetricsProvider				X
ReplProv1		X		X
RouteEventProvider	X	X	X	X
RouteProvider	X	X	X	X
Rsop Logging Mode Provider	X	X	X	X
Rsop Planning Mode Provider	X	X	X	X
SANTEventLogFilterEvent ConsumerProvider		X		
SCM Event Provider	X	X	X	X
SECRCW32	X	X	X	X

***Tabelle 14.8**: Vergleich der Provider in Windows XP, Windows Server 2003, Windows Vista und Windows Server 2008 R2 (Forts.)*

Provider	Windows XP SP2	Windows Server 2003 R2	Windows Vista Business	Windows Server 2008 R2
ServerFeatureProvider				X
ServiceModel	X	X	X	X
SessionBrokerTargetEvent-Provider				X
SessionProvider	X	X	X	X
SmonlogProv	X	X		
SMTPEventConsumer	X	X	X	X
SoftwareLicensingProduct_Provider			X	
SoftwareLicensingService_Provider			X	
SoftwareLicensingTokenActivationLicense_Provider				X
SppProvider				X
SQLServerEventProvider	X	X		
SrmEventProvider		X		
Standard Non-COM Event Provider	X	X	X	
Standard Non-COM Event Provider				X
SystemConfiguration-ChangeEvents	X	X	X	
SystemConfiguration-ChangeEvents				X
SystemRestoreProv	X		X	
TrustPrv		X		X
UserProfileProvider				X
VolumeChangeEvents	X	X	X	X
WBEMCORE	X	X	X	X
WebAdministration-Provider			X	X
WhqlProvider	X	X	X	X
Win32_Encryptable-VolumeProvider				X
Win32_OfflineFilesProvider			X	X

***Tabelle 14.8**: Vergleich der Provider in Windows XP, Windows Server 2003, Windows Vista und Windows Server 2008 R2 (Forts.)*

Provider	Windows XP SP2	Windows Server 2003 R2	Windows Vista Business	Windows Server 2008 R2
Win32_OsBaseline			X	
Win32_SDVirtualDesktop Server_Prov				X
Win32_TpmProvider			X	X
Win32_TSCentralPublisher_ Prov				X
Win32_TSPublished Application_Prov				X
Win32_WIN32_PROXY_ Prov	X	X		
Win32_WIN32_SESSION BROKERFARM_Prov				X
Win32_WIN32_SESSION BROKERFARMACCOUNT_ Prov				X
Win32_WIN32_ SESSIONBROKER-SERVICEPROPERTIES_Prov				X
Win32_WIN32_SESSION-BROKERTARGET_Prov				X
Win32_WIN32_SESSION-DIRECTORYCLUSTER_Prov				X
Win32_WIN32_SESSION-DIRECTORYSERVER_Prov				X
Win32_WIN32_SESSION-DIRECTORYSESSION_Prov				X
Win32_WIN32_SESSION-DIRECTORYVMMPLUGIN_ Prov				X
Win32_WIN32_TERMINAL_ Prov	X	X	X	X
Win32_WIN32_ TERMINALSERVICE_Prov	X	X	X	X
Win32_WIN32_TERMINAL-SERVICESETTING_Prov	X	X	X	X
Win32_WIN32_TERMINAL-SERVICETOSETTING_Prov	X	X	X	X
Win32_WIN32_TERMINAL-TERMINALSETTING_Prov	X	X	X	X

***Tabelle 14.8**: Vergleich der Provider in Windows XP, Windows Server 2003, Windows Vista und Windows Server 2008 R2 (Forts.)*

Provider	Windows XP SP2	Windows Server 2003 R2	Windows Vista Business	Windows Server 2008 R2
Win32_WIN32_TERMSERVLICENSING_Prov				X
Win32_WIN32_TSACCOUNT_Prov	X	X	X	X
Win32_WIN32_TSCLIENTSETTING_Prov	X	X	X	X
Win32_WIN32_TSENVIRONMENTSETTING_Prov	X	X	X	X
Win32_WIN32_TSGENERALSETTING_Prov	X	X	X	X
Win32_WIN32_TSLOGONSETTING_Prov	X	X	X	X
Win32_WIN32_TSNETWORKADAPTERLISTSETTING_Prov	X	X	X	X
Win32_WIN32_TSNETWORKADAPTERSETTING_Prov	X	X	X	X
Win32_WIN32_TSPERMISSIONSSETTING_Prov	X	X	X	X
Win32_WIN32_TSREMOTECONTROLSETTING_Prov	X	X	X	X
Win32_WIN32_TSSESSIONDIRECTORY_Prov	X	X	X	X
Win32_WIN32_TSSESSIONDIRECTORYSETTING_Prov	X	X	X	X
Win32_WIN32_TSSESSIONSETTING_Prov	X	X	X	X
Win32_WIN32_TSVIRTUALIP_Prov				X
Win32_WIN32_TSVIRTUALIPSETTING_Prov				X
Win32_WIN32_WINDOWSPRODUCTACTIVATION_Prov	X	X		

***Tabelle 14.8**: Vergleich der Provider in Windows XP, Windows Server 2003, Windows Vista und Windows Server 2008 R2 (Forts.)*

Provider	Windows XP SP2	Windows Server 2003 R2	Windows Vista Business	Windows Server 2008 R2
Win32_WinSAT			X	
Win32ClockProvider	X	X	X	X
WMI Kernel Trace Event Provider	X	X	X	X
WMI Self-Instrumentation Event Provider	X	X	X	X
WMIEventProv	X	X	X	X
WmiPerfClass			X	X
WmiPerfInst			X	X
WMIPingProvider	X	X	X	X
WMIProv	X	X	X	X

***Tabelle 14.8**: Vergleich der Provider in Windows XP, Windows Server 2003, Windows Vista und Windows Server 2008 R2 (Forts.)*

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.

MOF für Microsoft Office

Das nachfolgende Listing zeigt Ausschnitte aus der Datei *msioff9.mof*, welche die MOF-Beschreibung für Informationen über Microsoft Office liefert. Die dort definierten Klassen entsprechen den im „MSInfo" anzeigbaren Daten. Das MOF-File definiert zunächst einen neuen Namensraum *MSAPPS* und dann über eine CLSID den Provider, der die Funktionalität der im Folgenden definierten Klassen implementiert.

***Listing 14.1** Ein kleiner Ausschnitt aus dem MOF-File MSIOff9.mof*

```
//************************************************************
//* File: MSIOff9.mof - Office Extension MOF File for MSInfo 5.0
//************************************************************
//***Creates namespace for MSAPPS
#pragma namespace ("\\\\.\\Root")
instance of __Namespace
{
    Name = "MSAPPS";
};
//* Declare an instance of the __Win32Provider so as to "register"
the
//* Office provider.
instance of __Win32Provider as $P
{
    Name = "OffProv";
```

```
    ClsId = "{D2BD7935-05FC-11D2-9059-00C04FD7A1BD}";
};

//* Class: Win32_WordDocument
//* Derived from:
[dynamic: ToInstance, provider("OffProv")]
class Win32_WordDocument
{
    [key, read: ToInstance ToSubClass] string Name;
    [read: ToInstance ToSubClass] string Path;
    [read: ToInstance ToSubClass] real32 Size;
    [read: ToInstance ToSubClass] datetime CreateDate;
};
//* Class: Win32_AccessDatabase
//* Derived from:
 [dynamic: ToInstance, provider("OffProv"), Singleton:
DisableOverride ToInstance ToSubClass]
class Win32_AccessDatabase
{
    [read: ToInstance ToSubClass] string Name;
    [read: ToInstance ToSubClass] string Path;
    [read: ToInstance ToSubClass] real32 Size;
    [read: ToInstance ToSubClass] datetime CreateDate;
    [read: ToInstance ToSubClass] string User;
    [read: ToInstance ToSubClass] string JetVersion;
};
```



Einige Provider werden nicht automatisch in das WMI-Repository eingebunden. In diesen Fällen muss man die zugehörige MOF-Datei erst „kompilieren“ mit dem Befehl:

```
mofcomp.exe Dateiname.mof
```

Instanzenbeschreibung

Auch Instanzen können in MOF beschrieben werden, dabei werden die Attribute mit ihren Werten angeführt. Nachstehend sieht man die MOF-Repräsentation einer Instanz der Klasse `Win32_ComputerSystem`.

```
instance of Win32_ComputerSystem
{
    AdminPasswordStatus = 3;
    AutomaticResetBootOption = TRUE;
    AutomaticResetCapability = TRUE;
    BootROMSupported = TRUE;
    BootupState = "Normal boot";
    Caption = "F108";
    ChassisBootupState = 3;
    CreationClassName = "Win32_ComputerSystem";
    CurrentTimeZone = 120;
    DaylightInEffect = TRUE;
    Description = "AT/AT COMPATIBLE";
```

```
    Domain = "FBI.net";
    DomainRole = 1;
    EnableDaylightSavingsTime = TRUE;
    FrontPanelResetStatus = 3;
    InfraredSupported = FALSE;
    KeyboardPasswordStatus = 3;
    Manufacturer = "System Manufacturer";
    Model = "System Name";
    Name = "BYFANG";
    NetworkServerModeEnabled = TRUE;
    NumberOfProcessors = 1;
    OEMStringArray = {"0", "0"};
    PartOfDomain = TRUE;
    PauseAfterReset = "-1";
    PowerOnPasswordStatus = 3;
    PowerState = 0;
    PowerSupplyState = 3;
    PrimaryOwnerName = "www.IT-Visions.de";
    ResetCapability = 1;
    ResetCount = -1;
    ResetLimit = -1;
    Roles = {"LM_Workstation", "LM_Server", "Print", "NT",
"Potential_Browser"};
    Status = "OK";
    SystemStartupDelay = 30;
    SystemStartupOptions = {"\"Microsoft Windows XP Professional\"
       /fastdetect"};
    SystemStartupSetting = 0;
    SystemType = "X86-based PC";
    ThermalState = 3;
    TotalPhysicalMemory = "536309760";
    UserName = "FBI\\hs";
    WakeUpType = 6;
};
```

WMI-Sicherheit

Sicherheits-funktionen

WMI basiert auf COM und verwendet die COM-Sicherheitsfunktionen und die entsprechend verfügbaren Security Provider. Sicherheitseinstellungen können auf der Ebene eines jeden Namensraums 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 Namensraums geprüft, zu dem das Objekt gehört. Die Vergabe von Zugriffsrechten auf Objekt- oder Klassenebene unterstützt WMI bislang nicht.

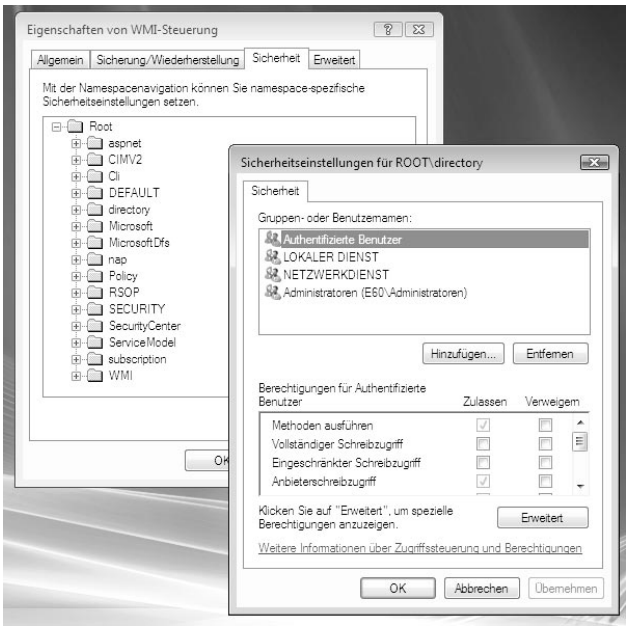


Abbildung 14.2
Einstellen der Zugriffsberechtigungen in der MMC „WMI-Steuerung“

Impersonifizierung

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.“

Programmatische Sicherheit

Im Gegensatz zu anderen Komponenten erlaubt WMI Vorgaben für die COM-Sicherheit durch den Client. Sowohl Impersonifizierungs- als auch Authentifizierungsmodus können beim Verbindungsaufbau eingestellt werden. Der Client kann ab Windows 2000 auch den Security Service Provider (SSPI) zwischen der NT 4.0-LAN-Manager-(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.

Privilegien

Privilegien sind Zusatzrechte, die bei Nutzung einiger WMI-Klassen benötigt werden und bei der Instanziierung gesetzt werden müssen. Ohne diese Zusatzangaben ist die Klasse nicht nutzbar. Die Zusatzangaben werden entweder im WMI-Pfad gesetzt oder aber über das Unterobjekt `Security_.Privileges` in der Klasse `SWbemLocator`. Welche Privilegien eine WMI-Klasse erwartet, erfährt man durch den Klassen-Qualifier `EnumPrivileges`.

WMI Query Language (WQL)

WQL

WMI erlaubt Suchanfragen in einer Syntax, 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.

Das WQL-`SELECT` unterstützt neben den Standardschlüsselwörtern `FROM`, `WHERE`, `GROUP BY`, `HAVING` und `WITHIN` auch die nicht in ADSI-SQL definierten Schlüsselwörter `ASSOCIATORS OF` und `REFERENCES OF`.

Operatoren

Unterstützte Operatoren sind =, <, >, <=, >=, != (alternativ: <>) sowie `IS NULL`, `IS NOT NULL` und `ISA`. Der `ISA`-Operator ermöglicht die Abfrage nach Unterklassen einer bestimmten Klasse. Wenn `Großstadt` eine Unterklasse von `Stadt` ist, dann erfüllt ein Objekt `München` vom Typ `Großstadt` auch die Bedingung `ISA Stadt`. Der `like`-Operator für den Zeichenketten-Mustervergleich wird erst ab Windows XP unterstützt.

Typen von WQL-Anfragen

WMI-Suchanfragen

WMI unterstützt drei Typen von WQL-Anfragen:

- Anfragen nach Instanzen (*Data Queries*)
- Anfragen nach Schemainformationen (*Schema Queries*)
- Definitionen von Ereignisfiltern (*Event Queries*), die nach Änderungen von Klassen und Instanzen fragen

Datenabfragen (Data Queries)

Data Queries

Eine Datenabfrage hat die allgemeine Form:

```
SELECT attributliste FROM class WHERE bedingung
```

Dabei ist `Class` ein beliebiger WMI-Klassenname. Die Ergebnismenge lässt sich durch die Angabe von Attributnamen und die Verwendung einer `FROM`-Klausel hinsichtlich der Breite und Länge einschränken. Andere Schlüsselwörter werden nicht unterstützt.

Beispiele

Beispiele für Datenabfragen zeigt die folgende Tabelle.

WQL	Erläuterung
`SELECT * FROM Win32_Service WHERE state='running' and startmode='manual'`	Alle Windows-Dienste, die laufen, aber manuell gestartet wurden
`SELECT Name, CategoryId FROM Win32_ComponentCategory`	Name und CATID aller Komponentenkategorien
`SELECT IPAddress FROM Win32_NetworkAdapter-Configuration 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 Ereignisprotokoll „Anwendung“

Tabelle 14.9 *Beispiele für WQL-Datenabfragen*

Schemaabfragen (Schema Queries)

Schema-Queries

Anfragen zum Schema haben die Form:

```
SELECT attributliste FROM META_CLASS WHERE bedingung
```

wobei `META_CLASS` hier ein feststehender Ausdruck ist. Andere Schlüsselwörter werden nicht unterstützt. Mit der `WHERE`-Klausel werden das zu beobachtende Managed Object (MO) und die in ihm zu beobachtenden Attribute definiert. Schemaabfragen können mit `WMI_Print-Query(WQL)` ausgegeben werden.

WQL	Erläuterung
`SELECT * FROM meta_class WHERE __Class = "Win32_LogicalDisk"`	Zugriff auf die WMI-Klasse `Win32_LogicalDisk`
`SELECT * FROM meta_class WHERE __this ISA "Win32_LogicalDisk"`	Zugriff auf alle von `Win32_LogicalDisk` abgeleiteten Klassen

Tabelle 14.10 *Beispiele für Schemaabfragen*

Ereignisabfragen (Event Queries)

Ereignisabfragen

Eine Ereignisabfrage bezieht sich immer auf eine Ereignisklasse.

Mit der `WHERE`-Klausel werden die zu suchenden Klassen entweder direkt über ihren Namen (__Class =) oder über den Namen einer Oberklasse (__this ISA) festgelegt.

.

```
SELECT * FROM eventklasse WHERE bedingung
```

Alle Ereignisklassen sind Unterklassen eines Ereignistyps. Die Ereignistypen wiederum sind Unterklassen der Klasse `__Event`. Sie sind in der Regel an dem führenden doppelten Unterstrich und der Endung auf `Event` erkennbar. WMI unterscheidet vier Typen von Ereignissen (siehe Tabelle).

Tabelle 14.11 *WMI-Ereignisklassen. Die Ereignistypen sind die Oberklassen zu den rechts genannten Ereignisklassen.*

Ereignistyp (Oberklasse)	Ereignisklasse

`__ClassOperationEvent`	`__ClassCreationEvent`
	`__ClassDeletionEvent`
	`__ClassModificationEvent`

`__ExtrinsicEvent`	`__SystemEvent` `Win32_PowerManagementEvent`

`__InstanceOperationEvent`	`__InstanceCreationEvent`
	`__InstanceDeletionEvent`
	`__InstanceModificationEvent`

`__NamespaceOperationEvent`	`__NamespaceCreationEvent`
	`__NamespaceDeletionEvent`
	`__NamespaceModificationEvent`



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%).

Zusätzliche Schlüsselwörter

Ereignisabfragen unterstützen als zusätzliche SQL-Schlüsselwörter `WITHIN`, `GROUP BY` und `HAVING`:

- Dabei gibt `WITHIN 10` das Abfrageintervall in Sekunden an (wird immer gebraucht, wenn es keinen speziellen Ereignissender gibt!).
- `GROUP` bündelt eine Anzahl von Einzelereignissen zu einem Gesamtereignis.
- `HAVING` dient der Definition einer Bedingung innerhalb der Gruppierung.

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 *Stopped* 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 *System* und *Application* löst ein Ereignis aus.

Tabelle 14.12 *Beispiele für Ereignisabfragen*

WMI-Ereignissystem

Ereignisarten

WMI bietet ein komplexes System für Ereignisse. Es gibt zwei Gruppen von Ereignissen:

- **Intrinsic Events** sind Ereignisse, die direkt Klassen oder Objekte im WMI-Repository verändern oder betreffen, z.B. das Löschen einer Instanz (`__InstanceDeletionEvent`), das Verändern einer Instanz (`__InstanceModificationEvent`) oder der Aufruf einer Methode (`__MethodInvocationEvent`). Zu den Intrinsic Events gehören auch Zeitgeberereignisse (Timer Events), z.B. `__AbsoluteTimerInstruction` und `__IntervallTimerInstruction`. Der Name dieser Ereignisse beginnt mit einem doppelten Unterstrich. Alle Klassen sind direkt oder indirekt abgeleitet von `__Event`.
- **Extrinsic Events** sind speziellere Ereignisse, die von irgendeinem Teil des Systems ausgelöst werden, aber nicht direkt auf das CIM-Repository wirken müssen, z.B. Ändern eines Schlüssels in der Registrierungsdatenbank (`RegistryValueChangeEvent`), Ändern der Systemkonfiguration (`Win32_SystemConfigurationChangeEvent`) und Herunterfahren des Computers (`Win32_ComputerShutdownEvent`). Alle diese Klassen sind abgeleitet von `__ExtrinsicEvent`.

WMI-Ereignisse werden ausgelöst von *Ereignisprovidern (Event Provider)*, die im CIM-Repository einmalig registriert werden müssen.

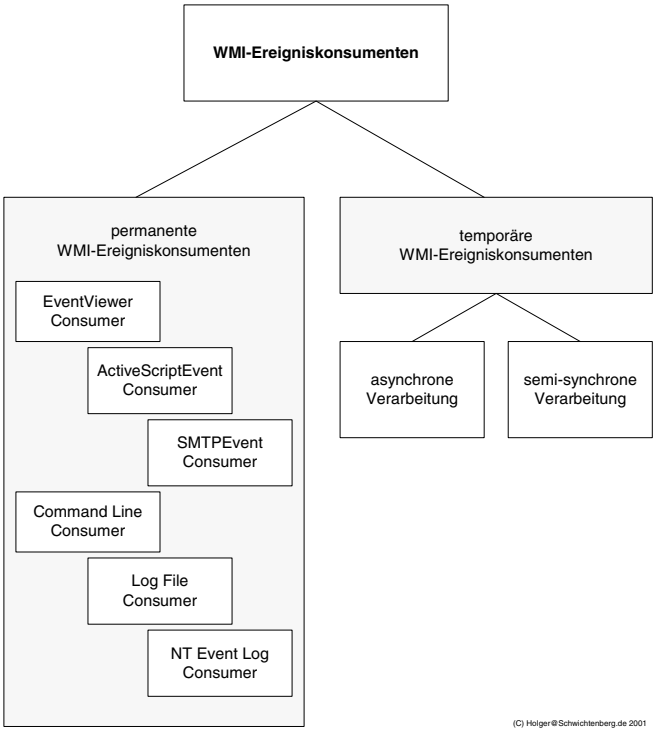
Konsumenten

WMI-Ereignisse werden behandelt von *Ereigniskonsumenten (Event Consumers)*. Ereigniskonsumenten registrieren sich bei WMI für bestimmte Ereignisse. Der Ereigniskonsument 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 Skripts/Programms 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 14.3 Überblick über die Ereigniskonsumenten



Permanente Ereigniskonsumenten sind z.B.:

EventViewer-Consumer

- Der `EventViewerConsumer` (im Namensraum */root/cimv2*), der die aufgetretenen Ereignisse in einem Bildschirmfenster, dem WMI Event Viewer, darstellt. Der Event Viewer wird in *Kapitel „WMI-Werkzeuge“* vorgestellt.

ActiveScript-EventConsumer

- Der `ActiveScriptEventConsumer` (im Namensraum */root/default*), der bei Auftreten eines Ereignisses ein Active Script ausführt.

SMTPEvent-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 `SMTPEvent-`

`Consumer` ist standardmäßig nicht registriert. Um ihn benutzen zu können, muss die Datei *smtpcons.mof*, die sich im Verzeichnis *%System-Root%\Winnt\wbem* befindet, mit Hilfe des MOF-Compilers *(mofcomp.exe)* kompiliert und ins WMI-Repository aufgenommen werden. Der Provider wird im Namensraum *root\default* registriert.

Seit Windows XP gibt es drei weitere permanente Ereigniskonsumenten:

- `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

Filter

Der Ereigniskonsument definiert sein Interesse gegenüber WMI in Form eines WMI-Filters. Ein WMI-Filter ist im Wesentlichen eine *WQL Event Query* (vgl. Unterkapitel zu WQL-Abfragen).

Administration

Ein Ereigniskonsument ist eine Instanz einer von der Systemklasse `__EventConsumer` erbenden Klasse. Ein WMI-Filter ist eine Instanz der Systemklasse `__EventFilter`. Die Bindung zwischen einem Consumer und einem Filter ist als Instanz von `__FilterToConsumerBinding` gespeichert.

Es gibt zwei Möglichkeiten, Ereigniskonsument und Filter zu definieren und aneinander zu binden:

- Über das Werkzeug *WMI Event Registration* aus dem *WMI Administrative Tools*
- Über Programmcode. Dadurch, dass Consumer, Filter und Bindungen selbst wieder als WMI-Objekte gespeichert werden, können diese leicht auch per WMI automatisiert verwaltet werden.

Provider

Kern des Ereignissystems ist der *Event Provider.* Er informiert WMI über Veränderungen in der Managementumgebung oder im Repository. WMI leitet die Ereignisse an die für dieses Ereignis registrierten Konsumenten weiter.

WMI-Werkzeuge

WMI kann man mit folgenden Werkzeugen verwenden:

- WMIC-Kommandozeilenwerkzeug (*wmic.exe*)
- Die kostenlosen Werkzeuge im Rahmen der WMI Administrative Tools
- WMI-Scripting-Komponente für Windows Script Host (WSH)
- Windows PowerShell (WPS) über das Commandlet `Get-WmiObject` und die eingebauten PowerShell-Typen `[WMI]`, `[WMICLASS]` und `[WMISEARCHER]`
- Kommerzielle Systemmanagementwerkzeuge wie Microsoft System Center oder HP OpenView (nicht in diesem Buch beschrieben)

Die MMC-Konsole für WMI (z.B. enthalten in der „Computerverwaltung“) dient nicht dem Zugang zu WMI-Informationen, sondern nur der Sicherheitskonfiguration (wer hat Zugang zu den Informationen?) sowie der Sicherung und Wiederherstellung des WMI-Repositorys.

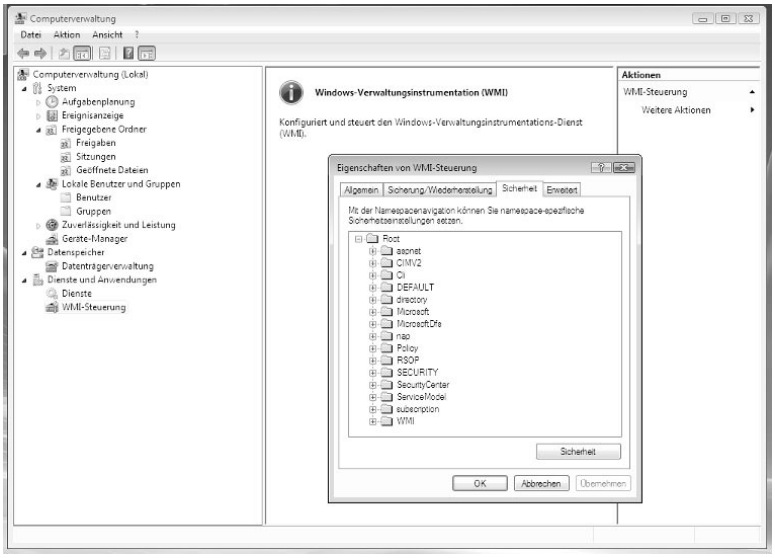


Abbildung 14.4 *WMI-Steuerung in der MMC „Computerverwaltung“*



Daran, dass es in Windows Vista kein mitgeliefertes grafisches Werkzeug für den Zugang zu den eigentlichen Daten in WMI gibt, sieht man schon, dass WMI als Plattform für Scripting und nicht für die GUI-basierte Administration betrachtet wird.

WMI Object Browser

Der WMI Object Browser ist eines der Werkzeuge in den WMI Administrative Tools. Es ermöglicht, die auf dem lokalen oder einem entfernten System vorhandenen WMI-Objekte zu betrachten und entlang der Hierarchie der Objekte das System zu erforschen, mit dem Ziel, Objekte und Attribute zu finden, welche die gewünschten Systeminformationen enthalten. An vielen Stellen können WMI-Objekte auch verändert werden (z.B. Name eines Laufwerks ändern) oder es ist möglich, Aktionen auf den Objekten zu initiieren (z.B. Festplatte prüfen).



Das Werkzeug *WMI Object Browser* läuft als ActiveX-Komponente innerhalb des Internet Explorers. Sie müssen ActiveX zulassen, um das Werkzeug nutzen zu können. Eine deutsche Version ist nicht verfügbar. Da die Hilfetexte zu den WMI-Klassen aber im WMI-Repository stehen, sind zumindest davon viele in Deutsch verfügbar.

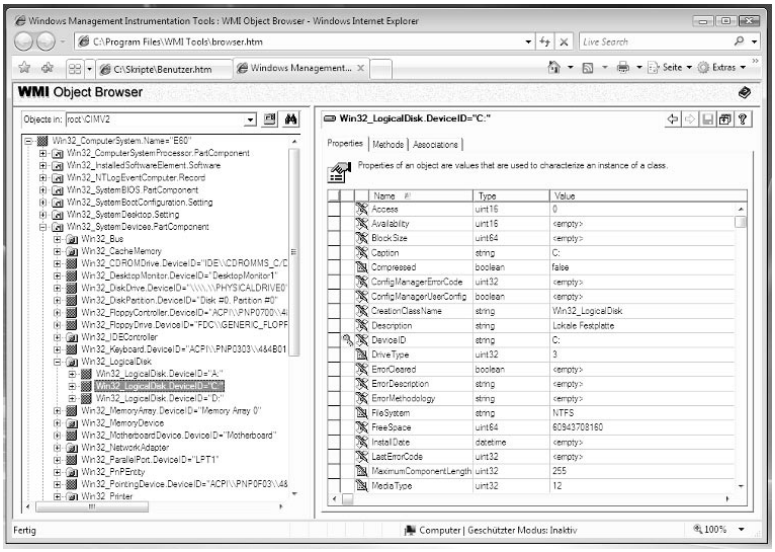


Abbildung 14.5
Der WMI Object Browser zeigt an, dass der Computer drei Laufwerke besitzt. Die Details zu Laufwerk „C" werden hier dargestellt.

Weitere Werkzeuge

Weitere Werkzeuge in den WMI Administrative Tools sind:

- Das WMI CIM Studio setzt im Gegensatz zum WMI Object Browser auf der Klassenebene an und stellt die Vererbungshierarchie der CIM-Klassen dar. Diese wird in der linken Fensterhälfte, dem Class Explorer, gezeigt. Die rechte Fensterhälfte (genannt Class Viewer) ähnelt der Ansicht des WMI Object Browsers: Hier werden die Attribute, Methoden und Beziehungen angezeigt.
- Das WMI Event Registration Tool ist ein GUI zur Konfiguration von Ereigniskonsumenten.
- Der WMI Event Viewer ist das einzige unter den WMI Administrative Tools, das keine HTML-Anwendung, sondern eine ausführbare Datei (wbemeventviewer.exe) ist. Der Event Viewer ist ein permanenter WMI-Ereigniskonsument, der durch eine `__EventFilter`-Instanz definierte Ereignisse auf dem Bildschirm darstellt. Der Event Viewer wird über die WMI-Klasse `Event ViewerConsumer` konfiguriert. Diese Konfiguration können Sie über das WMI Event Registration Tool oder über den Event Viewer selbst durchführen.

WMI-Programmierschnittstellen

Microsoft stellt für WMI drei Programmierschnittstellen bereit:

- Die COM-basierte Softwarekomponente *SWbemDisp.dll* mit dem WMI-COM-Objektmodell, in dessen Zentrum die Klasse `Wbem-Scripting.SWbemObject` steht. Fernzugriff ist über DCOM möglich.
- Die COM-basierte Softwarekomponente *WsmAuto.dll*, die einen Fernzugriff über HTTP und SOAP ermöglicht
- Die .NET-basierte Softwarekomponente *System.Management.dll* mit einem Objektmodell, in dessen Zentrum die Klassen `System.Management.ManagementObject` und `System.Management.ManagementClass` stehen



Metaobjektmodelle

Alle oben genannten Komponenten sind **Metaobjektmodelle**, d.h., es gibt dort nicht für jede der unzähligen WMI-Klassen eine Entsprechung, sondern es gibt einige wenige Klassen, deren Instanzen durch Abgabe von WMI-Pfaden auf die eigentlichen WMI-Objekte abgebildet werden.

System.Management

Die Windows PowerShell verwendet primär die .NET-Komponente *System.Management.dll* mit dem Namensraum `System.Management`. Ein Zugriff auf die COM-Komponenten ist aber auch möglich, nur umständlicher und wird in diesem Buch nicht gezeigt.

Zentrale Klassen des neuen Objektmodells (siehe Bildschirmabbildung) von `System.Management` sind:

- `ManagementObject`

  Diese Klasse repräsentiert ein WMI-Objekt.

- `ManagementClass`

  Diese Klasse repräsentiert eine WMI-Klasse. `ManagementClass` ist von `ManagementObject` abgeleitet.

- `ManagementBaseObject`

  Beide Klassen sind von `ManagementBaseObject` abgeleitet. Diese Klasse ist nicht abstrakt, sondern wird an verschiedenen Stellen im Objektmodell auch verwendet.

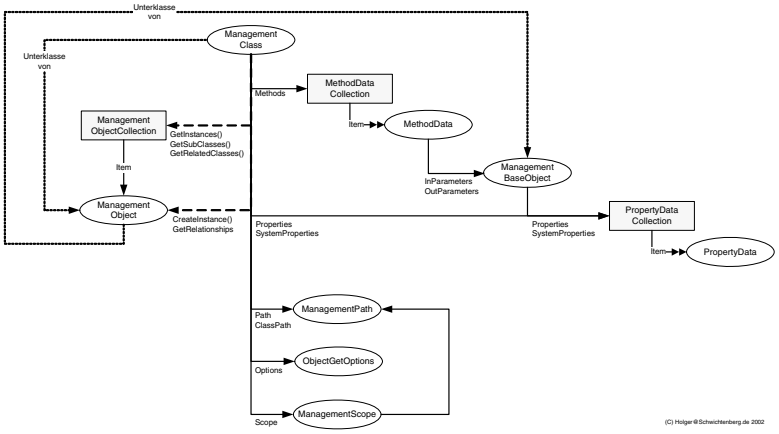


Abbildung 14.6
Objektmodell von System.Management

Bei *System.Management.dll* dient die Klasse `ManagementObject` als Metaklasse für alle WMI-Klassen, d.h., eine Instanz von `ManagementObject` wird bei seiner Erzeugung durch Verwendung eines WMI-Pfads an ein WMI-Objekt gebunden und bildet dieses ab. Leider ist die Abbildung nicht so leicht zu verwenden, wie man es sich wünschen würde, denn man muss alle Attribute über die `PropertyDataCollection` ansteuern (siehe obige Abbildung des Objektmodells) und die Methodenaufrufe umständlich durch `InvokeMethod()` ausführen.

Im Folgenden werden Sie sehen, dass die Windows PowerShell den Zugriff auf dieses komplexe Objektmodell stark vereinfacht.



14.2 WMI in der Windows PowerShell

Möglichkeiten

Die Windows PowerShell bietet die Möglichkeit zum Zugriff auf das lokale WMI-Repository und auch WMI-Repositories auf entfernten Systemen.

Dafür bietet die PowerShell folgende Konstrukte:

- Die Commandlets `Get-WmiObject` (ab PowerShell 1.0), `Remove-WmiObject`, `Set-WmiInstance` und `Invoke-WmiMethod` (ab PowerShell 2.0)
- Die eingebauten PowerShell-Typen `[WMI]`, `[WMICLASS]` und `[WMISEARCHER]`
- Den PowerShell-WMI-Objektadapter, der den Zugriff auf WMI-Objekte vereinfacht

Diese Konzepte werden in den folgenden Kapiteln aufgabenorientiert erläutert.

14.3 Zugriff auf einzelne WMI-Objekte

Einzelobjekte Um auf ein bestimmtes WMI-Objekt gezielt zuzugreifen, gibt es drei Möglichkeiten:

- Verwendung des Commandlets `Get-Wmiobject` mit einem Filter und optional mit der Angabe eines Computernamens
- Verwendung der eingebauten PowerShell-Typen `[WMI]` und `[WMIClass]` mit WMI-Pfaden
- Direkte Instanziierung der Klassen `System.Management.ManagementObject` bzw. `System. Management.ManagementClass` jeweils unter Angabe eines WMI-Pfads im Konstruktor



Klassen, von denen es sowieso nur immer eine Instanz geben kann, lassen sich ohne Filter aufrufen:

```
Get-Wmiobject win32_Computersystem
Get-Wmiobject win32_OperatingSystem
```

	Get-WmiObject mit Filter	Eingebaute Power-Shell-Typen	Direkte Instanziierung
WMI-Objekt aus einer WMI-Klasse mit einem Schlüsselattribut	`Get-WmiObject Win32_LogicalDisk -Filter "DeviceID='C:'"`	`[WMI] "\\.\root\cimv2:Win32_LogicalDisk.DeviceID='C:'"`	`New-Object System.Management.ManagementObject("\\.\root\cimv2:Win32_LogicalDisk.DeviceID='C:'")`
WMI-Objekt aus einer WMI-Klasse mit zwei Schlüsselattributen	`Get-Wmiobject Win32_Account -filter "name='hs' and domain='FBI'"`	`[WMI] "\\.\root\cimv2:Win32_UserAccount.Domain='FBI',Name='hs'"`	`New-Object System.Management.ManagementObject("\\.\root\cimv2:Win32_UserAccount.Domain='FBI',Name='hs'")`
WMI-Objekt auf einem entfernten System	`Get-WmiObject Win32_LogicalDisk -Filter "DeviceID='C:'" -computer "SERVERF112"`	`WMI] "\\SERVERF112\root\cimv2:Win32_UserAccount.Domain='FBI',Name='hs'"`	`New-Object System.Management.ManagementObject("\\SERVERF112\root\cimv2:Win32_UserAccount.Domain='FBI',Name='hs'")`
WMI-Klasse	`Nicht möglich`	`[WMICLASS] "\\.\root\cimv2:Win32_UserAccount"`	`New-Object System.Management.ManagementClass("\\E01\root\cimv2:Win32_UserAccount")`

***Tabelle 14.13**: Beispiele zum Zugriff auf einzelne WMI-Objekte*

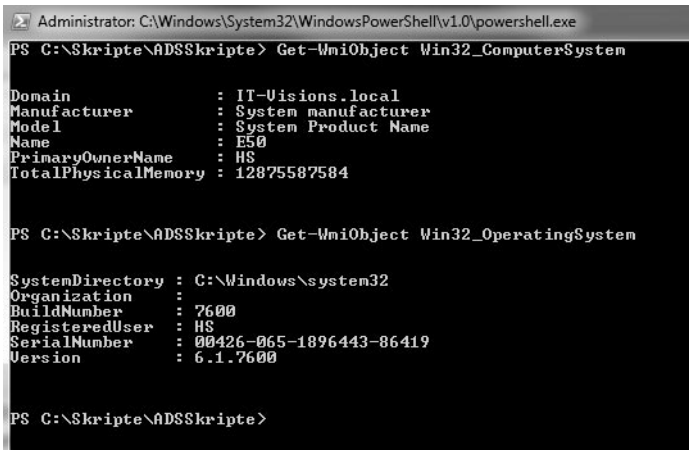


Abbildung 14.7 *Win32_Computersystem und Win32_OperatingSystem gibt es sowieso immer nur einmal im WMI-Repository.*



Bei der Verwendung der Typbezeichner `[WMI]` und `[WMIClass]` wird häufig übersehen, den Pfadnamen zu klammern, wenn dieser zusammengesetzt wird.

Falsch:

```
$Computer = "F111"
[WMI] "\\" + $Computer + "\root\cimv2:Win32_
PingStatus.Address='SERVERF112'"
```

Richtig:

```
$Computer = "ServerF111"
[WMI] ("\\" + $Computer + "\root\cimv2:Win32_
PingStatus.Address='SERVERF112'")
```

14.4 Der PowerShell-WMI-Objektadapter

Kapselung der WMI-Metaklassen

Eigentlich ist der Zugriff auf WMI-Objekte über .NET nicht gerade „geschmeidig“, weil immer umständlich die `PropertyDataCollection` anzusprechen ist. Hier bietet die Windows PowerShell auf Basis des Extended Type Systems (ETS) eine Vereinfachung, denn die PowerShell erstellt durch den eingebauten WMI-Objektadapter dynamisch wieder Objekte, die den WMI-Klassen entsprechen. Diesen komplexen Zusammenhang veranschaulicht nachstehende Grafik.

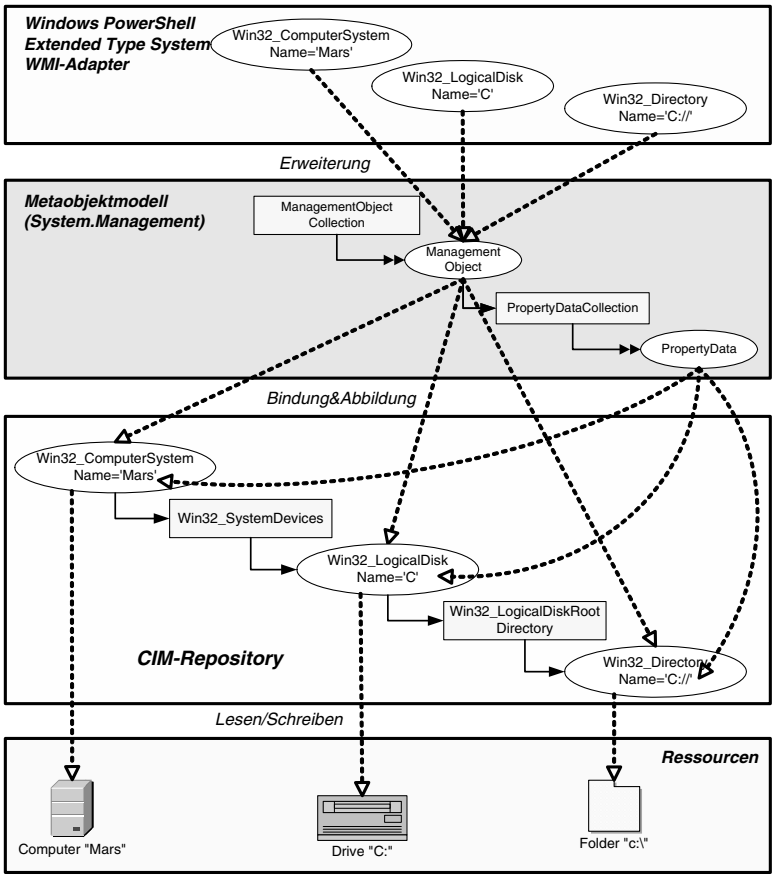


Abbildung 14.8
Abbildung von WMI-Objekten in der PowerShell



Auf die Frage, wozu man als fortgeschrittener PowerShell-Nutzer diesen Mechanismus überhaupt kennen muss und wissen muss, was im Hintergrund passiert, gibt es drei Antworten:

- Damit Sie in der Lage sind, Codebeispiele, die mit dem WSH oder .NET arbeiten, auf die PowerShell zu übertragen
- Damit Sie verstehen, in welcher Dokumentation Sie suchen müssen
- Damit Sie auch verstehen, woran es liegen könnte, wenn etwas nicht funktioniert

WMI ist nicht die einzige Komponente, für welche die PowerShell einen solchen PowerShell-Objektadapter bereitstellt. Der Zugriff auf Verzeichnisdienste, Datenbanken und XML-Dokumente funktioniert ähnlich.

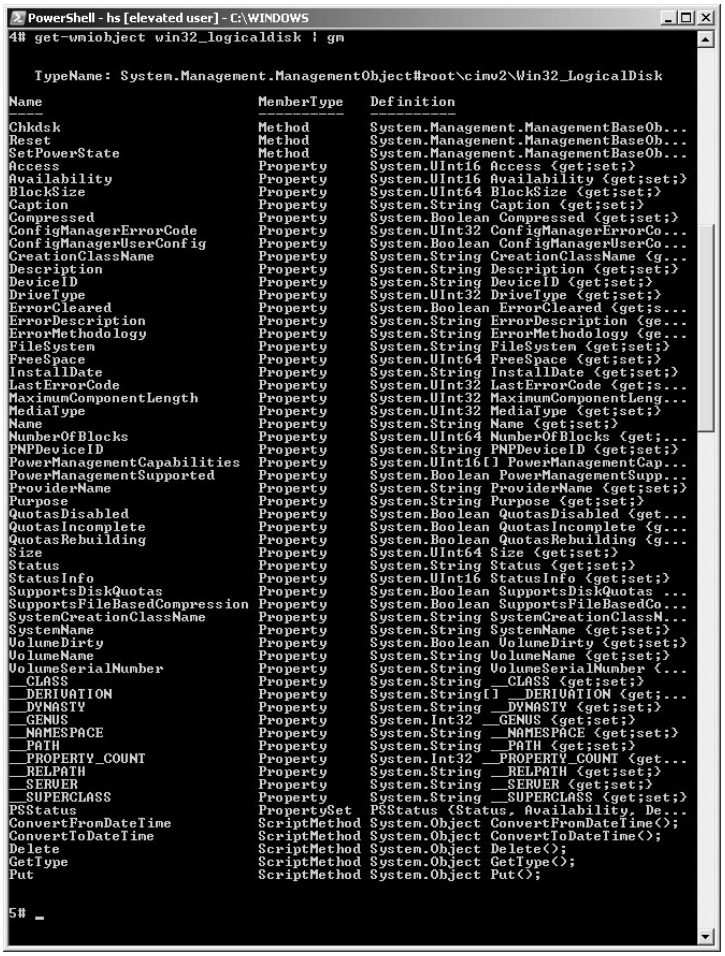
Die Windows PowerShell hat ihre eigene Art und Weise, die durch den WMI-Objektadapter erzeugten Klassen zu benennen. Sie verwendet den Namen der .NET-Metaklasse (`System.Management.ManagementObject`) und den Pfad der WMI-Klasse getrennt durch das Zeichen „#“:

```
System.Management.ManagementObject#root\cimv2\Win32_LogicalDisk
```

In der nachstehenden Bildschirmabbildung sehen Sie, dass das Commandlet `Get-Member` solche Typnamen ausgibt.



Die von `Get-Member` angezeigten Attribute (Property) und Methoden (Method) sind keine Mitglieder der .NET-Klasse `ManagementObject`, sondern der WMI-Klasse `Win32_LogicalDisk`. Wenn Sie Hilfeinformationen zu den Objekten in der Pipeline suchen, müssen Sie also in die Dokumentation des WMI-Schemas [MSDN05] schauen, nicht in die Dokumentation von `System.Management` [MSDN06].



```
PowerShell - hs [elevated user] - C:\WINDOWS
4# get-wmiobject win32_logicaldisk | gm


   TypeName: System.Management.ManagementObject#root\cimv2\Win32_LogicalDisk

Name                           MemberType   Definition
----                           ----------   ----------
Chkdsk                         Method       System.Management.ManagementBaseOb...
Reset                          Method       System.Management.ManagementBaseOb...
SetPowerState                  Method       System.Management.ManagementBaseOb...
Access                         Property     System.UInt16 Access {get;set;}
Availability                   Property     System.UInt16 Availability {get;set;}
BlockSize                      Property     System.UInt64 BlockSize {get;set;}
Caption                        Property     System.String Caption {get;set;}
Compressed                     Property     System.Boolean Compressed {get;set;}
ConfigManagerErrorCode         Property     System.UInt32 ConfigManagerErrorCo...
ConfigManagerUserConfig        Property     System.Boolean ConfigManagerUserCo...
CreationClassName              Property     System.String CreationClassName {g...
Description                    Property     System.String Description {get;set;}
DeviceID                       Property     System.String DeviceID {get;set;}
DriveType                      Property     System.UInt32 DriveType {get;set;}
ErrorCleared                   Property     System.Boolean ErrorCleared {get;s...
ErrorDescription               Property     System.String ErrorDescription {ge...
ErrorMethodology               Property     System.String ErrorMethodology {ge...
FileSystem                     Property     System.String FileSystem {get;set;}
FreeSpace                      Property     System.UInt64 FreeSpace {get;set;}
InstallDate                    Property     System.String InstallDate {get;set;}
LastErrorCode                  Property     System.UInt32 LastErrorCode {get;s...
MaximumComponentLength         Property     System.UInt32 MaximumComponentLeng...
MediaType                      Property     System.UInt32 MediaType {get;set;}
Name                           Property     System.String Name {get;set;}
NumberOfBlocks                 Property     System.UInt64 NumberOfBlocks {get;...
PNPDeviceID                    Property     System.String PNPDeviceID {get;set;}
PowerManagementCapabilities    Property     System.UInt16[] PowerManagementCap...
PowerManagementSupported       Property     System.Boolean PowerManagementSupp...
ProviderName                   Property     System.String ProviderName {get;set;}
Purpose                        Property     System.String Purpose {get;set;}
QuotasDisabled                 Property     System.Boolean QuotasDisabled {get...
QuotasIncomplete               Property     System.Boolean QuotasIncomplete {g...
QuotasRebuilding               Property     System.Boolean QuotasRebuilding {g...
Size                           Property     System.UInt64 Size {get;set;}
Status                         Property     System.String Status {get;set;}
StatusInfo                     Property     System.UInt16 StatusInfo {get;set;}
SupportsDiskQuotas             Property     System.Boolean SupportsDiskQuotas ...
SupportsFileBasedCompression   Property     System.Boolean SupportsFileBasedCo...
SystemCreationClassName        Property     System.String SystemCreationClassN...
SystemName                     Property     System.String SystemName {get;set;}
VolumeDirty                    Property     System.Boolean VolumeDirty {get;set;}
VolumeName                     Property     System.String VolumeName {get;set;}
VolumeSerialNumber             Property     System.String VolumeSerialNumber {...
__CLASS                        Property     System.String __CLASS {get;set;}
__DERIVATION                   Property     System.String[] __DERIVATION {get;...
__DYNASTY                      Property     System.String __DYNASTY {get;set;}
__GENUS                        Property     System.Int32 __GENUS {get;set;}
__NAMESPACE                    Property     System.String __NAMESPACE {get;set;}
__PATH                         Property     System.String __PATH {get;set;}
__PROPERTY_COUNT               Property     System.Int32 __PROPERTY_COUNT {get...
__RELPATH                      Property     System.String __RELPATH {get;set;}
__SERVER                       Property     System.String __SERVER {get;set;}
__SUPERCLASS                   Property     System.String __SUPERCLASS {get;set;}
PSStatus                       PropertySet  PSStatus {Status, Availability, De...
ConvertFromDateTime            ScriptMethod System.Object ConvertFromDateTime();
ConvertToDateTime              ScriptMethod System.Object ConvertToDateTime();
Delete                         ScriptMethod System.Object Delete();
GetType                        ScriptMethod System.Object GetType();
Put                            ScriptMethod System.Object Put();


5# _
```

Abbildung 14.9
Auflisten des Pipeline-Inhalts mit Get-Member, wenn sich WMI-Objekte in der Pipeline befinden

14.5 Umgang mit Datumsangaben

ToDateTime()

Datum und Uhrzeit werden in WMI als Zeichenkette der Form `yyyymmddHHMMSS.mmmmmmsUUU` gespeichert, wobei neben dem selbst erklärenden Kürzel anzumerken ist, dass `mmmmmm` die Anzahl der Millisekunden ist und `UUU` die Anzahl der Minuten, welche die lokale Zeit von der Universal Coordinated Time (UTC) abweicht. Das `s` ist das Vorzeichen. In Deutschland steht daher für `UUU` der Wert *+060*.

Zur Konvertierung eines WMI-Datumsformats in das normale Datumsformat der PowerShell (Klasse `System.DateTime`) steht die statische Methode `ToDateTime()` in der Klasse `System.Management.ManagementDateTimeConverter` zur Verfügung.

Listing 14.2 *Umwandlung von WMI-Datumsformaten in System.DateTime [WMI_Date.PS1]*

```
$cs = Get-WMIObject -Class Win32_OperatingSystem
"Startzeit des Systems in WMI-Format: " + $cs.LastBootUpTime
[System.DateTime] $startzeit =
[System.Management.ManagementDateTimeConverter]::ToDateTime
($cs.LastBootUpTime)
"Startzeit des Systems in normalem Format: " + $startzeit
```

Wenn die PowerShell Community Extensions installiert sind, verfügt die Klasse `ManagementObject` über eine zusätzliche Methode `ConvertToDateTime()`, welche die Konvertierung erledigen kann:

```
$cs = Get-WMIObject -Class Win32_OperatingSystem -property
LastBootUpTime
$cs.ConvertToDateTime($cs.LastBootUpTime)
```

14.6 Zugriff auf WMI-Objektmengen

Die Verwendung von `Get-WmiObject` ohne Angabe eines Filters in der Form

```
Get-Wmiobject WMIKlassenname
```

Get-WmiObject

liefert alle Instanzen der angegebenen WMI-Klasse (sofern es die WMI-Klasse auf dem lokalen System gibt).

Beispiel:

```
# Name und Treiberdatei für alle Grafikkarten in diesem Computer
Get-Wmiobject Win32_VideoController
```

liefert alle installierten Grafikkarten.

Dies ist eine Kurzform für

```
Get-WmiObject -class Win32_VideoController
```

Sofern die Klasse nicht im Standardnamensraum "root\cimv2" liegt, muss man den Namensraum mit dem Parameter -Namespace explizit benennen:

```
Get-WmiObject IISwebserver -Namespace root\microsoftIISv2
```

Auch hier kann man mit dem Parameter -Computer auf entfernte Systeme zugreifen:

```
Get-WmiObject -class Win32_VideoController -computer SERVERF112
```

14.7 Filtern und Abfragen

Möglichkeiten

Wenn man nicht alle Instanzen, sondern nur ausgewählte Instanzen ermitteln möchte, die bestimmten Kriterien entsprechen, kann man folgende drei alternative Möglichkeiten nutzen:

- Verwendung eines Filters im Commandlet Get-WmiObject
- Verwendung von WQL-Abfragen mit dem Parameter -Query im Commandlet Get-WmiObject
- Verwendung von WQL-Abfragen mit dem Typ [WMISEARCHER]
- Verwendung von WQL-Abfragen mit der .NET-Klasse System.Management.
- ManagementObjectSearcher

Filtern mit Get-WmiObject

-Filter

Das Commandlet Get-Wmiobject bietet die Möglichkeit, schon beim Abruf die Objekte zu filtern. Der Filter ist nach dem Parameter -Filter in einer Zeichenkette anzugeben.

Beispiele:

- Alle Benutzerkonten aus der Domäne „FBI“:

```
Get-Wmiobject Win32_account -filter "domain='FBI'"
```

- Alle Benutzerkonten aus der Domäne „FBI“, deren Benutzerkontenname mit „H“ beginnt:

```
Get-Wmiobject Win32_account -filter "domain='FBI' and name like 'h%'"
```

WQL-Abfragen

SQL

Abfragen in der Windows Management Instrumentation Query Language (WQL) kann man in der PowerShell mit dem Parameter -Query im Commandlet Get-WmiObject oder mit dem eingebauten Typ [WMISEARCHER] ausführen.

Beispiel

-Query Der folgende Befehl selektiert alle Netzwerkkarten, in denen die Zahlen „802“ im Netzwerkkartentyp vorkommen.

```
Get-WmiObject -query "Select * from Win32_Networkadapter where
adaptertype like '%802%'" | select adaptertype,description
```

[WMISearcher] Alternativ kann man auch diese Abfrage mit dem eingebauten PowerShell-Typ `[WMISEARCHER]` ausführen:

```
([WMISEARCHER] "Select * from Win32_Networkadapter where
adaptertype like '%802%'").get()| select adaptertype,description
```

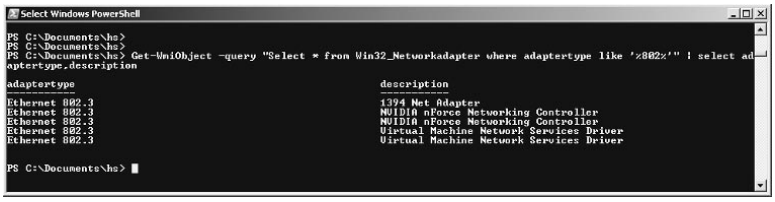


Abbildung 14.10 *Ausführung einer WMI-Abfrage*

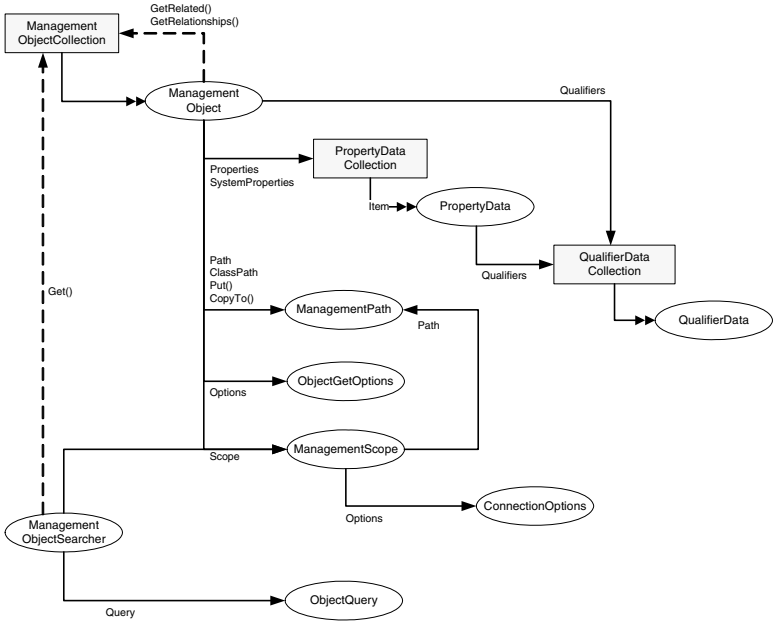


Abbildung 14.11 *Objektmodell für Suchfunktionen über [WMISearcher] bzw. System.Management.ManagementObjectSearcher*

14.8 Zugriff auf Mitglieder von WMI-Klassen

Auf die Attribute und auch auf die Methoden von WMI-Klassen kann man zugreifen wie auf die Mitglieder von .NET-Klassen. Die PowerShell abstrahiert von der Metaobjektmodell-Implementierung in der .NET-Klasse `System.Management.ManagementObject`. Der komplizierte Zugriff auf das Attribut `Properties` und die Methode `Invokemethod()` ist daher nicht notwendig.

Sowohl beim Zugriff auf einzelne Objekte als auch Objektmengen erhält man eine lange Ausgabeliste. Im Standard werden mit `Format-List` die zahlreichen Eigenschaften der ermittelten WMI-Objekte ausgegeben (siehe *Abbildung 14.12* am Beispiel `Win32_Videocontroller`).

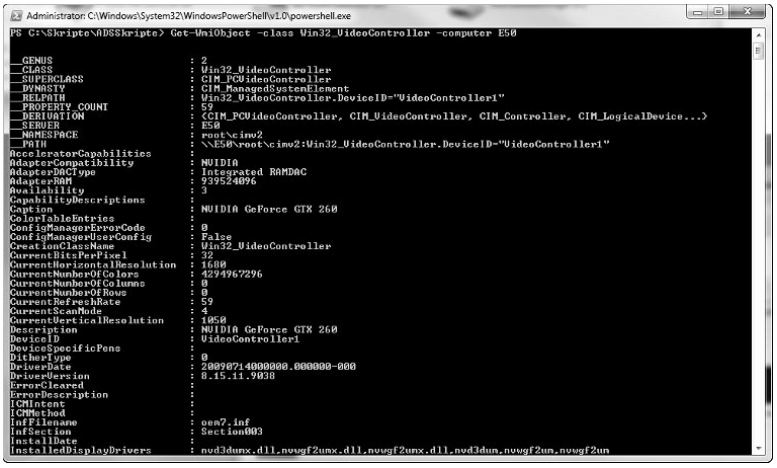


Abbildung 14.12 *Eigenschaften der Klasse Win32_VideoController*

Auch eine Ausgabe mit dem Commandlet `Format-Table` hilft nicht. Dies macht die Ausgabe zwar kürzer, aber viel breiter. Gut wäre es, das resultierende Objekt mit `Select-Object` auf die interessanten Eigenschaften zu „beschneiden“:

```
Get-Wmiobject Win32_VideoController |
Select-Object name,installeddisplaydrive
```

types.ps1xml

Auch für einige WMI-Klassen ist in der *types.ps1xml*-Datei festgelegt, welche Attribute ausgegeben werden. Für `Win32_Videocontroller` gibt es eine solche Festlegung nicht; daher werden alle Attribute ausgegeben. Die folgenden Bildschirmabbildungen zeigen aber die Wirkung der Deklarationen für `Win32_CDRomDrive`.

Abbildung 14.13 *Standardausgabe des Befehls Get-Wmiobject Win32_CDRomDrive*

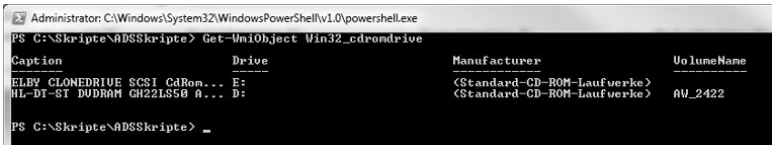


Abbildung 14.14 *Festlegung der auszugebenden Attribute für die WMI-Klasse Win32_CDRom-Drive*

```
<Type>
    <Name>System.Management.ManagementObject#root\cimv2\Win32_CDROMDrive</Name>
    <Members>
        <PropertySet>
            <Name>PSStatus</Name>
            <ReferencedProperties>
                <Name>Availability</Name>
                <Name>Drive</Name>
                <Name>ErrorCleared</Name>
                <Name>MediaLoaded</Name>
                <Name>NeedsCleaning</Name>
                <Name>Status</Name>
                <Name>StatusInfo</Name>
            </ReferencedProperties>
        </PropertySet>
        <MemberSet>
            <Name>PSStandardMembers</Name>
            <Members>
                <PropertySet>
                    <Name>DefaultDisplayPropertySet</Name>
                    <ReferencedProperties>
                        <Name>Caption</Name>
                        <Name>Drive</Name>
                        <Name>Manufacturer</Name>
                        <Name>VolumeName</Name>
                    </ReferencedProperties>
                </PropertySet>
            </Members>
        </MemberSet>
    </Members>
</Type>
```

Das folgende Listing zeigt weitere Beispiele zum Einsatz von `Get-WmiObject` in Zusammenarbeit mit Commandlets zur Pipeline-Steuerung:

```
# Name und freie Bytes auf allen Laufwerken
Get-Wmiobject Win32_logicaldisk | Select-Object deviceid,freespace

# Name und Domain der Benutzerkonten, deren Kennwort niemals
verfällt
Get-Wmiobject Win32_account | Where-Object {$_.Kennwortexpires -eq
0 } | Select-Object Name,Domain
```



Statische Klassenmitglieder

Anders als bei .NET-Objekten macht die PowerShell bei WMI keine syntaktischen Unterschiede zwischen statischen Methoden und Instanzmethoden, d.h., es ist immer nur der einfache Punkt-Operator zu verwenden (in .NET-Objekten ist der doppelte Doppelpunkt für statische Methoden zu verwenden). Bei WMI ist nur zu beachten, dass mit dem PowerShell-Typ `[WMIClass]` auf den WMI-Pfad der WMI-Klasse, nicht einer konkreten Instanz verwiesen wird.

Beispiel:

```
[WMIClass] "Win32_Product").Install("c:\name.msi")
```

14.9 Werte setzen mit Set-WmiInstance

Set-WmiInstance

Mit dem Commandlet `Set-WmiInstance` kann man ab PowerShell 2.0 Werte in WMI-Objekten setzen:

Beispiel: Ändern des Namens des Laufwerks C auf dem Computer „F171“:

```
Set-WmiInstance -path "\\F171\root\cimv2:Win32_
LogicalDisk.DeviceID='c:'" -Arguments @{ VolumeName="System" }
```

14.10 Methodenaufrufe mit Invoke-WmiMethod

Invoke-WmiMethod

Neu ab PowerShell 2.0 ist das Commandlet `Invoke-WmiMethod`, mit dem man WMI-Methoden direkt aufrufen kann.

Chkdsk()

Beispiel: Aufruf der Methode `Chkdsk()` in der WMI-Klasse `Win32_LogicalDisk` mit sechs Parametern (auf Computer F171):

```
Invoke-WmiMethod -Path "\\F171\root\cimv2:Win32_
LogicalDisk.DeviceID='c:'" -Name "Chkdsk" -ArgumentList
false,false,false,false,false
```

14.11 Instanzen entfernen

Invoke-WmiMethod

Mit `Remove-WmiObject` (ab PowerShell 2.0) kann man ein WMI-Objekt löschen, z.B. einen Dateisystemordner auf einem entfernten System:

```
Remove-WmiObject -path "\\F171\root\cimv2:Win32_Directory.Name='c:\
\temp'"
```

Diesen obigen Befehl könnte man alternativ ausdrücken ohne Einsatz von WMI:

```
Invoke-Command -ComputerName F171 { Remove-Item "c:\temp" }
```

Der Unterschied ist: Der erste Befehl verwendet nur WMI-Klassen, die es auf jedem Windows-System (zumindest seit Windows 2000) gibt. Der zweite Befehl erfordert auf dem Zielsystem die PowerShell 2.0 und dort aktiviertes PowerShell-Remoting.

14.12 Ermittlung der Mitglieder des WMI-Objekts

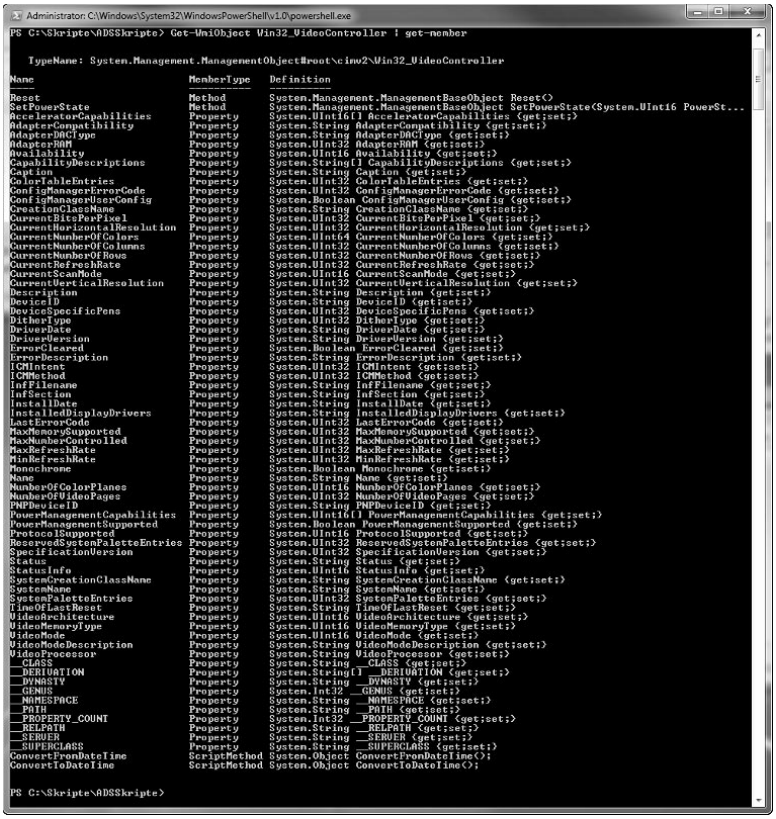
Get-Member

Die Menge der verfügbaren Attribute und Methoden ermittelt man wie bei .NET-Objekten mit `Get-Member`. Obwohl die Mitglieder einer WMI-Klasse (z.B. `Win32_Videocontroller`) nicht auch gleichzeitig Mitglieder der die WMI-Klasse verpackenden .NET-Metaklasse (`System.Management.ManagementObject`) sind, listet `Get-Member` dennoch die Mitglieder aus beiden Abstraktionsebenen auf.

Für die Darstellung des Typnamens hat die PowerShell eine eigene Notation, bei der die Metaklasse durch die Raute (#) von der WMI-Klasse getrennt wird, z.B.:

```
System.Management.ManagementObject#root\cimv2\Win32_VideoController
```

Abbildung 14.15
Ermittlung der Mitglieder einer WMI-Klasse



14.13 Liste aller WMI-Klassen

Zugriff auf das WMI-Repository

Eine Liste aller verfügbaren WMI-Klassen auf einem System erhält man mit dem Parameter `-List` im Commandlet `Get-WmiObject`. Eine Klasse darf dabei nicht angegeben werden.

```
Get-Wmiobject -list
```

Wenn nichts angegeben wird, wird immer der Namensraum "`root\cimv2`" verwendet. Man kann einen Namensraum auch explizit angeben:

```
Get-Wmiobject -list -Namespace root/cimv2/Anwendungs/microsoftIE
```

Man kann auch auf das WMI-Repository eines bestimmten Computers zugreifen, da die Menge der Klassen vom Betriebssystem und von den installierten Anwendungen abhängig ist.

```
Get-Wmiobject -list -Computer SERVERF112
```

14.14 Instanziieren von WMI-Klassen

Viele WMI-Klassen sind so gestaltet, dass zum Anlegen neuer Systemelemente eine Instanz der Klasse erzeugt wird. Dafür werden auf Klassenebene statische Methoden angeboten mit Namen `Create()`, vgl. *Abbildung 14.16*.

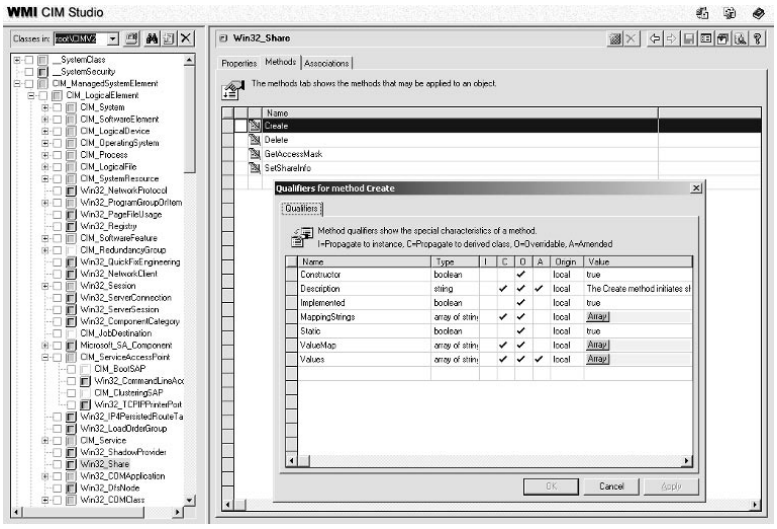


Abbildung 14.16 *Methoden der Klasse Win32_Share*

Beispiel

Das folgende Beispiel zeigt das Anlegen einer Freigabe mit Standardrechten. Die Vergabe von Rechten ist ein komplexeres Thema, das später noch behandelt wird.

Listing 14.3 *New-Share-Without-Permissions.ps1*

```
# Create Win32_Share

$pfad = "c: \Temp"
$Sharename = "Freigabe"
$Comment = "Freigabe von c: \Temp"
$class = [WMIClass] "ROOT\CIMV2:Win32_Share"
$Access = $Null
$R = $class.Create($pfad, $Sharename, 0, 10, $Comment, "", $Access)
if ( $R.ReturnValue -ne 0) { Write-Error "Fehler beim Anlegen: "+
$R.ReturnValue; Exit}
"Freigabe wurde angelegt!"
```

15 Dynamische Objekte

In den meisten Programmiersprachen ist der Aufbau von Objekten durch Klassendefinitionen zur Entwicklungszeit festgelegt. Die Klasse definiert die Mitglieder (Attribute, Methoden und Ereignisse) und die Instanzen dieser Klasse erhalten genau die in der Klasse definierten Mitglieder. Einige Programmiersprachen erlauben die Erweiterungen von Objekten zur Laufzeit um neue Mitglieder. Die PowerShell gehört zu diesen Programmierumgebungen, die diese Möglichkeit besitzen.

Add-Member

Zentrales Instrument ist dafür das Commandlet `Add-Member`, mit dem man ein bestehendes Objekt um Mitglieder erweitern kann. Ein zusätzliches Attribut sind ein „NoteProperty", eine zusätzliche Methode eine „ScriptMethod" (weil man sie in der PowerShell-Skriptsprache implementiert).

Add-Member erwartet folgende Angaben:

- `-InputObject`: das zu erweiternde Objekt
- `-MemberType`: NoteProperty oder ScriptMethode
- `-Name`: Bezeichner des neuen Mitglieds
- `-Value`: bei einem NoteProperty der Wert, bei einer ScriptMethod der Skriptblock

15.1 Erweitern bestehender Objekte

Größe von Dateisystemverzeichnissen

Eine Schwäche in der .NET-Klassenbibliothek ist sicherlich, dass die Klasse `System.IO.DirectoryInfo`, die Ordner im Dateisystem repräsentiert, kein Attribut besitzt, das die Größe aller Dateien in diesem Ordner wiedergibt. Es gibt lediglich für einzelne Dateien (`System.IO.FileInfo`) das Attribut `Length`. Die nachstehende dokumentierte Funktion `Get-DirSize` ermittelt ausgehend von einem beliebigen Ordner die Größe der Unterordner, in dem rekursiv die Größe aller Dateien ermittelt wird. Die ermittelte Größe könnte man in einer separaten Datenstruktur halten. Es ist aber viel eleganter, diese Größe an die bestehenden DirectoryInfo-Objekte für die Ordner anzuheften.

Listing 15.1 *Ermitteln der Ordnergrößen und Anheften der Informationen an die Klasse DirectoryInfo*

```
# DirSize.ps1
# Erweitern der Klasse DirectoryInfo um Attribut Size

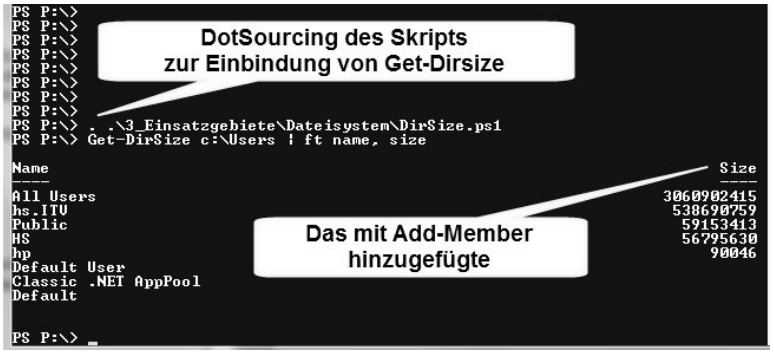
function Get-DirSize($Path)
{
# Zugriff auf Unterverzeichnisse des Wurzelverzeichnises
$WurzelVerzeichnisse = Get-ChildItem $Path -Force  | where { $_
.GetType().Name -eq "DirectoryInfo" }

# Iteration über die Unterordner
foreach ($AktuellesVerzeichnis in $WurzelVerzeichnisse)
{
# Ermitteln der Größe aller Dateien in dem Ordner und seinen
Unterordnern
$groesse = ($AktuellesVerzeichnis | Get-ChildItem -Recurse -ea
silentlycontinue  | where { $_.GetType().Name -ne "DirectoryInfo" }
|  Measure-Object length -Sum)
# Anfügen des Mitglieds "Size" an das DirectoryInfo-Objekt
Add-Member -InputObject $AktuellesVerzeichnis -Value $groesse.Sum -
Name "Size" -MemberType "NoteProperty"
}
# Sortierte Ausgabe der Größen
$WurzelVerzeichnisse | sort Size -desc
}
```

Diese Funktion kann man beispielsweise benutzen, um die Größe der Benutzerprofile auf einem System zu ermitteln.

```
Get-DirSize c:\users | ft Name, Size
```

Abbildung 15.1 *Ausgabe der Ordnergrößen von c:\users*



```
PS P:\> Get-DirSize c:\Users | gm

   TypeName: System.IO.DirectoryInfo

Name                       MemberType     Definition
----                       ----------     ----------
Mode                       CodeProperty   System.String Mode{get=Mode;}
Create                     Method         System.Void Create(System.Security....
CreateObjRef               Method         System.Runtime.Remoting.ObjRef Crea...
CreateSubdirectory         Method         System.IO.DirectoryInfo CreateSubdi...
Delete                     Method         System.Void Delete(), System.Void D...
Equals                     Method         bool Equals(System.Object obj)
GetAccessControl           Method         System.Security.AccessControl.Direc...
GetDirectories             Method         System.IO.DirectoryInfo[] GetDirect...
GetFiles                   Method         System.IO.FileInfo[] GetFiles(), Sy...
GetFileSystemInfos         Method         System.IO.FileSystemInfo[] GetFileS...
GetHashCode                Method         int GetHashCode()
GetLifetimeService         Method         System.Object GetLifetimeService()
GetObjectData              Method         System.Void GetObjectData(System.Ru...
GetType                    Method         type GetType()
InitializeLifetimeService  Method         System.Object InitializeLifetimeSer...
MoveTo                     Method         System.Void MoveTo(string destDirName)
Refresh                    Method         System.Void Refresh()
SetAccessControl           Method         System.Void SetAccessControl(System....
ToString                   Method         string ToString()
PSChildName                NoteProperty   System.String PSChildName=All Users
PSDrive                    NoteProperty   System.Management.Automation.PSDriv...
PSIsContainer              NoteProperty   System.Boolean PSIsContainer=True
PSParentPath               NoteProperty   System.String PSParentPath=Microsof...
PSPath                     NoteProperty   System.String PSPath=Microsoft.Powe...
PSProvider                 NoteProperty   System.Management.Automation.Provid...
Size                       NoteProperty   System.Double Size=3060902415
Attributes                 Property       System.IO.FileAttributes Attributes...
CreationTime               Property       System.DateTime CreationTime {get;s...
CreationTimeUtc            Property       System.DateTime CreationTimeUtc {ge...
Exists                     Property       System.Boolean Exists {get;}
Extension                  Property       System.String Extension {get;}
FullName                   Property       System.String FullName {get;}
LastAccessTime             Property       System.DateTime LastAccessTime {get...
LastAccessTimeUtc          Property       System.DateTime LastAccessTimeUtc {...
LastWriteTime              Property       System.DateTime LastWriteTime {get;...
LastWriteTimeUtc           Property       System.DateTime LastWriteTimeUtc {g...
Name                       Property       System.String Name {get;}
Parent                     Property       System.IO.DirectoryInfo Parent {get;}
Root                       Property       System.IO.DirectoryInfo Root {get;}
BaseName                   ScriptProperty System.Object BaseName {get=$this.N...

PS P:\>
```

Abbildung 15.2 *Der Einsatz von Get-Member beweist, das System.IO.DirectoryInfo tatsächlich erweitert wurde.*

Bitte beachten Sie, dass der Standort der Profilordner vom Betriebssystem abhängig ist.



15.2 Komplett dynamische Objekte

Man kann auch eigene Objekte komplett neu aufbauen. Immer notwendige Basis ist dabei lediglich die Klasse `PSCustomObject`. Die Klasse erbt direkt von `System.Object`. Von `System.Object` erhält `PSCustomObject` die vier öffentlichen Mitglieder, die jedes .NET-Objekt besitzt.

```
   TypeName: System.Management.Automation.PSCustomObject

Name        MemberType Definition
----        ---------- ----------
Equals      Method     bool Equals(System.Object obj)
GetHashCode Method     int GetHashCode()
GetType     Method     type GetType()
ToString    Method     string ToString()
```

Abbildung 15.3 *Die vier Standardmitglieder eines jeden .NET-Objekts*

Beispiel

Person-Objekt

Das Beispiel zeigt die Erstellung eines Person-Objekts, das die Attribute `Geschlecht`, `Titel`, `Vorname` und `Name` besitzt, sowie eine Script-Method `GetAnrede()`, die diese beiden Zeichenketten korrekt zu einer Anrede zusammenfügt.

Listing 15.2 *Dynamischer-Objekte.ps1*

```
[PSCustomObject] $person = new-object PSCustomObject

Add-Member -InputObject $person -MemberType NoteProperty -Name
Geschlecht "M"
Add-Member -InputObject $person -MemberType NoteProperty -Name Name
"Schwichtenberg"
Add-Member -InputObject $person -MemberType NoteProperty -Name
Vorname "Holger"
Add-Member -InputObject $person -MemberType NoteProperty -Name
Titel "Dr."
Add-Member -InputObject $person -MemberType ScriptMethod -Name
GetAnrede -Value {

if ($Geschlecht -eq "W") { $Anrede = "Frau"}
else { $Anrede = "Herr" }
if ($Titel -ne "") { $Anrede = $Anrede + " " + $this.Titel}
return $Anrede + " " + $this.Vorname + " " + $this.Name

}

$person.GetAnrede() # liefert: "Herr Dr. Holger Schwichtenberg"
```



Innerhalb einer ScriptMethod nimmt man mit `$this` Bezug auf die übrigen Mitglieder des aktuellen Objekts. Dies ist wohl zu unterscheiden von `$_`, mit dem man das aktuelle Objekt in der Pipeline adressiert.

Natürlich kann man innerhalb einer ScriptMethod auch Pipeline-Befehle verwenden.

16 Einbinden von C# und VB.NET

Add-Type

Das ab PowerShell 2.0 neu eingeführte Commandlet `Add-Type` erlaubt die dynamische Kompilierung einer .NET-Klasse. Dabei kann der Quelltext in einer externen Datei liegen oder in ein PowerShell-Skript eingebettet sein. Erlaubte Programmiersprachen sind C# („cs"), Visual Basic .NET („VisualBasic") und JScript .NET („js").

In PowerShell 1.0 konnte man Programmcode aus einer der drei vorgenannten Sprachen nur verwenden, indem man den Programmcode statisch kompiliert und dann die entstandene Assembly in die PowerShell geladen hat. Mit `Add-Type` kann man nun mit den .NET-Sprachen umgehen wie mit Skriptsprachen.

Das folgende Listing zeigt am Beispiel einer „HelloWorld"-Anwendung verschiedene Möglichkeiten, C#-Programmcode aufzurufen.

Listing 16.1
Hello World in C#, eingebettet in ein PowerShell-Skript [WPS2_CSClass.ps1]

```
# CSharp-Code eingebettet in PowerShell-Skripte
$CSKlasseCode = @"
   // Statische Methode mit Parameter
   public static string Hallo1(string name)
   {
   return "Hallo " + name + "!";
   }

   // Instanz-Methode mit Parameter
       public string Hallo2(string name)
   {
   return "Hallo " + name + "!";
   }

   public string Name;

   // Instanz-Methode ohne Parameter
   public string Hallo3()
   {
   return "Hallo " + this.Name + "!";
   }

"@

# Dynamisches Kompilieren
$CSKlasse = Add-Type -MemberDefinition $CSKlasseCode -Name "Welt" -
language csharp -UsingNamespace
```

```
"System.Reflection","System.Diagnostics" -PassThru

# Testen
$CSKlasse::Hallo1("Holger")
$o = New-Object $CSKlasse
$o.Hallo2("Holger")
$o.Name = "Holger"
$o.Hallo3()
```

Listing 16.2 *Hello World in Visual Basic .NET, eingebettet in ein PowerShell-Skript [WPS2_VBClass.ps1*

```
$MeineKlasse= @"

' Statische Methode mit Parameter
Public Shared Function Hallo1(ByVal name As String) As String
    Return "Hallo " & name & "!"
End Function

' Instanz-Methode mit Parameter
Public Function Hallo2(ByVal name As String) As String
    Return "Hallo " & name & "!"
End Function

Public _Name As String

' Instanz-Methode ohne Parameter
Public Function Hallo3() As String
    Return "Hallo " & Me._Name & "!"
End Function
"@


$myclass = Add-Type -MemberDefinition $MeineKlasse-Name
"MeineKlasse" -language VisualBasic -UsingNamespace
"System.Reflection","System.Diagnostics" -PassThru

$myclass::Hallo1("Holger")

$o = New-Object $myclass
$o.Hallo2("Holger")

$o.Name = "Holger"
$o.Hallo3()
```

17 Win32-API-Aufrufe

Plattform Invoke

Man kann aus der PowerShell heraus direkt auf .NET-Klassen, COM-Klassen und WMI-Klassen zugreifen. Es gibt aber keine unmittelbare Zugriffsmöglichkeit auf die Win32-API-Funktionen. Auch schon zu Zeiten von PowerShell 1.0 gab es einen Trick mit der FCL, um Win32-API-Aufrufe über den in das .NET Framework eingebauten Mechanismus „Plattform Invoke“ (kurz: PInvoke) aufzurufen. In PowerShell 2.0 ist dies stark vereinfacht durch die Möglichkeit, C#- oder Visual Basic.NET-Code direkt in PowerShell-Skriptcode einzubetten. So kann man dort PInvoke in die Win32-API-Aufrufe kapseln.

Das Beispiel zeigt eine in ein PowerShell-Skript eingebettete C#-Klasse, die Zugriff auf die Bildschirmschonereinstellungen über die Win32-API-Methode `SystemParametersInfo` bietet.

Listing 17.1
Die in das PowerShell-Skript eingebettete Klasse „ScreenSaver“ kapselt den Zugriff auf eine Win32-API-Funktion. [WPS2_Win32API._Screensaver.ps1]

```
$CSharpKlassenDefinition = @"

        [DllImport("user32.dll", CharSet = CharSet.Auto)]
        private static extern bool SystemParametersInfo(
           int uAction, int uParam, ref bool lpvParam,
           int flags);

private const int SPI_GETSCREENSAVERACTIVE = 16;
private const int SPI_SETSCREENSAVERACTIVE = 17;
private const int SPIF_SENDWININICHANGE = 2;
private const int SPI_GETSCREENSAVERTIMEOUT = 14;
        private const int SPI_SETSCREENSAVERTIMEOUT = 15;

 public static bool GetStatus()
        {
            bool isActive = false;

            SystemParametersInfo(SPI_GETSCREENSAVERACTIVE, 0,
               ref isActive, 0);

            return isActive;
        }


        public static void TurnOn()
        {
            int nullVar = 0;
```

```
            bool result = SystemParametersInfo(SPI_
SETSCREENSAVERACTIVE,
               1, ref nullVar, SPIF_SENDWININICHANGE);

            if (!result)
            {
                throw new
System.ComponentModel.Win32Exception(Marshal.GetLastWin32Error());
            }
        }


        public static void TurnOff()
        {
            int nullVar = 0;

            bool result = SystemParametersInfo(SPI_
SETSCREENSAVERACTIVE,
               0, ref nullVar, SPIF_SENDWININICHANGE);

            if (!result)
            {
                throw new
System.ComponentModel.Win32Exception(Marshal.GetLastWin32Error());
            }
        }

 public static Int32 GetTimeout()
        {
            Int32 value = 0;

            bool res = SystemParametersInfo(SPI_
GETSCREENSAVERTIMEOUT, 0,
               ref value, 0);
            if (!res)
            {
                throw new
System.ComponentModel.Win32Exception(Marshal.GetLastWin32Error());
            }
            return value;
        }


        public static void SetTimeout(Int32 Value)
        {
            int nullVar = 0;

            bool res = SystemParametersInfo(SPI_
```

```
SETSCREENSAVERTIMEOUT,
               Value, ref nullVar, SPIF_SENDWININICHANGE);
            if (!res)
            {
                throw new
System.ComponentModel.Win32Exception(Marshal.GetLastWin32Error());
            }

        }
   "@

$Klasse = Add-Type -MemberDefinition $CSharpKlassenDefinition -Name
"ScreenSaver" -UsingNamespace
"System.Reflection","System.Diagnostics" -PassThru
$Klasse::GetStatus()
$Klasse::GetTimeout()
$Klasse::TurnOff()
$Klasse::GetStatus()
$Klasse::GetStatus()
$Klasse::SetTimeout(800)
$Klasse::GetStatus()
```

18 Fernausführung (Remoting)

Eine der schmerzlich vermissten Funktionen in der Windows PowerShell 1.0 war die generelle Unterstützung für Fernzugriffe auf andere Systeme bzw. Fernverwaltung von entfernten Systemen. Mit den Bordmitteln der PowerShell 1.0 konnte man im Wesentlichen nur über WMI via Distributed COM (DCOM) Daten von anderen Systemen abrufen. Eine generelle Möglichkeit zur Fernausführung von Commandlets und Skripten gab es nicht. Hier hat Microsoft nun in der Version 2.0 deutlich nachgebessert.

In PowerShell 2.0 gibt es über das Protokoll WS-Management (kurz: WS-Man) die Fernausführungsmöglichkeit („PowerShell Remoting") für einzelne Commandlets und ganze Skripte.

Webservice

WS-Management ist ein Netzwerkprotokoll auf Basis von XML-Webservices unter Verwendung des Simple Object Access Protocol (SOAP). WS-Management dient dem Austausch von Verwaltungsinformationen zwischen (heterogenen) Computersystemen. WS-Management ist ein Standard der Desktop Management Task Force (DMTF), der im Jahr 2006 verabschiedet wurde.

WS-Management bietet eine enge Verbindung zum Web Based Enterprise Management (WBEM) alias Windows Management Instrumentation (WMI).

Microsoft bietet eine Implementierung von WS-Management unter dem Namen „Windows Remote Management (WinRM)" für Windows XP, Windows Server 2003, Vista, Windows 7 und Windows Server 2008.

In Vista und Windows Server 2008 ist die Version 1.1 der Implementierung enthalten. Windows 7 und Windows Server 2008 R2 enthalten Version 2.0. Für Windows XP, Windows Vista sowie Windows Server 2003 (inkl. R2) und Windows Server 2008 gibt es ein Add-On für WinRM v2 mit Namen „Windows Management Framework".

WinRM

Für einen Fernaufruf müssen sowohl der lokale (der Aufrufer, der Client) als auch der entfernte Computer (der Aufgerufene, der Server) Windows Remote Management (WinRM) 2.0 unterstützen. Außerdem muss die PowerShell 2.0 auf beiden Systemen installiert sein. WinRM benutzt (seit Version 2.0) im Standard die Ports 5985 (HTTP) und 5986 (HTTPS). Die Authentifizierung erfolgt im Normalfall über Kerberos, alternativ sind auch Basisauthentifizierung, Digest und NTLM möglich.

Die Verbindung zwischen Client und Server kann permanent oder temporär sein.

In PowerShell 2.0 ist nicht nur ein Fernaufruf eines Computers, sondern auch gleichzeitig mehrerer Computer möglich. So kann man z.B. ein Skript gleichzeitig auf mehreren entfernten Systemen starten.

18.1 Fernabfrage ohne WS-Management

RPC

Einige Commandlets in der PowerShell 2.0 besitzen eingebaute Fernabfragemöglichkeiten abseits von WS-Management. Diese Commandlets haben einen Parameter „-Computername“ und die Fernaufrufmöglichkeiten des Betriebssystems, die auf einem Remote Procedure Call (RPC) basieren.

Commandlets mit "-Computername"

Folgende Commandlets besitzen den Parameter „-Computername“:

- `Clear-EventLog`
- `Limit-EventLog`
- `Get-Counter`
- `New-EventLog`
- `Get-EventLog`
- `Remove-EventLog`
- `Get-HotFix`
- `Restart-Computer`
- `Get-Process`
- `Show-EventLog`
- `Get-Service`
- `Show-Service`
- `Get-WinEvent`
- `Stop-Computer`
- `Get-WmiObject` (schon in PowerShell 1.0 vorhanden)
- `Write-EventLog`

Der Fernaufruf mit vorgenannten Commandlets funktioniert auch, wenn WS-Management nicht installiert und konfiguriert ist.

Derartige Commandlets findet man mit:

```
Get-Command | where { $_.parameters.keys -contains
"ComputerName" -and $_.parameters.keys -notcontains "Session"}
```



Der folgende Befehl ermittelt vom Computer „F170“ alle Dienste, die mit dem Buchstaben „i“ beginnen. **Beispiel**

```
Get-Service -ComputerName F170 i*
```

Die Abfrage mehrerer Computer ist nur nacheinander durch Übergabe in der Pipeline möglich, da man bei diesen Commandlets bei Computername kein Array als solchen übergeben kann.

Falsch: **Falsch**

```
Get-Service -ComputerName F173, F170 i*
```

Richtig: **Richtig**

```
"F171", "F172", "F173" | % { Get-Service i* -ComputerName $_ }| ft
Name, status, machinename
```

Über das Attribut `MachineName` kann man jeweils sehen, welcher der abgefragten Computer das Ergebnis geliefert hat.



Abbildung 18.1
Abfrage der Dienste auf zwei Computern

18.2 Anforderungen

Für einen Fernaufruf müssen sowohl der lokale (der Aufrufer, der Client) als auch der entfernte Computer (der Aufgerufene, der Server) folgende Voraussetzungen erfüllen:

- Microsoft .NET Framework 2.0 oder höher
- Windows PowerShell 2.0 oder höher
- Windows Remote Management (WinRM) 2.0

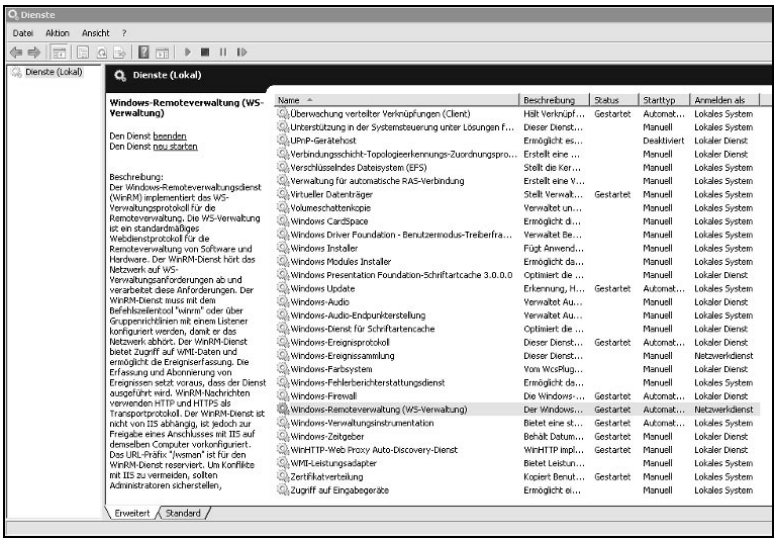


Abbildung 18.2 *Der WinRM-Dienst in einem deutschen Windows 7*

18.3 Rechte für Fernaufrufe

Fernaufrufe sind sowohl domänenintern als auch domänenübergreifend (durch Vertrauensstellungen oder gleiche Benutzername-Kennwort-Kombination) möglich, im zweiten Fall allerdings nur unter expliziter Angabe von Benutzernamen und Kennwort, selbst wenn auf dem Zielsystem eine zu dem aufrufenden System identische Benutzername-Kennwort-Kombination existiert. Fernaufrufe können nur Benutzer ausführen, die auf dem Zielsystem zur Administratorengruppe gehören. Man kann das Recht zum Fernaufruf durch Änderung der sogenannten Sitzungskonfigurationen steuern.

Auch auf dem lokalen Computer werden Administratorrechte für einige Aktionen im Zusammenhang mit dem Fernaufruf benötigt. Dies sind insbesondere die Einrichtung von WS-Management und die Konfiguration von PowerShell-Sitzungen. Auch ein Fernaufruf gegen den eigenen Computer („Loopback-Aufruf“) erfordert Administratorrechte.

18.4 Einrichten von WinRM

EnablePS-Remoting

Im Auslieferungszustand der PowerShell sind PowerShell-Fernaufrufe deaktiviert. Mit `Enable-PSRemoting` konfiguriert man einen Computer zum Empfang von Fernaufrufen von anderen Rechnern. Dieser Befehl startet den WinRM-Systemdienst, konfiguriert die Windows PowerShell und trägt das Protokoll WS-Management als Ausnahme in der

Windows Firewall ein. `Enable-PSRemoting` ist nicht notwendig auf Computern, die nur (!) PowerShell-Befehle an andere Rechner senden wollen. Zum Ausführen von `Enable-PSRemoting` muss man auf dem System Administrator sein.

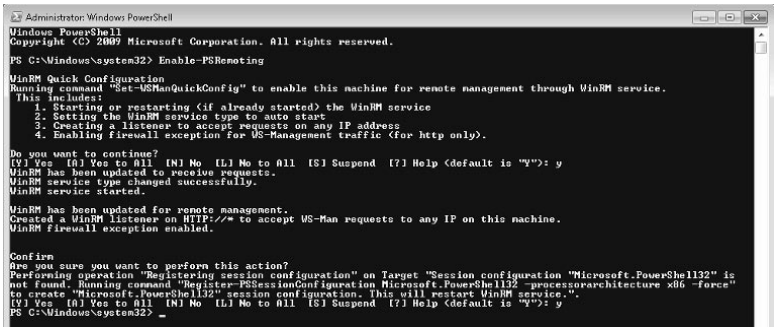


Abbildung 18.3
Erstkonfiguration von PowerShell Remoting



Zum Unterdrücken der Nachfragen geben Sie ein:

```
Enable-Psremoting -force
```

Zum Testen der Einrichtung geben Sie ein:

```
New-PSSession
```

Dann sollte das nachstehende Ergebnis erscheinen.

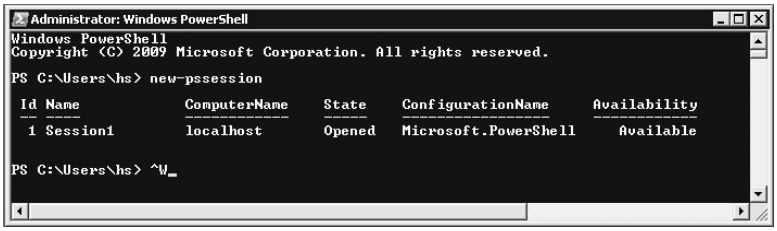


Abbildung 18.4
Testen der Fernverbindungskonfiguration mit New-PSSession



In Domänen kann WinRM über die Gruppenrichtlinie „Computer Configuration\Administrative Templates\Windows Components\Windows Remote Management (WinRM)\WinRM service“ konfiguriert werden.

Zum Deaktivieren des PowerShell Remoting gibt es das Commandlet:

```
Disable-PSRemoting
```

18.5 Überblick über die Commandlets

Die wichtigsten Commandlets für die Fernausführung sind:

- `Enter-PSSession`: Starten einer Fernausführungssitzung im Telnet-Stil
- `Invoke-Command`: Fernausführung eines einzelnen PowerShell-Commandlets oder eines Skripts
- `New-PSSession`: Erstellen einer permanenten Verbindung für die Fernausführung

Alle oben genannten Commandlets bieten ein Attribut `-Computername`. Bei `Enter-PSSession` kann man nur einen Computer angeben, bei `Invoke-Command` und `New-PSSession` auch ein Array mehrerer Computer.

Wichtiger Hinweis: Im Standard vorgesehen ist die Angabe von Computernamen. IP-Adressen können alternativ verwendet werden, erfordern aber ein anderes Authentifizierungsverfahren, da das im Standard verwendete Kerberos keine IP-Adressen unterstützt (siehe Kapitel zur Sicherheit bei PowerShell Remoting).

18.6 Interaktive Fernverbindungen im Telnet-Stil

Enter-PSSession

Mit dem Commandlet `Enter-PSSession` eröffnet man eine interaktive Sitzung zu einem entfernten System im Stil des Telnet-Protokolls. Anzugeben ist der Computername, z.B.

```
Enter-PSSession -Computername F170
```

Nach erfolgreicher Ausführung des Befehls wird der Computername vor der PowerShell-Eingabeaufforderung angezeigt. Alle eingegebenen Befehle werden nun auf dem entfernten System ausgeführt. Alle Ausgaben landen auf dem lokalen System.

Testen kann man zum Beispiel, indem man mit `[System.Environment]::MachineName` den Computernamen abruft.

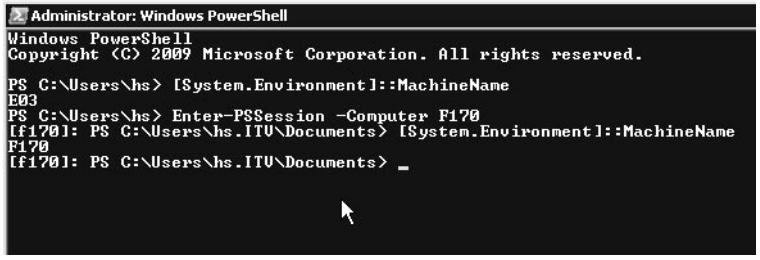


Abbildung 18.5 *Aufbau einer interaktiven Fernsitzung*



Man kann sich in einer interaktiven Sitzung immer nur mit genau einem entfernten System verbinden. Man kann aber auf einem System mehrere PowerShell-Fenster öffnen und sich darin mit jeweils einem anderen entfernten System verbinden.

Möglichkeiten einer Fernsitzung

In einer Fernsitzung kann jegliche Form von Änderungen durchgeführt werden, sowohl durch Ausführung von Commandlets, z.B.

```
(Get-service BITS) | start-service
```

als auch durch den Aufruf von Methoden

```
(Get-service BITS).Start()
```

Man kann auch Windows-Prozesse starten. Zu beachten ist jedoch, dass man auf dem entfernten System Benutzeroberflächen von diesen Prozessen nicht sehen kann, selbst wenn man dort lokal angemeldet ist.

Exit-PSSession

Zum Verlassen der Fernsitzung gibt man ein:

```
Exit-PSSession
```

18.7 Fernausführung von Befehlen

InvokeCommand

Um einen einzelnen Befehl auf einem entfernten System auszuführen, kann man auch das Commandlet `Invoke-Command` mit dem Parameter `-Computername` verwenden. Beim Parameter `-ScriptBlock` kann man einen oder mehrere (durch Zeilenumbruch oder Semikolon getrennte) Befehle angeben und auch Pipelines nutzen. Nicht nur Commandlets, auch klassische Kommandozeilenbefehle sind möglich.

Beispiel 1

```
Invoke-Command -ComputerName F170 -scriptblock { Get-Service b* }
```

Beispiel 2

```
Invoke-Command -ComputerName F170 -scriptblock { Get-Service | sort
status | ft name, status }
```

Beispiel 3

```
Invoke-Command -computer F170 -Script { "Computername: " +
[System.Environment]::MachineName ; "Zeit: " + [DateTime]::Now ;
"Sprache: " +
(Get-Culture) }
```

Beispiel 4

```
Invoke-Command -computer F170 -Script { ping www.it-visions.de }
```



Alle Commandlets oder Anwendungen, die in dem Skriptblock gestartet werden, müssen auf dem Zielsystem verfügbar sein.

Auch hier wird das Ergebnis auf dem lokalen System angezeigt, oft mit der zusätzlichen Spalte „PSComputerName“, die den Namen des aufgerufenen Computers enthält.

Abbildung 18.6
Lokaler Aufruf versus entfernter Aufruf



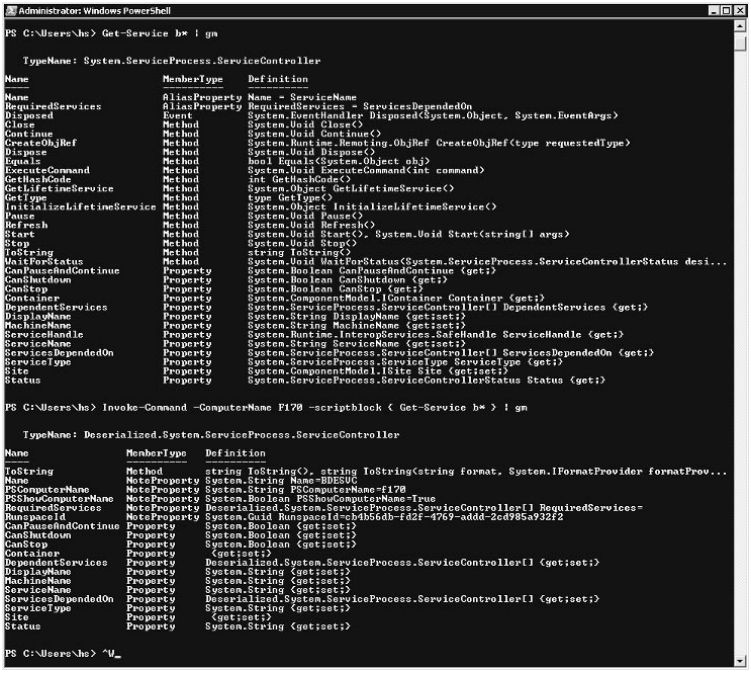
Serialisierung/ Deserialisierung

Allerdings muss man beachten, dass die Ergebnismenge keineswegs die gleiche Struktur wie bei einem lokalen Aufruf hat. Die Objekte in der Pipeline sind nicht vom Typ `System.ServiceProcess.ServiceController`, sondern `Deserialized.System.ServiceProcess.ServiceController`. Zwischen den Rechnergrenzen hat, für den Transport über das Netzwerk, eine Serialisierung/Deserialisierung der Objekte stattgefunden. Dabei sind die Methoden der Objekte „verloren“ gegangen. Methodenaufrufe wie

```
(Invoke-Command -ComputerName F170 -scriptblock { Get-Service bits
}).Start()
```

sind also nicht möglich!

Abbildung 18.7
Get-Member nach einem lokalen und einem entfernten Aufruf von Get-Service





Beim Pipelining kann man in eine sehr tiefe Falle tappen. Ein Benutzer, der merkt, dass

```
(Invoke-Command -ComputerName F170 -scriptblock { Get-Service
Bits }).Start()
```

nicht funktioniert, würde wohl auf

```
(Invoke-Command -ComputerName F170 -scriptblock { Get-Service
Bits }) | Start-Service
```

ausweichen wollen.

Dieser Befehl würde ohne Fehlermeldung abgeschlossen – er hätte aber nicht getan, was gewünscht war. In diesem Fall würde der „BITS"-Dienst auf dem lokalen System, nicht auf dem entfernten System gestartet. Der Grund liegt darin, dass `Start-Service` das `PSComputer-Name`-Attribut ignoriert und nur den Namen des Dienstes berücksichtigt. Die folgende Bildschirmabbildung liefert den Beweis.

```
PS C:\Users\hs> Invoke-Command -Session $s -ScriptBlock { Get-Service BITS }
Status   Name               DisplayName                            PSComputerName
------   ----               -----------                            --------------
Stopped  BITS               Background Intelligent Transfer Ser... f170

PS C:\Users\hs> Get-Service BITS
Status   Name               DisplayName
------   ----               -----------
Stopped  BITS               Intelligenter Hintergrundübertragun...

PS C:\Users\hs> Invoke-Command -Session $s -ScriptBlock { Get-Service BITS } | Start-Service
PS C:\Users\hs> Invoke-Command -Session $s -ScriptBlock { Get-Service BITS }
Status   Name               DisplayName                            PSComputerName
------   ----               -----------                            --------------
Stopped  BITS               Background Intelligent Transfer Ser... f170

PS C:\Users\hs> Get-Service BITS
Status   Name               DisplayName
------   ----               -----------
Running  BITS               Intelligenter Hintergrundübertragun...
```

Abbildung 18.8 *Unerwartetes Verhalten beim Fernaufruf*

Viele Commandlets funktionieren so, auch weitreichende wie `Remove-Item`. Der folgende Befehl löscht also nicht Textdateien auf dem entfernten, sondern gleichnamige Dateien auf dem lokalen System!

```
Invoke-Command -ComputerName F170 -scriptblock { Get-Item d:\Daten\
*.txt } | remove-Item
```

Richtig ist hier, den Befehl `Remove-Item` mit in den Skriptblock zu nehmen:

```
Invoke-Command -ComputerName F170 -scriptblock { Get-Item d:\Daten\
*.txt | remove-Item }
```

Grund für diese Falle ist, dass die von `Get-Item` gelieferten Dateiobjekte zwar von der PowerShell um den Parameter `PSComputername` angereichert wurden, aber das Commandlet `Remove-Item` diese Zusatzinformation leider ignoriert.



Einige Commandlets in der PowerShell 2.0, darunter `Get-Process` und `Get-Service`, bieten auch noch einen kürzeren Weg für die Fernabfrage an. Bei diesen Commandlets kann man ein einzelnes entferntes System über den Parameter `-Computer` angeben, z.B. `Get-Process -Computer F111` (Details siehe *Abschnitt 18.1, „Fernabfrage ohne WS-Management“*).

Vorteil dieser Methode ist, dass man dafür nicht WS-Management braucht, also auch ältere Betriebssysteme abfragen kann, für die es kein WS-Management gibt. Nachteil ist, dass sich die Fernabfrage immer nur auf den einzelnen Befehl bezieht. Man kann weder Befehlsfolgen noch Skripten angeben. Außerdem kann man immer nur ein einzelnes entferntes System ansprechen.

18.8 Fernausführung von Skripten

Invoke-Command

Mit `Invoke-Command` kann man natürlich ein auf dem entfernten System vorhandenes Skript starten, z.B.:

```
Invoke-Command -computer F170 -scriptblock { d:\Skripte\WPS2_
Computername.ps1 }
```

Voraussetzung ist natürlich, dass auf dem entfernten System die Skriptausführung erlaubt ist und alle für das Skript benötigten Dateien dort sind.



Abbildung 18.9 *Fehlermeldung, wenn das Starten des Skripts auf einem entfernten System nicht erlaubt ist*

Mit folgendem Befehl kann man die Skriptausführung auf einem entfernten System aktivieren:

```
Invoke-Command -computer F170 -scriptblock { Set-executionpolicy
unrestricted }
```

Das folgende Listing zeigt ein Beispiel, bei dem man die Skriptausführung aktiviert, ein Skript kopiert, dann ausführt und anschließend das Skript löscht und die Skriptausführung wieder deaktiviert.

Listing 18.1 *Entfernte Skriptausführung durch Skriptkopieren [Aufbauwissen/Remoting/WPS2_Remoting_Script.ps1]*

```
"Start Session..."
$a = New-PSSession  -ComputerName F170

"Enable Script Execution on remote System..."
Invoke-Command -Session $s -scriptblock { Set-executionpolicy
unrestricted }

"Copy Script..."
Copy-Item H:\WPS2_Computername.ps1 \\f170\c$\temp\wps2_
Computername.ps1
```

```
"Start Script..."
Invoke-Command -Session $s -scriptblock { c:\temp\WPS2_
Computername.ps1 }

"Delete Script..."
Remove-Item \\f170\c$\temp\wps2_Computername.ps1

"Disable Script Execution on remote System..."
Invoke-Command -Session $s -scriptblock { Set-executionpolicy
default }

"End Session!"
Remove-PSSession $s
```

Das gleiche Ergebnis kann man aber auch viel einfacher haben, denn das Commandlet `Invoke-Command` bietet auch die Möglichkeit, ein lokales Skript auf den entfernten Computer zu übertragen und dort zu starten:

```
Invoke-Command -computer F170 -FilePath H:\WPS2_Computername.ps1
```

18.9 Ausführung auf mehreren Computern

Das Commandlet `Invoke-Command` bietet auch die Möglichkeit, mehrere Computer in Form eines Arrays (eine durch Komma getrennte Liste) anzugeben.

Beispiel 1: Setzen von Datum und Uhrzeit auf mehreren Computern

```
Invoke-Command -computer F170, F171, F172, F173 -Script { Set-date
-date }
```



Abbildung 18.10 *Ausführen von Set-Date auf mehreren Computern*

Beispiel 2: Auslesen von c:\Temp auf mehreren Computern

```
$computer = F170, F171, F172, F173, F174
Invoke-Command -computer $computer { Get-childitem c:\temp }
```

Tipp: Für den lokalen Computer kann man „localhost“ oder „.“ verwenden.

PSComputer-Name

Das Ergebnis ist eine Gesamtliste der Ergebnisse von allen genannten Computern. Die Objekte in der Menge besitzen ein zusätzliches Attribut (NoteProperty) `PSComputerName`, das den Namen des Computers zeigt, der das Objekt geliefert hat. Dadurch ist ein Filtern/Sortieren/Gruppieren auf dem aufrufenden Computer möglich.



Die PowerShell fragt bei den einzelnen Computern in der Reihenfolge an, wie sie im Array angegeben sind. Die Reihenfolge der Ergebnisse ist jedoch abhängig davon, wann die Ergebnisse eintreffen.

Abbildung 18.11 Ergebnismenge eines Abrufs von Dienst-Objekten von drei Computern

```
PS C:\Users\HS> invoke-command -computer F171,F172,F173 -scriptblock { get-service i* } | ft name, status, PSComputer*

Name                           Status                         PSComputerName
----                           ------                         --------------
idsvc                          Stopped                        f173
IKEEXT                         Running                        f173
IPBusEnum                      Stopped                        f173
iphlpsvc                       Running                        f173
idsvc                          Stopped                        f171
IKEEXT                         Stopped                        f171
IPBusEnum                      Stopped                        f171
iphlpsvc                       Running                        f171
idsvc                          Stopped                        f172
IKEEXT                         Running                        f172
IPBusEnum                      Stopped                        f172
iphlpsvc                       Running                        f172

PS C:\Users\HS> _
```



Bei der Ausgabe von einigen Klassen (z.B. auch bei ServiceController) wird `PSComputerName` automatisch ausgegeben. Dies kann man durch `-HideComputername` unterdrücken.

```
Invoke-Command -Computer F170, F171 -ScriptBlock { Get-Service
I* } -HideComputername
```

Abbrechen eines Fernbefehls

Zum Abbrechen eines entfernt ausgeführten Befehls kann man wie bei lokalen Befehlen die Tastenkombination STRG + C verwenden.

18.10 Sitzungen

`Invoke-Command` erzeugt im Standard eine temporäre Verbindung. Alle Definitionen (Variablen und Funktionen), die im Rahmen der Ausführung von `Invoke-Command` auf einem entfernten System erzeugt wurden, sind nach Ende des Befehls wieder ungültig.

PSSession

Die Alternative ist eine permanente Verbindung (Sitzung). Eine Sitzung (engl. Session, alias „PSSession“) ist Host für die PowerShell, in der die PowerShell Befehle ausführt. Aus der Sicht von Windows ist eine Session ein Prozess.

Eine Sitzung gestaltet man durch das Laden von Snap-Ins und Modulen sowie durch die Definition von Variablen, Funktionen und Aliasen. Alle diese Einstellungen leben so lange, wie die Sitzung dauert.

Beim Start der PowerShell durch PowerShell.exe wird automatisch eine Sitzung erzeugt (Default Session). Durch Commandlets kann man weitere Sitzungen auf dem lokalen Computer oder entfernten Computern erzeugen.



Alle Fernaufrufe der PowerShell erfolgen im Rahmen einer Sitzung. Die PowerShell unterscheidet temporäre Sitzungen (mit `Invoke-Command` unter Angabe eines Computernamens) und permanente Sitzungen (mit `Invoke-Command` unter Angabe eines Sitzungsobjekts, das vorher mit `New-PSSession` erzeugt wurde).

Commandlets zur Sitzungsverwaltung

Es folgt ein Überblick über die Commandlets zur Sitzungsverwaltung:

- `New-PSSession`: Erzeugen einer neuen Sitzung auf dem lokalen oder einem entfernten Computer
- `Get-PSSession`: Liste aller Sitzungen, die aus der aktuellen Sitzung heraus gestartet wurden (zeigt aber nicht Sitzungen, die andere Computer auf dem lokalen Computer geöffnet haben)
- `Remove-PSSession`: Entfernt eine Session oder alle Sessions (`Remove-PSSession *`)
- `Enter-PSSession`: Start einer interaktiven Sitzung auf dem lokalen oder einem entfernten Computer
- `Exit-PSSession`: Ende einer interaktiven Sitzung
- `Disable-PSSessionConfiguration`: Sperren einer/aller Sitzungskonfigurationen
- `Enable-PSSessionConfiguration`: Entsperren einer/aller Sitzungskonfigurationen
- `Get-PSSessionConfiguration`: Auflisten der Sitzungskonfigurationen
- `Register-PSSessionConfiguration`: permanente Registrierung einer Sitzungskonfiguration
- `Set-PSSessionConfiguration`: Setzen von Eigenschaften einer Sitzungskonfiguration
- `Unregister-PSSessionConfiguration`: Löschen einer Sitzungskonfiguration

Sitzungen erstellen

NewPSSession

Eine Sitzung erzeugt man über New-PSSession:

```
$s = New-PSSession  -computername F171
```

Auf diese offene Sitzung muss man dann im Commandlet Invoke-Command Bezug nehmen. Ein Computername ist dann nicht mehr erforderlich.

```
Invoke-Command -session $s -scriptblock {$p = Get-Process }
```

Man kann auch permanente Sitzungen zu mehreren Computern aufbauen, indem man bei New-PSSession mehrere Computer angibt:

```
$s = New-PSSession  -computername S1, S2, S3
Invoke-Command -session $s -scriptblock {Get-culture}
```

Alternativ kann man auch mehrere einzelne Sitzungen erstellen und diese bei `Invoke-Command` angeben.

```
"Sitzungen erstellen..."
$s1 = New-PSSession  -ComputerName F173
$s2 = New-PSSession  -ComputerName E04
$s3 = New-PSSession  -ComputerName E05

"Fernzugriff auf alle drei Rechner..."
Invoke-Command -Session $s1, $s2, $s3 -ScriptBlock { Get-Service
spooler }
```

Die PowerShell-Profilskripte werden weder in temporären noch in permanenten Verbindungen automatisch auf dem entfernten System geladen. Bei Bedarf müssen sie explizit gestartet werden, z.B.:

```
Invoke-Command -session $s {. "$home\Documents\WindowsPowerShell\
Microsoft.PowerShell_profile.ps1"}
```

Schließen von Sitzungen

Remove-PSSession

Sitzungen werden automatisch geschlossen, wenn die Sitzung (Elternsitzung) beendet wird, aus der heraus die Sitzung (Kindsitzung) gestartet wurde. Entfernte Sitzungen enden zudem automatisch, wenn entfernte Computer für vier Minuten lang nicht mehr erreichbar sind. Manuell kann man eine Sitzung mit `Remove-PSSession` schließen.

Sitzungskonfigurationen

Eine Sitzungskonfiguration legt durch zahlreiche Einstellungen fest, wer eine Sitzung aufbauen darf und welche Befehle in der Sitzung zur Verfügung stehen. Typische Einstellungen sind:

- Benutzer, die sich mit dem Computer entfernt verbinden dürfen
- Größe der Objekte, die die entfernten Benutzer übertragen dürfen
- Festlegung der verfügbaren Commandlets und Funktionen



Die Konfiguration einer Sitzung ist nur möglich, wenn die PowerShell als Administrator gestartet wurde.

Die Standardkonfiguration trägt den Namen „Microsoft.PowerShell“. Auf 64-Bit-Computern gibt es zusätzlich „Microsoft.PowerShell32“. Auf Windows Server 2008 R2 gibt es außerdem „Microsoft.ServerManager“.

GetPSSessionconfiguration

Die verfügbaren Konfigurationen mit zahlreichen Details zeigt:

```
Get-PSSessionconfiguration | fl
```

oder alternativ

```
dir wsman:localhost/plugin
```

Zugriffsrechte für Fernaufrufe

Im Standard können nur Administratoren Fernaufrufe ausführen. Man kann aber die Zugriffsrechteliste (Access Control List – ACL) der Sitzungskonfiguration ändern.

```
Set-PSSessionConfiguration Microsoft.PowerShell -
ShowSecurityDescriptorUI
```

SDDL

Etwas kurios für eine kommandozeilenbasierte Shell ist, dass sich dadurch ein Windows-Fenster öffnet, wie man es von den Zugriffsrechtelisten von Windows Explorer und der Windows-Registrierungsdatenbank kennt. Rein kommandozeilenbasiert kann man arbeiten, indem man im Parameter `SecurityDescriptorSDDL` eine SDDL-Zeichenkette (SDDL = Security Descriptor Definition Language) übergibt.

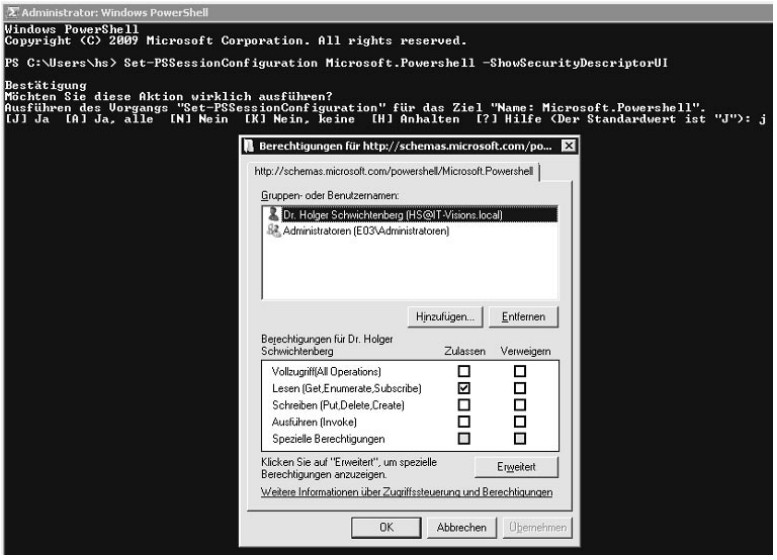


Abbildung 18.12 *Ändern der ACL für die PowerShell-Sitzungskonfiguration*

Hinweis: Das Commandlet `Disable-PSSessionConfiguration` verändert die Zugriffsrechtelisten für eine oder alle vorhandenen Sitzungskonfigurationen so, dass kein Benutzer mehr Rechte für den Fernaufruf hat.

Beispiel:

```
disable-PSSessionConfiguration -name Microsoft.PowerShell
```

Die Blockade kann man rückgängig machen mit `Enable-PSSsession-Configuration`.

Mit `Register-PSSessionConfiguration` kann man eine neue Konfiguration erzeugen. Diese wird permanent auf dem Computer gespeichert. Der erforderliche Neustart des WS-Management-Dienstes wird auf Nachfrage ausgeführt.

Beispiel Der folgende Befehl legt eine neue Konfiguration unter dem Namen „FBIConfig“ an mit einer Erhöhung der Datenmenge auf 200 MB (Standard sind 50 MB) und einem Skript, das beim Starten der Sitzung ausgeführt werden soll.

```
register-PSSessionConfiguration -name FBIConfig
-MaximumReceivedDataSizePerCommandMB 200
-StartupScript h:\Skripte\WPS2_Remoting_SessionStartSkript.ps1
```

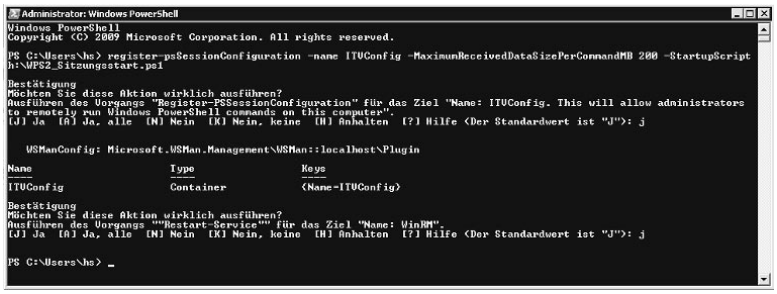


Abbildung 18.13 *Ablauf der Registrierung einer Sitzungskonfiguration*

Anschließend kann man eine Sitzung mit Bezug auf diese Konfiguration starten:

```
$s = New-PSSession  -ConfigurationName FBIConfig
```

Auch `Invoke-Command` hat den Parameter `-ConfigurationName`:

```
Invoke-Command -ConfigurationName "FBIConfig" -scriptblock { Start-
Service BITS } -computer F173
```

In `$PSSessionConfigurationName` ist der Name der Konfiguration abgelegt, die verwendet wird, wenn man den Parameter `-Configuration-Name` nicht angibt. Den Inhalt dieser Variablen kann man ändern.

Zum Löschen einer Konfiguration verwendet man:

Unregister-PSSession-Configuration

```
Unregister-PSSessionConfiguration -name FBIConfig
```

18.11 Zugriff auf entfernte Computer außerhalb der eigenen Domäne

Der Zugriff auf entfernte Computer außerhalb der eigenen bzw. einer vertrauenden Domäne ist möglich.

Grundsätzlich gibt es zwei Möglichkeiten für den domänenübergreifenden Zugriff:

- Expliziter Eintrag des Zielsystems in die Liste vertrauter Systeme
- Einrichten von Secure Socket Layer (SSL) für die HTTP-Kommunikation alias HTTPS

Herleitung des Problems

Authentifizierungsmethode

Wenn man einen domänenübergreifenden Aufruf versucht (z.B. `Enter-PSSession F171`), wird man auf folgende Fehler stoßen:

„Die Anforderung kann von WinRM nicht verarbeitet werden. Bei Verwendung der Kerberos-Authentifizierung ist der folgende Fehler aufgetreten: Der Netzwerkpfad wurde nicht gefunden. Mögliche Ursachen:

Der angegebene Benutzername oder das angegebene Kennwort ist ungültig.

– Kerberos wird verwendet, wenn keine Authentifizierungsmethode und kein Benutzername angegeben werden.

Kerberos akzeptiert Domänenbenutzernamen, aber keine lokalen Benutzernamen.

Der Dienstprinzipalname (Service Principal Name, SPN) für den Remotecomputernamen und -port ist nicht vorhanden.

Der Clientcomputer und der Remotecomputer befinden sich in unterschiedlichen Domänen, zwischen denen keine Vertrauensbeziehung besteht.

Wenn Sie die oben genannten Ursachen überprüft haben, probieren Sie folgende Aktionen aus:

– Suchen Sie in der Ereignisanzeige nach Ereignissen im Zusammenhang mit der Authentifizierung.

Ändern Sie die Authentifizierungsmethode; fügen Sie den Zielcomputer der Konfigurationseinstellung „TrustedHosts" für WinRM hinzu, oder verwenden Sie den HTTPS-Transport. Beachten Sie, dass Computer in der TrustedHosts-Liste möglicherweise nicht authentifiziert sind."

Dieser ausführliche Fehlertext liefert schon recht genaue Hinweise auf das Problem: Die Authentifizierungsmethode Kerberos funktioniert nur in Domänen. Es ist also eine andere Authentifizierungsmethode zu wählen.

Verfügbare Standardauthentifizierungsmethoden sind Default (= Kerberos), Basic, Negotiate (= Aushandlung zwischen Client und Server mit dem Simple and Protected GSSAPI Negotiation Mechanism – SPNEGA), NegotiateWithImplicitCredential, Credssp, Digest, Kerberos.

Ein neuer Versuch könnte dann also Digest oder Basic sein, also:

```
Enter-PSSession F171 -Authentication Digest
```

oder

```
Enter-PSSession F171 -Authentication Basic
```

Nun ist in beiden Fällen der Fehlertext:

„Der WinRM-Client kann die Anforderung nicht verarbeiten. Wenn der Basic- oder Digest-Authentifizierungsmechanismus verwendet wird, müssen Anforderungen den Benutzernamen und das Kennwort enthalten. Fügen Sie den Benutzernamen oder das Kennwort hinzu, oder ändern Sie den Authentifizierungsmechanismus, und wiederholen Sie die Anforderung.

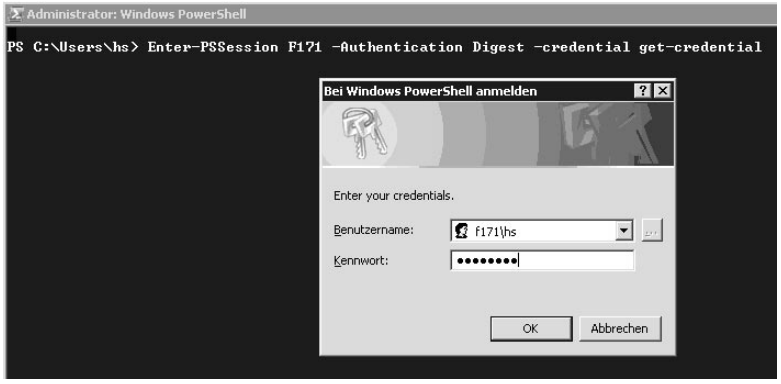
Expliziter Benutzername

Dies bedeutet also, dass die Daten des am lokalen System angemeldeten Benutzers nicht automatisch übermittelt werden (selbst wenn es auf dem Zielsystem eine gleiche Kombination aus Benutzername und Kennwort gibt).

Nun ein dritter Versuch, wobei man Client und Server die Authentifizierung aushandeln lässt:

```
Enter-PSSession F171 -Authentication Negotiate -credential Get-
credential
```

Die PowerShell zeigt den Authentifizierungsdialog …



und dann wieder einen Fehler:

Der WinRM-Client kann die Anforderung nicht verarbeiten. Wenn das Authentifizierungsschema nicht Kerberos ist oder der Clientcomputer nicht Mitglied einer Domäne ist, muss der HTTPS-Datentransport verwendet werden, oder der Zielcomputer muss der TrustedHosts-Konfigurationseinstellung hinzugefügt werden. Verwenden Sie winrm.cmd, um TrustedHosts zu konfigurieren. Beachten Sie, dass Computer in der TrustedHosts-Liste möglicherweise nicht authentifiziert sind.

Auch hier ist die Anweisung klar: Entweder ist HTTPS zu verwenden oder aber ein Eintrag in TrustedHosts vorzunehmen. Letzteres ist einfacher.

Eintrag in die Liste vertrauter Systeme

TrustedHost

Diesen Eintrag kann man über den WSMan-Navigationsprovider der PowerShell 2.0 recht elegant vornehmen:

```
cd WSMan:\localhost\Client
Set-Item trustedhosts "F170, F171, F172, F173, F174, F175" -force
Restart-Service winrm
```

Die Veränderungen der Eigenschaft „TrustedHost" erfordern eigentlich eine Rückbestätigung, daher das `-force`. TrustedHost ist eine Liste der zu vertrauenden Rechner (Rechnername oder IP-Adresse). Das Skript muss mit Administratorrechten gestartet werden.

Danach kann man dann mit einem Befehl wie folgt einen Fernzugriff ausführen, wobei die PowerShell explizit nach Benutzername und Kennwort fragen wird, selbst wenn auf dem Zielsystem die gleiche Benutzername-Kennwort-Kombination existiert.

```
Invoke-Command F175 { Get-ChildItem c:\ } -authentication negotiate
-credential
Get-Credential
```



Abbildung 18.14 *Nachfrage der PowerShell beim Zugriff auf Computer, die nicht zur (vertrauten) Domäne gehören*

Man kann auch eine dauerhafte Sitzung zu einem entfernten Computer, der nicht zur Domäne gehört, erstellen:

```
$s = New-PSSession  F175 -authentication negotiate -credential Get-
Credential
```

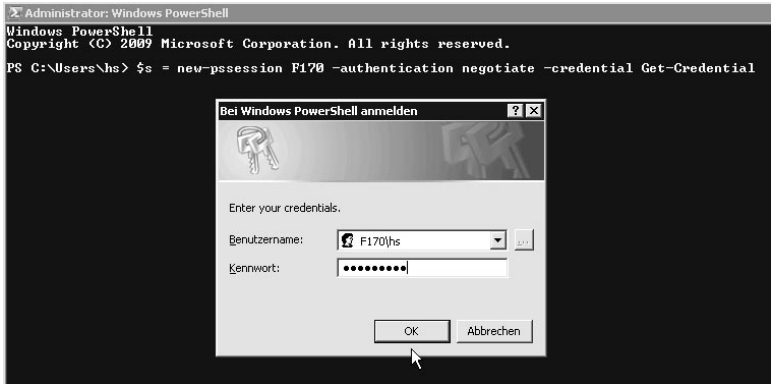


Abbildung 18.15 *Nachfrage der PowerShell beim Erstellen einer dauerhaften Sitzung zu einem Computer, die nicht zur (vertrauten) Domäne gehören*



Die Anmeldedaten immer wieder eingeben zu müssen, kann man vermeiden, indem man sich die Anmeldedaten in einer Variablen merkt:

```
$cred = Get-Credential
```

Tipp: Man muss die Anmeldedaten nicht pro Computer eingeben. Sofern Benutzername und Kennwort auf allen Systemen gleich sind, kann man auch Folgendes schreiben:

```
$cred = Get-Credential
$s = New-PSSession  -auth negotiate -cred $cred -computer F170,
F171, F171, F173, F174
Invoke-Command -Script { Set-date -date "24.6.2009 15:20:00" } -
session $s
```

***Listing 18.2**: Setzen des Datums mit Set-Date auf mehreren Nicht-Domänencomputern, die die gleiche Benutzername-Kennwort-Kombination haben [WPS2_Remoting_MultiComputer_SetDate.ps1*

18.12 Verwaltung des WS-Management-Dienstes

WSMan:

In PowerShell 2.0 ist der WS-Management-Dienst (WS-Man) über einen PowerShell-Provider (mit Namen „WSMan“) administrierbar. Im Standard erscheint in der Liste das Laufwerk „WSMan“.

```
PS C:\Users\hs> get-psdrive

Name           Used (GB)     Free (GB) Provider      Root                                               CurrentLocation
----           ---------     --------- --------      ----                                               ---------------
A                                      FileSystem    A:\
Alias                                  Alias
C                   10,90        86,65 FileSystem    C:\                                                Users\hs
cert                                   Certificate   \
Env                                    Environment
Function                               Function
G                                      FileSystem    G:\
H                    ,09        97,57 FileSystem    H:\
HKCU                                   Registry      HKEY_CURRENT_USER
HKLM                                   Registry      HKEY_LOCAL_MACHINE
S                   5,24       227,65 FileSystem    S:\
T                    ,09        37,40 FileSystem    T:\
Variable                               Variable
WSMan                                  WSMan
```

Abbildung 18.16
Liste der PowerShell-Provider in PowerShell 2.0

In diesem Laufwerk kann man wie bei anderen Providern mit den Standard-Navigations-Commandlets wie `Get-ChildItem` (`dir`) und `Get-Item`/`Set-Item` arbeiten.

```
PS C:\Users\hs>
PS C:\Users\hs> cd wsman:\localhost
PS WSMan:\localhost> dir

   WSManConfig: Microsoft.WSMan.Management\WSMan::localhost

Name                           Value                                              Type
----                           -----                                              ----
MaxEnvelopeSizekb              150                                                System.String
MaxTimeoutms                   60000                                              System.String
MaxBatchItems                  32000                                              System.String
MaxProviderRequests            4294967295                                         System.String
Client                                                                            Container
Service                                                                           Container
Shell                                                                             Container
Listener                                                                          Container
Plugin                                                                            Container
ClientCertificate                                                                 Container
```

Abbildung 18.17
Auflisten von WSMan:/localhost

Im Standard erscheint unterhalb der Laufwerkswurzel WSMan: nur „localhost“. Durch das Commandlet `Connect-WSMan` unter Angabe eines Computernamens kann man hier aber weitere Computer integrieren und im Folgenden ansteuern.

Abbildung 18.18
Einsatz von Connect-WSMan



Beispiel Setzen der vertrauenswürdigen Computer, zu denen eine Authentifizierung mit Basisauthentifizierung oder Digest toleriert werden soll.

```
cd WSMan:\localhost\Client
Set-Item trustedhosts "F170, F171, F172, F173, F174, F175" -force
Restart-Service winrm
```

19 Fehlersuche

Zur Fehlersuche bieten die Commandlets einheitliche Parameter:

Einheitliche Parameter

- Mit den Parametern `-Verbose` und `-Debug` erhält der Administrator mehr Ausgaben als normal.
- Mit `-Confirm` verlangt er, dass alle Änderungsaktionen per Nachfrage rückbestätigt werden müssen.
- Wer ganz sichergehen will, lässt mit `-WhatIf` die Aktion vor der tatsächlichen Ausführung simulieren.



Die Parameter `-Confirm` und `-WhatIf` werden nicht von allen Commandlets unterstützt.

Bei der Verwendung von `-WhatIf` beim Commandlet `Stop-Service` listet die PowerShell detailliert auf, welche Dienste Windows aufgrund von Dienstabhängigkeiten tatsächlich anhält.

Hilfreich ist `-WhatIf` auch, wenn man einen Befehl mit Platzhalter absetzt. Die folgende Abbildung zeigt, welche Dienste stoppen würden, wenn man `Stop-Service a*` ausführen würde.

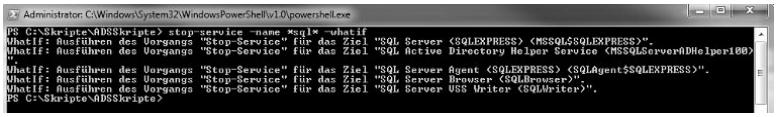


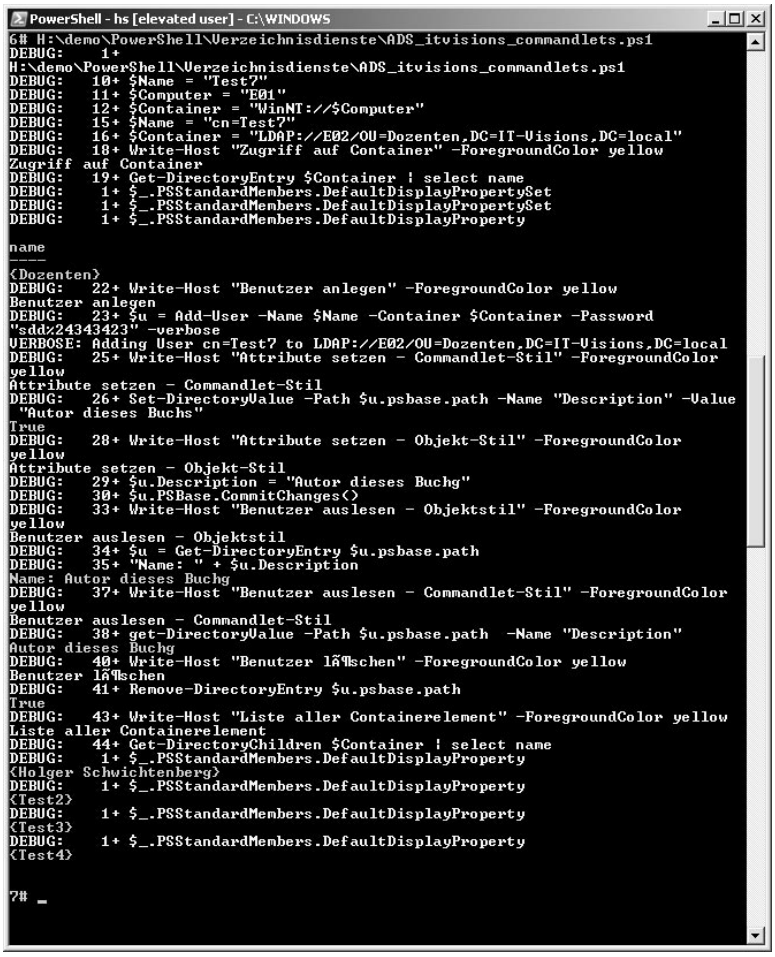
Abbildung 19.1 *Operationen mit Platzhaltern können schlimme Konsequenzen haben – whatif zeigt, welche Dienste betroffen wären.*

19.1 Detailinformationen

Verbose und Debug

Detailinformationen über die Arbeit eines einzelnen Commandlets erhält man durch den Standardparameter –verbose. Das Gleiche für ganze Skripte erreicht man durch `Set-PsDebug -trace 1` oder `Set-PsDebug -trace 2`. Die nachfolgende *Abbildung 19.2* zeigt die Ausgaben bei `-trace 1`. Bei `-trace 2` wäre die Ausgabe noch detaillierter.

Abbildung 19.2
Protokollieren eines Skriptablaufs



19.2 Einzelschrittmodus

Set-PsDebug

Mit dem Commandlet `Set-PsDebug -step` kann man Skripte schrittweise durchlaufen, d.h., die PowerShell gibt die Schritte nicht nur aus, sondern fragt auch noch nach jedem Schritt, ob die Ausführung fortgesetzt werden soll.

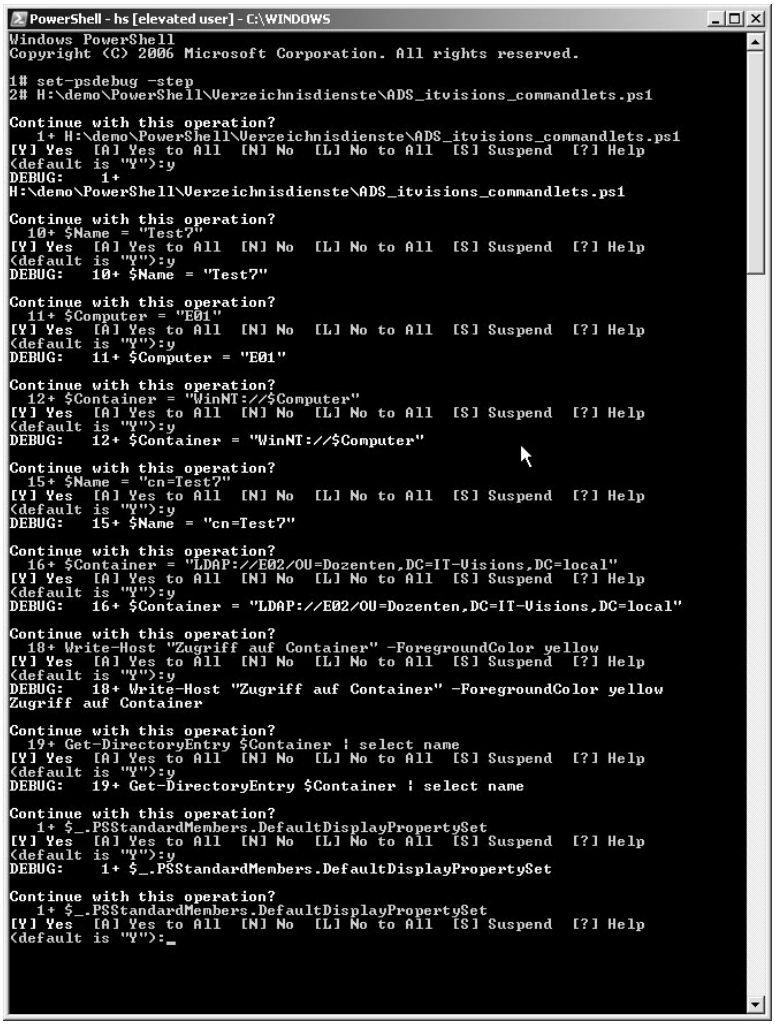


Abbildung 19.3
Durchlaufen eines Skripts in Einzelschritten

19.3 Zeitmessung

Measure-Command

Das Commandlet `Measure-Command` liefert die Angabe, wie lange ein Befehl zur Ausführung braucht – in Form eines `TimeSpan`-Objekts.

Beispiel:

```
Measure-Command { Get-Process  | Foreach-Object { $_.ws } }
```

19.4 Ablaufverfolgung

Set-TraceSource

Mit dem Commandlet `Set-TraceSource` kann man eine Ablaufverfolgung aktivieren, die detaillierte Informationen über jeden Verarbeitungsschritt liefert. `Get-TraceSource` listet alle überwachbaren Protokollquellen auf. Im Standard sind es 176. Damit wird die Komplexität des Themas deutlich, das den Rahmen dieses Buchs sprengen würde.



Beim Experimentieren mit `Set-TraceSource` können Sie schnell zu einem Punkt kommen, wo Sie vor lauter Protokollausgaben die eigentlichen Aktionen nicht mehr sehen. Um die Protokollierung wieder zu deaktivieren, verwendet man `Set-TraceSource` mit dem Parameter `-RemoveListener`.

20 Transaktionen

Die PowerShell ab Version 2.0 unterstützt Transaktionen. Dies bedeutet, dass von einer Reihe von Befehlen entweder alle oder keiner ausgeführt werden. Ein Aufruf von `Get-PSProvider` zeigt in der Spalte Capabilities, welche PowerShell-Provider Transaktionen unterstützen. In der Basisausstattung der PowerShell 2.0 ist dies aber leider nur der Registry Provider.



Abbildung 20.1
In PowerShell 2.0 unterstützt nur der Registrierungsdatenbankprovider die PowerShell-Transaktionen.

20.1 Commandlets für Transaktionen

Zur Transaktionssteuerung gibt es vier Commandlets:

- `Start-Transaction`: Beginn einer neuen Transaktion
- `Get-Transaction`: zeigt den Status der laufenden Transaktion
- `Complete-Transaction`: Erfolgreiches Ende der Transaktion, d.h., alle Änderungen werden wirksam.
- `Undo-Transaction`: Rücksetzen auf den Zustand vor der Transaktion

20.2 Start und Ende einer Transaktion

Transaktionssteuerung

Eine Transaktion wird eingeleitet mit `Start-Transaction` und mit `Complete-Transaction` erfolgreich beendet. Parameter sind dabei keine notwendig.

Eine Transaktion darf beliebig viele Schritte enthalten. Alle Befehle, die Teil der Transaktion sein sollen, benötigen den Zusatzparameter

`-useTransaction`. Man sieht nur ein Zwischenergebnis bei Lesebefehlen, die auch `-useTransaction` verwenden; alle Befehle ohne diesen Zusatz arbeiten auf dem Stand vor der Transaktion.

In dem folgenden Beispiel wird ein Registrierungsdatenbankschlüssel mit zwei Werten in einer Transaktion erzeugt.

Listing 20.1 *Transaktionsbeispiel [Aufbauwissen/Transaktionen/WPS2_Transaktion__Registry.ps1]*

```
# WPS2 / Transaktionen / Registry

$parent =  "HKCU:\Software\"
$keyname = "www.IT-Visions.de"
$key =  "HKCU:\Software\" + $keyname
$ErrorActionPreference = "continue"

if (Test-Path $key )
{ remove-Item $key  -Force }

"Transaktion starten..."
Start-Transaction

"Transaktionsstatus: " + (Get-Transaction).Status
Test-Path $key -usetransaction

New-Item -Path $parent -Name $keyname -UseTransaction -ItemType key

New-ItemProperty -path $key -name Name -value "www.IT-Visions.de -
Softwareentwicklung, Technologieberatung, Schulung" -useTransaction
New-ItemProperty -path $key -name Inhaber -value "Dr. Holger
Schwichtenberg" -useTransaction

# So sieht man den Schlüssel nicht
Test-Path $key
Get-Item $key #führt zu Fehler, der aber Transaktion nicht
abbricht!
Get-ItemProperty $key "Inhaber" #führt zu Fehler, der aber die
Transaktion nicht abbricht!

Unsinn #führt zu Fehler, der aber Transaktion nicht abbricht!

#Get-Item "HCCU:/Unsinn" -UseTransaction #würde zu Abbruch der
Transaktion führen!

# So sieht man den Schlüssel!
Test-Path $key -UseTransaction
Get-Item $key -UseTransaction
Get-ItemProperty $key "Website" -UseTransaction

"Transaktion abschließen..."
Complete-Transaction

# So sieht man den Schlüssel jetzt auch!
```

```
Test-Path $key
Get-Item $key
Get-ItemProperty $key "Inhaber"

"Transaktionsstatus: " + (Get-Transaction).Status
```

20.3 Zurücksetzen der Transaktion

Ende einer Transaktion

Ein Zurücksetzen auf den Ursprungszustand (Rollback, Abort, Undo) findet statt, wenn `Undo-Transaction` aufgerufen wird, die bei `Start-Transaction` im Parameter `-TimeOut` angegebene Zeit abgelaufen ist oder eines der an der Transaktion beteiligten Commandlets einen Fehler verursacht. Mit dem Parameter `-Rollbackpreference` in `Start-Transaction` kann man das Verhalten im Fehlerfall aber ändern. Mögliche Zustände sind:

- `Error`: Rücksetzen der Transaktion bei jeder Art von Fehler in einem der Commandlets, die `-ustransaction` verwenden (Standardeinstellung)
- `Terminating`: Rücksetzen der Transaktion nur bei Fehlern, die zum Abbruch des Befehls führen („Terminating Errors")
- `Never`: Die Transaktion wird niemals automatisch zurückgesetzt.

Wichtig: Bei der Frage, ob ein Fehler aufgetreten ist, geht es hier immer nur um die Commandlets, die `-ustransaction` verwenden. In dem folgenden Beispiel wird die Transaktion erfolgreich beendet, obwohl „unsinn" einen Fehler lieferte.

Listing 20.2 *Beispiel für Fehler in Transaktionen [Aufbauwissen/Transaktionen/WPS2_Transaktion_Fehler.ps1]*

```
$key =  "HKCU:\Software\ix"

$parent =  "HKCU:\Software\"
$keyname = "iX"
$key =  "HKCU:\Software\" + $keyname

"Transaktion starten..."
Start-Transaction
if (Test-Path $key) { remove-Item $key -UseTransaction }
New-Item -Path $parent -Name $keyname -UseTransaction -ItemType key

New-ItemProperty -path $key -name Name -value "iX - Magazin für
professionelle Informationstechnik" -useTransaction
New-ItemProperty -path $key -name Website -value "www.ix.de" -
useTransaction

Unsinn #führt zu fehler

"Transaktion abschließen..."
Complete-Transaction
```

20.4 Mehrere Transaktionen

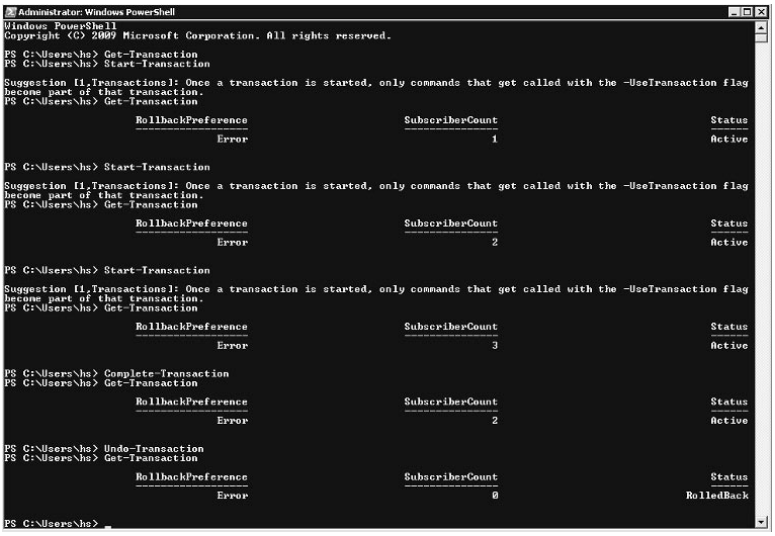
Mehrere Transaktionen

Man kann `Start-Transaction` mehrfach aufrufen, dies führt aber nicht zu unabhängigen Transaktionen. Vor Ende einer Transaktion gestartete Transaktionen sind Untertransaktionen (Teiltransaktionen, eingebettete Transaktionen) der übergeordneten Transaktion.

Die Anzahl der Ebenen sieht man bei `Get-Transaction` in der Anzahl „SubscriberCount".

Ein Aufruf von `Complete-Transaction` in einer Situation mit Untertransaktion führt zum erfolgreichen Ende der Untertransaktion. Ein Aufruf von `Undo-Transaction` in einer Situation mit Untertransaktion führt zum Abbruch der Gesamttransaktion (siehe Bildschirmabbildung).

Abbildung 20.2
Beispiel mit mehreren Aufrufen von Start-Transaction



21 Hintergrundaufträge („Jobs“)

Jobs

Windows PowerShell bietet ab Version 2.0 die Möglichkeit, Befehl und Skripte im Hintergrund auszuführen, ohne dass es während der Ausführungszeit eine Interaktion mit der aktuellen PowerShell-Sitzung gibt. Hintergrundaufträge blockieren die aktuelle PowerShell-Sitzung nicht, d.h., der Benutzer kann währenddessen andere Befehle eingeben. Hintergrundaufträge bieten sich also insbesondere für lang dauernde Befehle und Skripte an.

Folgende Commandlets sind ab PowerShell 2.0 für die Handhabung der Hintergrundaufträge implementiert:

- `Start-Job`: Start eines Hintergrundauftrags
- `Get-Job`: Liste aller laufenden und abgeschlossenen Hintergrundaufträge
- `Receive-Job`: Zugriff auf das Ergebnis eines Hintergrundauftrags
- `Stop-Job`: Beenden eines Hintergrundauftrags
- `Wait-Job`: Warten auf das Ende eines Hintergrundauftrags
- `Remove-Job`: Entfernen eines Hintergrundauftrags

21.1 Voraussetzungen

Fernausführung

Die Windows PowerShell muss für die Fernausführung konfiguriert sein, auch wenn die Hintergrundaufträge auf dem gleichen Rechner laufen. Hintergrund ist, dass die Kommunikation zwischen Hauptprozess und den Hintergrundprozessen für die Hintergrundaufträge auch über WS-Management erfolgt.

Details lesen Sie bitte in *Kapitel 18 „Fernausführung (Remoting)“*.

21.2 Architektur

Jobs laufen grundsätzlich in einem eigenen Prozess (*PowerShell.exe*). Die Kommunikation zwischen aufrufendem PowerShell-Prozess und dem ausführenden PowerShell-Prozess erfolgt über WinRM (siehe *Kapitel 18*). Bei lokalen Hintergrundaufträgen erfolgt die Kommunikation zwischen aufrufendem PowerShell-Prozess und dem ausführenden PowerShell-Prozess durch den IPC-Kanal, im entfernten Fall über HTTP.

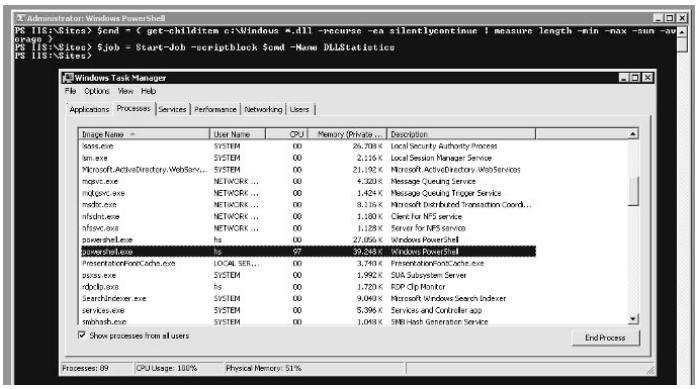


Abbildung 21.1 *Durch den Start des Hintergrundauftrags wurde eine neue Instanz der PowerShell.exe gestartet.*

21.3 Starten eines Hintergrundauftrags

Start-Job Einen Hintergrundauftrag startet der Administrator über `Start-Job` oder `Invoke-Command -AsJob`.

```
start-job -scriptblock { Get-childitem  c:\Windows -recurse}
```

`Start-Job` liefert als Ergebnis eine Instanz der Klasse `System.Management.Automation.PSRemotingJob`.

Beispiel:

Der folgende Befehl erstellt eine Statistik über alle DLLs im Windows-Installationsverzeichnis und dauert in der Regel mehrere Sekunden:

```
Get-childitem c:\Windows *.dll -recurse -ea silentlycontinue |
measure length
-min -max -sum -average
```

Es bietet sich an, diesen Befehl als Job zu starten:

```
$cmd = { Get-childitem c:\Windows *.dll -recurse -ea
silentlycontinue | measure length -min -max -sum -average }
$job = Start-Job -scriptblock $cmd -Name DLLStatistics
```

21.4 Hintergrundaufträge abfragen

PSRemotingJob Das von `Start-Job` gelieferte `PSRemotingJob`-Objekt dient der weiteren Nutzung des Auftrags.

`$job = Start-Job -scriptblock $cmd -Name DLLStatistics` `Get-Job` liefert den Zustand aller Hintergrundaufträge. Es gibt die Status: Running, Failed, Stopped und Completed.

Receive-Job Das Ergebnis des Hintergrundauftrags, also den Inhalt der Pipeline nach der Ausführung, kann man abrufen über:

```
Receive-Job $job
```

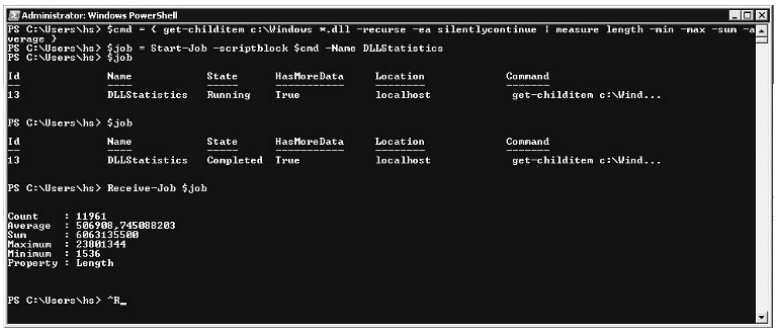


Abbildung 21.2
Ausführen und Auswerten eines Hintergrundauftrags

Genau wie bei der Fernausführung findet auch bei Hintergrundaufträgen eine Serialisierung/Deserialisierung statt, so dass die Ergebnisobjekte keine Methoden mehr besitzen.

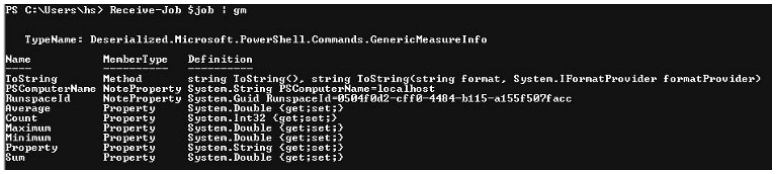


Abbildung 21.3
Inhalt der Pipeline nach Receive-Job

`Receive-Job` kann auch ausgeführt werden, wenn der Auftrag noch läuft. `Receive-Job` liefert dann alle bis dahin von dem Auftrag erzeugten Ergebnisobjekte. Bei weiteren Aufrufen von `Receive-Job` werden dann nur noch die verbliebenen Objekte geliefert.

Tipp: Mit dem Parameter `-keep` in `Receive-Job` kann man erreichen, dass bereits abgerufene Ergebnisse erneut abgerufen werden können.

21.5 Warten auf einen Hintergrundauftrag

Wait-Job

Mit `Wait-Job` kann man auf die Fertigstellung eines Hintergrundauftrags warten. Dabei kann man optional angeben, wie lange man warten möchte (Timeout in Sekunden):

```
Wait-Job -ID 10 -timeout 60
```

21.6 Abbrechen und Löschen von Aufträgen

Stop-Job

Einen Hintergrundauftrag kann man mit `Stop-Job` vorzeitig beenden:

```
Stop-Job -id 23 # Abbrechen des Auftrags Nummer 23
Stop-Job $job # Abbrechen anhand des Auftragsobjekts
```

Der Auftrag wird dann in der Auftragsliste als „Stopped“ angezeigt.

Remove-Job Mit `Remove-Job` kann man Aufträge aus der Auftragsliste löschen:

```
Remove-Job -job 23
Remove-Job $job
```

Hinweis: Man kann Aufträge nur löschen, wenn diese im Status „Stopped“ oder „Completed“ sind.

21.7 Analyse von Fehlermeldungen

JobStateInfo Die Ursache für einen fehlgeschlagenen Hintergrundauftrag kann man auslesen über:

```
$job.ChildJobs[0].JobStateInfo.Reason
```

In vielen Fällen sieht man den Fehler aber auch über `Receive-Job`.

21.8 Fernausführung von Hintergrundaufträgen

–AsJob Auch Fernaufrufe kann man als Hintergrundaufträge starten. Gerade bei langsamen Verbindungen bzw. vielen abzufragenden Computern bietet sich dies an. Hier kommt `Invoke-Command` mit dem Parameter `-AsJob` (und optional `-JobName`) zum Einsatz.

```
Invoke-Command -computername F170 -scriptblock { Get-childitem c:\
windows *.dll -recurse } -asjob -JobName "DLL_List"
```

Wahlweise kann man hier über `-ComputerName` eine temporäre Sitzung öffnen oder über `Start-PSSession` vorher eine permanente Sitzung.

Den Status des Hintergrundauftrags kann man, wie bei lokalen Aufträgen, über `Get-Job` abfragen. Das Attribut `Location` zeigt dann den entfernten Rechnernamen. Auch die Steuerung über `Receive-Job`, `Stop-Job` und `Remove-Job` steht zur Verfügung.



Abbildung 21.4 *Fernausführung eines Hintergrundauftrags*

22 Ereignissystem

In der Windows-(Programmier-)Welt gibt es verschiedene Formen von Ereignissen. Es gibt Einträge im Windows-Ereignisprotokoll, .NET-Objekte können Ereignisse auslösen und die Windows Management Instrumentation (WMI) kann Ereignisse auslösen. Auch die PowerShell kennt eigene Ereignisse.

Das Ereignissystem in PowerShell 2.0 bildet einen gemeinsamen Rahmen, in dem sich alle vier vorgenannten Ereignisarten behandeln lassen. Dabei werden die Windows-Ereignisprotokolle unter WMI-Ereignissen erfasst.

22.1 WMI-Ereignisse

WMI

Die Windows Management Instrumentation (WMI) bietet eine Ereignisinfrastruktur, die den Administrator über jegliche Veränderung im WMI-Repository (z.B. Erzeugen einer neuen Instanz, Löschen einer Instanz, Ändern des Werts einer Instanz) informieren kann. Auf diese Weise kann sich ein Administrator über Änderungen informieren lassen, z.B. das Starten oder Enden eines Prozesses oder Dienstes, das Erzeugen einer neuen Datei in einem bestimmten Pfad oder das Hinzufügen eines USB-Geräts. Den Ereignisauslöser spezifiziert man dabei durch eine sogenannte WMI-Ereignisabfrage.

22.2 WMI-Ereignisabfragen

WQL

WMI-Ereignisabfrage sind genau wie normale WMI-Abfragen in der WMI Query Language (WQL) verfasst, die SQL ähnlich ist, aber Unterschiede aufweist.

Eine WMI-Ereignisabfrage bezieht sich immer auf eine WMI-Ereignisklasse. Mit der `WHERE`-Klausel werden das zu beobachtende Managed Object (MO) und die in dem MO zu beobachtenden Attribute definiert.

```
SELECT * FROM eventklasse WHERE bedingung
```

Alle Ereignisklassen sind Unterklassen eines Ereignistyps. Die Ereignistypen wiederum sind Unterklassen der Klasse `__Event`. Sie sind in der Regel an dem führenden doppelten Unterstrich und der Endung

auf `Event` erkennbar. WMI unterscheidet vier Arten von Ereignissen (siehe Tabelle).

Tabelle 22.1 *WMI-Ereignisklassen. Die Ereignistypen sind die Oberklassen zu den rechts genannten Ereignisklassen.*

Ereignisart (Oberklasse)	Ereignisklasse
__ClassOperationEvent	__ClassCreationEvent __ClassDeletionEvent __ClassModificationEvent
__ExtrinsicEvent	__SystemEvent RegistryValueChangeEvent Win32_ComputerShutdownEvent Win32_ProcessStartTrace Win32_SystemConfigurationChangeEvent Win32_PowerManagementEvent u.a.
__InstanceOperationEvent	__InstanceCreationEvent __InstanceDeletionEvent __InstanceModificationEvent
__NamespaceOperationEvent	__NamespaceCreationEvent __NamespaceDeletionEvent __NamespaceModificationEvent



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%).

Zusätzliche Schlüsselwörter

Ereignisabfragen unterstützen als zusätzliche SQL-Schlüsselwörter `WITHIN`, `GROUP BY` und `HAVING`:

- Dabei gibt `WITHIN 10` das Abfrageintervall in Sekunden an (wird immer gebraucht, wenn es keinen speziellen Ereignissender gibt!).
- `GROUP` bündelt eine Anzahl von Einzelereignissen zu einem Gesamtereignis.
- `HAVING` dient der Definition einer Bedingung innerhalb der Gruppierung.

WQL	Erläuterung
`Select * From __InstanceCreationEvent within 3 Where TargetInstance ISA 'Win32_Process'`	Alle drei Sekunden wird geprüft, ob ein Prozess gestartet wurde.
`Select * From __InstanceDeletionEvent within 2 Where TargetInstance ISA 'Win32_Process'`	Alle zwei Sekunden wird geprüft, ob ein Prozess beendet wurde.
`SELECT * FROM __InstanceCreationEvent WITHIN 30 WHERE TargetInstance ISA 'Win32_Process' AND TargetInstance.Name = 'notepad.exe'`	Alle dreißig Sekunden wird geprüft, ob der Editor Notepad gestartet wurde.
`SELECT * FROM __InstanceModificationEvent WITHIN 5 where TargetInstance ISA 'Win32_ Service`	Alle fünf Sekunden wird geprüft, ob sich etwas an einem Systemdienst geändert hat.
`SELECT * FROM __InstanceModificationEvent WITHIN 5 WHERE TargetInstance ISA "Win32_S ervice" AND TargetInstance.State="Stopped"`	Alle fünf Sekunden wird geprüft, ob ein Systemdienst den Status *Stopped* bekommen hat.
`SELECT * FROM __InstanceCreationEvent WHERE TargetInstance ISA "Win32_NTLogEvent" AND TargetInstance.Logfile="Application" OR TargetInstance.Logfile="System"`	Jeder neue Eintrag in den Ereignisprotokollen *System* und *Application* löst ein Ereignis aus.
`SELECT * from Win32_SystemConfiguration- ChangeEvent`	Es wird ständig geprüft, ob sich die Systemkonfiguration ändert (z.B. Anstecken eines USB-Sticks).

Tabelle 22.2 *Beispiele für WMI-Ereignisabfragen*

22.3 WMI-Ereignisse mit PowerShell 1.0

Management EventWatcher

Grundsätzlich war es schon in PowerShell 1.0 möglich, eine WMI-Ereignisabfrage über die .NET-Klasse `System.Management.Management EventWatcher` zu nutzen (siehe Listing).

Listing 22.1 *Skript zum Warten auf WMI-Ereignisse [Aufbauwissen/WMI/WMI_ EventListener.wps]*

```
$scope = New-Object System.Management.ManagementScope("\\.\root\
cimV2")
$query = "SELECT * FROM __InstanceModificationEvent WITHIN 5 where
TargetInstance ISA 'Win32_Service' AND
TargetInstance.State='Stopped'"
$watcher = New-Object
System.Management.ManagementEventWatcher($scope,$query)
do
    {
        $b = $watcher.WaitForNextEvent()
        "Ereignis: Dienst gestoppt: " + $b.TargetInstance.Name
    }
while ($true) # Endlosschleife
```

22.4 Registrieren von WMI-Ereignisquellen

Ab PowerShell 2.0 ist die Unterstützung für WMI-Ereignisabfragen nun wesentlich komfortabler. Durch ein einziges Commandlet (Register-WMIEvent) kann der Administrator die WMI-Ereignisabfrage und die gewünschte Reaktion festlegen.

Register-WMIEvent

Das Commandlet `Register-WMIEvent` erzeugt ein Ereignis-Abonnement für WMI-Ereignisse. Parameter des Commandlets sind:

- `query`: Text der WMI-Ereignisabfage
- `computer`: System, auf dem die WMI-Ereignisabfrage ausgeführt werden soll. Standard ist das lokale System.
- `sourceIdentifier`: Name für das Abonnement. Dieser Name muss in der aktuellen PowerShell-Sitzung eindeutig sein.
- `messageData`: Text, der beim Auslösen des Ereignisses in die WMI-Ereignisliste geschrieben wird
- `action`: auszuführende Aktion in Form eines PowerShell-Befehls

Es folgen einige Beispiele.

Beispiel 1: Beobachten, ob sich der Status eines Dienstes ändert:

```
Register-WMIEvent -query "SELECT * FROM __InstanceModificationEvent
WITHIN 5 where TargetInstance ISA 'Win32_Service'" -
sourceIdentifier "Dienst:Status" -messageData "Der Status eines
Dienstes hat sich geändert!"
```

Beispiel 2: Beobachten, ob ein Prozess gestartet wird:

```
Register-WMIEvent -query "Select * From __InstanceCreationEvent
within 3 Where TargetInstance ISA 'Win32_Process'" -
sourceIdentifier "Prozessende" -MessageData "Ein neuer Prozess
wurde gestartet!"
```

Beispiel 3: Beobachten, ob ein Prozess beendet wird:

```
Register-WMIEvent -query "Select * From __InstanceDeletionEvent
within 3 Where TargetInstance ISA 'Win32_Process'" -
sourceIdentifier "Prozessstart" -messageData "Ein Prozess wurde
beendet!"
```

Es ist sehr einfach, ein entferntes System zu überwachen, sofern der Benutzer, der die Registrierung einleitet, dort Administrationsrechte hat. Da der Fernzugriff auf WMI basiert, ist PowerShell Remoting hier nicht erforderlich.

```
Register-WMIEvent -ComputerName F171 -query "Select * From __
InstanceCreationEvent within 1 Where TargetInstance ISA 'Win32_
Process'" -sourceIdentifier "F171 Neuer Prozess" -messageData "Ein
neuer Prozess wurde gestartet auf F171"
```

Zum Anzeigen aller registrierten Abonnements führt man aus:

```
Get-EventSubscriber
```

Zum Löschen eines Abonnements mit `Unregister-Event` muss man Bezug auf den beim Erzeugen angegebenen Namen nehmen, z.B.:

```
Unregister-Event "Dienst:Status "
```

22.5 Auslesen der Ereignisliste

Get-Event

Das Commandlet `Get-Event` listet alle aufgetretenen Ereignisse auf. Filtern ist möglich mit dem Parameter `-SourceIdentifier` oder natürlich mit dem Commandlet `Where-Object`. Jedes ausgelöste Ereignis hat einen eindeutigen EventIdentifier.

Beispiel 1:

```
Get-Event -sourceIdentifier "Dienst:Status"
```

Beispiel 2:

```
Get-Event | Where-Object {$_.MessageData -like "*Prozess*"}
```

Beispiel 3:

```
Get-event -eventidentifier 15
```

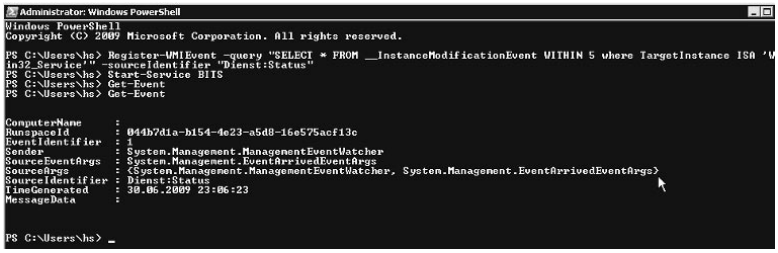


Abbildung 22.1 *Registrieren eines Ereignisses und Auslesen der Ereignisliste*

Um Details über das ausgelöste Ereignis zu erfahren, muss man das Attribut `SourceEventArgs` und dessen Unterattribute betrachten.

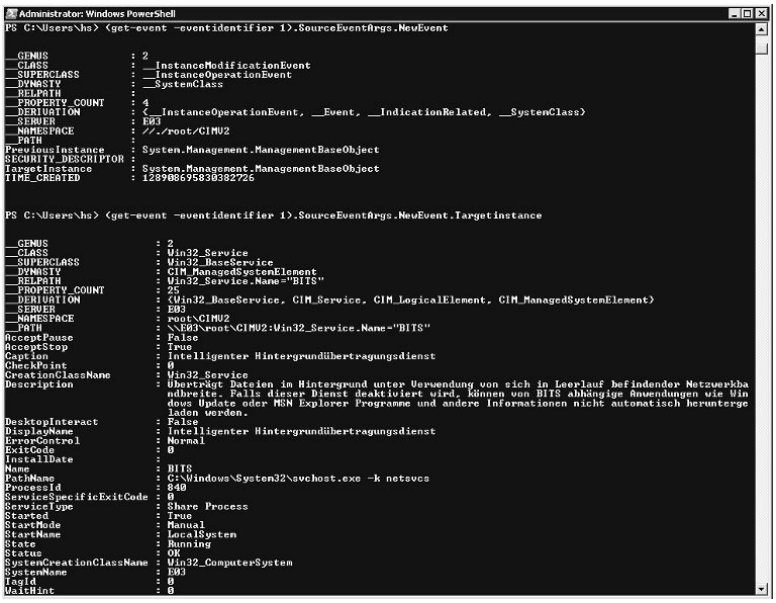
Der Zugriff auf `SourceEventArgs.NewEvent` liefert eine Instanz einer WMI-Ereignisklasse (z.B. `root/CIMV2\__InstanceCreationEvent`), verpackt in das .NET-Objekt `ManagementBaseObject`.

```
(Get-event -eventidentifier 15).SourceEventArgs.NewEvent
```

Um zu erfahren, welches WMI-Objekt das Ereignis ausgelöst hat, greift man auf `TargetInstance` zu:

```
(Get-event -eventidentifier
1).SourceEventArgs.NewEvent.Targetinstance
```

Abbildung 22.2
Ausgabe von Details zu einem Ereignis





Sie können nur aus der aktuellen PowerShell-Sitzung auf die Ereignisliste zugreifen. Ein Zugriff von einer zweiten Instanz der PowerShell ist nicht möglich!

Die Ereignisse verbleiben so lange in der Liste, bis sie explizit gelöscht werden oder die aktuelle PowerShell-Sitzung geschlossen wird.

Remove-Event

Beispiel: Löschen des Ereignisses 123

```
Remove-Event -EventIdentifier 123
```

Beispiel: Löschen aller Ereignisse von der Ereignisquelle „Neuer Prozess“

```
Remove-Event -Source "Neuer Prozess"
```



`Remove-Event` ohne Angabe eines Parameters löscht die gesamte Ereignisliste.

22.6 Reagieren auf Ereignisse

Register-Event

Mit dem Parameter `-action` kann man `Register-Event` auch eine direkte Vorgabe für eine Reaktion auf ein Ereignis in Form eines PowerShell-Skriptblocks mitgeben. In dem Skriptblock kann man über `$eventArgs` auf die gleichen Informationen zugreifen, die man bei `Get-Event` über `SourceEventArgs` erhält, z.B. bei einem WMI-Ereignis des Typs `Instance-OperationEvent`:

```
$eventArgs.NewEvent.TargetInstance.Name
```

Beispiel 1:

Einfache Ausgabe, wenn ein Prozess gestartet wird:

```
Register-WMIEvent -query "Select * From __InstanceCreationEvent
within 3 Where TargetInstance ISA 'Win32_Process'" -
sourceIdentifier "Neuer Prozess mit Aktion" -messageData "Ein neuer
Prozess wurde gestartet" -action { Write-Host "Neuer Prozess: "
$eventArgs.NewEvent.TargetInstance.Name }
```

Beispiel 2:

Einfache Ausgabe, wenn ein Dienst sich ändert:

```
Register-WMIEvent -query "SELECT * FROM __InstanceModificationEvent
WITHIN 5 where TargetInstance ISA 'Win32_Service'" -
sourceIdentifier "Dienst:Status:Aktion" -action { Write-Host  "Der
Status des Dienstes" $eventArgs.NewEvent.TargetInstance.Name " hat
sich geändert. Status ist jetzt: "
$eventArgs.NewEvent.TargetInstance.State }
```

Beispiel 3:

Einfache Ausgabe, wenn auf einem entfernten System ein Prozess startet:

```
Register-WMIEvent -ComputerName F171 -query "Select * From __
InstanceCreationEvent within 1 Where TargetInstance ISA 'Win32_
Process'" -sourceIdentifier "F171 Neuer Prozess: Aktion" -
messageData "Ein neuer Prozess wurde auf F171 gestartet " -action
{ Write-Host "Neuer Prozes auf F171: "
$eventArgs.NewEvent.TargetInstance.Name }
```

Der Skriptblock kann aber auch komplexer sein.

Listing 22.2 *Versuch, einen gestoppten Dienst neu zu starten [Aufbauwissen/Events/WPS2_WMIEvents_Service_Restart.ps1]*

```
$Aktion = {
[console]::beep(440,10)
$dienst = $eventArgs.NewEvent.TargetInstance
Write-Host  -ForegroundColor yellow
Write-Host "Dienst " $dienst.Name ": Der Status hat sich geändert.
Status ist nun: " $dienst.State  "!"  -ForegroundColor yellow
if ($dienst.State -eq "Stopped")
{
Write-Host "Neustart des Dienstes..."   -ForegroundColor yellow
Start-Service $dienst.Name
}
}
Register-WMIEvent -query "SELECT * FROM __InstanceModificationEvent
WITHIN 1 where TargetInstance ISA 'Win32_Service'" -
sourceIdentifier "Dienst:Status:Aktion" -action $Aktion
```

Das obige Skript registriert bei Änderungen an Diensten einen Skriptblock, der prüft, ob ein Dienst nun beendet ist, und diesen dann neu startet. Wie die folgende Bildschirmabbildung zeigt, wird durch den Neustart ein weiteres Ereignis ausgelöst.

Abbildung 22.3 *Ablauf des Skripts WPS2_WMIEvents_DienstRestart.ps1*



22.7 Registrieren von .NET-Ereignissen

FileSystem-Watcher

Auch viele .NET-Klassen besitzen Ereignisse, für die man sich in der PowerShell 2.0 registrieren kann. Das Beispiel zeigt die Registrierung für die Ereignisse `Created`, `Changed` und `Deleted` für die Klasse `System.IO.FileSystemWatcher`, mit der man Veränderungen im Dateisystem überwachen kann.

Listing 22.3 Warten auf Ereignisse im Dateisystem[WPS2_DOTNETEvents_Filesystemobject.ps1]

```
$fsw = New-Object System.IO.FileSystemWatcher
$fsw.Path = "c:\temp"

$aktion = {
[console]::beep(440,10)
Write-Host "Dateisystemereignis: " $eventArgs.FullPath ": "
$eventArgs.ChangeType
Write-Host
    }

Register-ObjectEvent -InputObject $fsw -EventName Created -Action
$aktion
Register-ObjectEvent -InputObject $fsw -EventName Changed -Action
$aktion
Register-ObjectEvent -InputObject $fsw -EventName Deleted -Action
$aktion
```

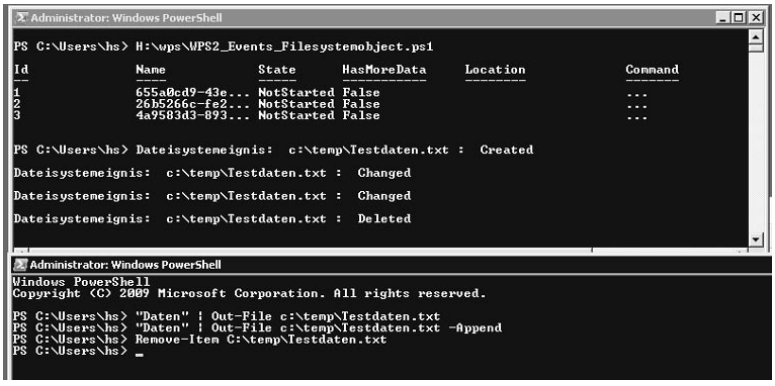


Abbildung 22.4 Einsatz des obigen Skripts zur Datensystemüberwachung. Das obere PowerShell-Fenster registriert die Änderungen, die das untere am Dateisystem vornimmt.



Die Ereignisbindung für mehrere Ereignisse kann man auch elegant in einer Schleife erledigen:

Trick: Alle Ereignisse auf einmal binden

```
foreach ($o in @("Created", "Deleted", "Changed", "Renamed"))
{
     Register-ObjectEvent $fsw $o -Action $aktion
}
```

22.8 Erzeugen von Ereignissen

New-Event

Die Ereignisinfrastruktur der PowerShell kann man auch unabhängig von WMI verwenden. Die PowerShell 2.0 bietet hierfür das Commandlet `New-Event` zum Erzeugen eigenständiger Ereignisse.

```
New-Event -SourceIdentifier "Import beendet" -sender
"Datenimportskript" -MessageData "Der Datenimport ist beendet" -
eventarguments "Anzahl: 23345", "Dauer: 12 sek"
```

Diese Ereignisse wandern auch in die Ereignisliste, die man mit `Get-Event` einsehen kann. Auch diese Ereignisse gelten aber nur für die aktuelle Sitzung.

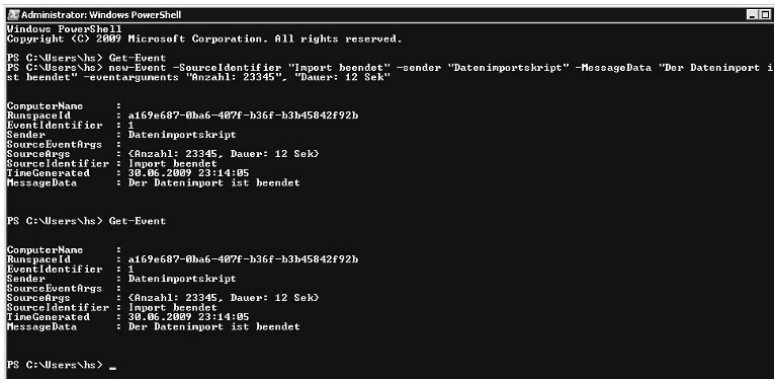


Abbildung 22.5 *Erzeugen eigener Ereignisse*

23 Datenbereiche und Datendateien

Windows PowerShell 2.0 unterstützt Datenbereiche in Skripten bzw. in externen Dateien, mit denen man Daten und Programmcode besser voneinander trennen kann. Die Daten können in mehreren Sprachen vorliegen, damit sich mehrsprachige Skripte einfacher erstellen lassen.

23.1 Datenbereiche

ConvertFrom-StringData

Das folgende Beispiel zeigt einen Datenbereich und seine Verwendung. Der Datenbereich hat den Namen `$Texte` und besitzt drei Elemente: Startmeldung, Fehlermeldung und Endemeldung. Die Anweisung „Data" in Verbindung mit `ConvertFrom-StringData` erzeugt ein Hash-Tabellen-Objekt. Im Programmcode kann man dann über `$Datenbereichname.Elementname` (z.B. `$Texte.Startmeldung`) auf die Texte zugreifen.

Der Datenbereich darf Kommentare beinhalten (beginnend mit dem Zeichen „#", siehe Beispiel).

Listing 23.1 Beispiel für die Nutzung von Datenbereichen [Aufbauwissen/Data/WPS2_DataSection.ps1]

```
$Texte = Data {
ConvertFrom-StringData @'
# Standardmeldungen
Startmeldung = Skript beginnt
Endemeldung = Skript beendet!
'@}

$Fehlertexte = Data {
ConvertFrom-StringData @'
# Besondere Meldungen
Fehlermeldung = Es ist ein Fehler aufgetreten: Arbeiten Sie nicht
nach 17 Uhr!
'@}

$Texte.Startmeldung

if ([DateTime]::Now.Hour -ge 17) { Write-Host
$Fehlertexte.Fehlermeldung }
```



Jede Zeile in der Datensektion ist ein Element. Kommentare müssen in eigenen Zeilen stehen.

23.2 Datendateien

.psd1 Ein Datenbereich kann in einer eigenständigen Datei mit der Dateinamenerweiterung .psd1 gespeichert werden. Ein Skript lädt automatisch zum Start des Skripts eine .psd1-Datei, die im gleichen Verzeichnis liegt und den gleichen Namen besitzt.

Listing 23.2 *[Aufbauwissen/Data/Data/WPS_DataFile.psd1]*

```
$Texte = Data {
ConvertFrom-StringData @'
# Standardmeldungen
Startmeldung = Skript beginnt
Endemeldung = Skript beendet!
'@}

$Fehlertexte = Data {
ConvertFrom-StringData @'
# Besondere Meldungen
Fehlermeldung = Es ist ein Fehler aufgetreten: Arbeiten Sie nicht
nach 17 Uhr!
```

Listing 23.3 *WPS_DataFile.ps1 [Aufbauwissen/Data/ WPS2_DataFile.ps1]*

```
$PSUICulture
$Texte.Startmeldung
if ([DateTime]::Now.Hour -ge 17) { Write-Host
$Fehlertexte.Fehlermeldung }
$Texte.Endemeldung
```

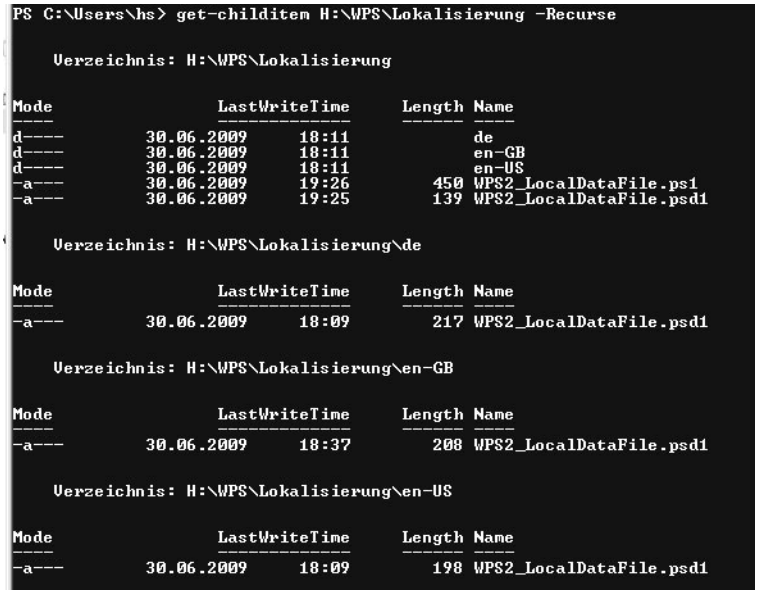
23.3 Mehrsprachigkeit/Lokalisierung

Die Windows PowerShell 2.0 unterstützt die Lokalisierung von Skripten durch das Anlegen von mehreren Datendateien für unterschiedliche Sprachregionen. Dabei verwendet die PowerShell ein Verfahren, das angelehnt ist an das im .NET Framework verwendete Verfahren, aber nicht ganz identisch ist.

Sprachkürzel Sprachen unterscheidet die PowerShell wie das .NET Framework mit Sprachkürzeln, z.B. „de“ für die deutsche Sprache und „en“ für die englische Sprache. Man kann auch genauer nach Regionen differenzieren, z.B. „de-DE“ für die deutsche Sprache in Deutschland, „de-CH“ für die deutsche Sprache in der Schweiz, „en-GB“ für Englisch in England und „en-US“ für Englisch in den USA.

Sofern im Skript keine andere Weisung erfolgt, orientiert die PowerShell sich an der für die Windows-Benutzeroberfläche eingestellten Anzeigesprache, die immer aus Sprache und Region besteht. Wird die Kombination (z.B. „de-DE“) nicht gefunden, wird nur nach der Sprache (z.B. „de“) gesucht. Wird auch diese nicht gefunden, sucht die PowerShell nach einer Standardsprache (neutrale Sprache).

Zunächst legt man für jedes Sprachkürzel ein Unterverzeichnis relativ zum Pfad des Skripts an. Jedes Unterverzeichnis erhält eine sprachspezifische Kopie der .psd1-Datendatei.



```
PS C:\Users\hs> get-childitem H:\WPS\Lokalisierung -Recurse

    Verzeichnis: H:\WPS\Lokalisierung

Mode                LastWriteTime     Length Name
----                -------------     ------ ----
d----         30.06.2009     18:11            de
d----         30.06.2009     18:11            en-GB
d----         30.06.2009     18:11            en-US
-a---         30.06.2009     19:26        450 WPS2_LocalDataFile.ps1
-a---         30.06.2009     19:25        139 WPS2_LocalDataFile.psd1

    Verzeichnis: H:\WPS\Lokalisierung\de

Mode                LastWriteTime     Length Name
----                -------------     ------ ----
-a---         30.06.2009     18:09        217 WPS2_LocalDataFile.psd1

    Verzeichnis: H:\WPS\Lokalisierung\en-GB

Mode                LastWriteTime     Length Name
----                -------------     ------ ----
-a---         30.06.2009     18:37        208 WPS2_LocalDataFile.psd1

    Verzeichnis: H:\WPS\Lokalisierung\en-US

Mode                LastWriteTime     Length Name
----                -------------     ------ ----
-a---         30.06.2009     18:09        198 WPS2_LocalDataFile.psd1
```

Abbildung 23.1 *Verzeichnisstruktur für Lokalisierung mit Deutsch, britischem Englisch und amerikanischem Englisch sowie einer „neutralen" Sprache*

Die Datendateien müssen etwas anders aufgebaut sein; es darf hier keine Variablendeklaration geben.

Listing 23.4 *[Data/en-US/WPS2_LocalDataFile.psd1]*

```
ConvertFrom-StringData @'
# Standardmeldungen
Startmeldung = Script starts...
Endemeldung = Script finished!
# Besondere Meldungen
Fehlermeldung = An error occured: Don't work after 5 p.m.!
'@
```

Listing 23.5 *[Data en-GB/WPS2_LocalDataFile.psd1] (etwas höflicher als im Amerikanischen)*

```
ConvertFrom-StringData @'
# Standardmeldungen
Startmeldung = Script starts...
Endemeldung = Script finished!
# Besondere Meldungen
Fehlermeldung = An error occured: Don't work after 5 p.m., please!
'@
```

Listing 23.6 *[Data de-DE/WPS2_LocalDataFile.psd1]*

```
ConvertFrom-StringData @'
# Standardmeldungen
Startmeldung = Skript beginnt!
Endemeldung = Skript beendet!
# Besondere Meldungen
```

```
Fehlermeldung = Es ist ein Fehler aufgetreten: Arbeiten Sie nicht
nach 17 Uhr!
'@
```

Listing 23.7 *[Data WPS2_LocalDataFile.psd1] – die „neutrale" Sprache*

```
ConvertFrom-StringData @'
# Standardmeldungen
Startmeldung = START
Endemeldung = END
# Besondere Meldungen
Fehlermeldung = ERROR
'@
```

Import-LocalizedData

Im Skript muss man explizit das Laden der sprachspezifischen Datendateien auslösen und einen Variablennamen zuordnen. Der folgende Befehl lädt die Datendatei gemäß der aktuellen Windows-Sprache und bindet die Elementnamen an den Variablennamen $Texte.

```
Import-LocalizedData -Binding Texte
```

Danach kann man über $Texte auf die Texte zugreifen, ohne eine weitere Fallunterscheidung nach Sprachen durchführen zu müssen. Die Sprache richtet sich automatisch nach der aktuellen Anzeigesprache von Windows.

```
$Texte.Startmeldung
if ([DateTime]::Now.Hour -ge 17) { Write-Host $Texte.Fehlermeldung
}
$Texte.Endemeldung
```

Die aktuelle Anzeigesprache kann man über die eingebaute Variable $PSUICulture abfragen. Leider kann man hier nicht schreibend zugreifen, wenn man die Sprache explizit setzen möchte. Hier gibt es zwei Möglichkeiten:

- Man kann in `Import-LocalizedData` die Sprache explizit angeben, z.B. `Import-LocalizedData -Binding Texte -UICulture en-US`.
- Man kann über die .NET-Klassenbibliothek die Sprache für den ganzen Thread ändern, z.B. `[System.Threading.Thread]::CurrentThread.CurrentUICulture = New-Object System.Globalization.CultureInfo("en-US")`.



Nach einem Ändern der Thread-Sprache muss man `Import-LocalizedData` erneut aufrufen!

Beispiel

Zum Test sei folgendes Skript mit den oben bereits dargestellten Dateidateien verwendet.

```
Import-LocalizedData -Binding Texte
$PSUICulture
$Texte.Startmeldung
if ([DateTime]::Now.Hour -ge 17) { Write-Host $Texte.Fehlermeldung
}
$Texte.Endemeldung
```

Listing 23.8
Beispielskript für Lokalisierung [WPS2_DataFile_Localized.ps1]

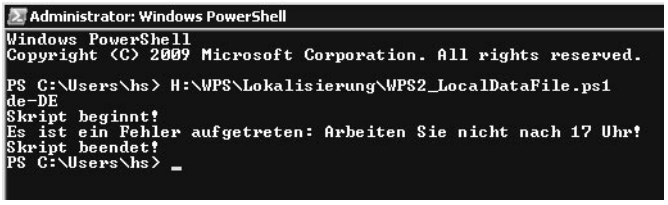


Abbildung 23.2
Ausgabe des Skripts auf einem Windows-System, auf dem die Anzeigesprache Deutsch ist

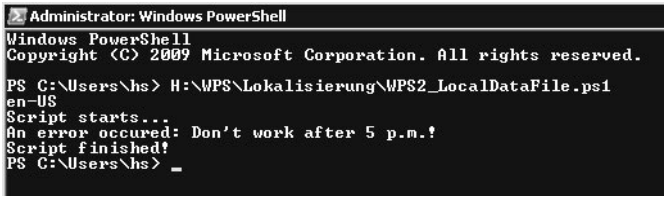


Abbildung 23.3
Ausgabe des Skripts auf einem Windows-System, auf dem die Anzeigesprache Englisch ist

`$PSUICulture` ist die eingestellte „Anzeigesprache". `$PSCulture` ist die „Formatsprache" für Datum, Uhrzeit und Währung.



24 PowerShell-Snap-Ins

Die Windows PowerShell besitzt keinen festen Satz von Commandlets. Zusätzliche Commandlets können beim Start der PowerShell oder jederzeit beim Betrieb hinzugefügt werden. Zusätzliche Commandlets sind entweder als Skriptdateien implementiert, die über das „Dot Sourcing" hinzugefügt werden (vgl. *Abschnitt 6.2 „Start eines Skripts"*) oder durch die Installation eines Snap-In (im Folgenden beschrieben).

24.1 Einbinden von Snap-Ins

Commandlet-Erweiterungen, die in Form einer Snap-In-DLL vorliegen, müssen in zwei Schritten in die PowerShell eingebunden werden:

- Registrieren der DLL (alias Assembly), welche die Commandlets enthält
- Hinzufügen des Snap-In zur PowerShell-Konsole

Registrieren der DLL

Das Registrieren der DLL erfolgt mit dem Kommandozeilenwerkzeug *installutil.exe*, das mit dem .NET Framework mitgeliefert wird. Das Werkzeug finden Sie im Installationsordner des .NET Frameworks (in der Regel *c:\Windows\Microsoft .NET\Framework\v x.y*). Die Windows PowerShell hat diesen Pfad automatisch als Suchpfad eingebunden.

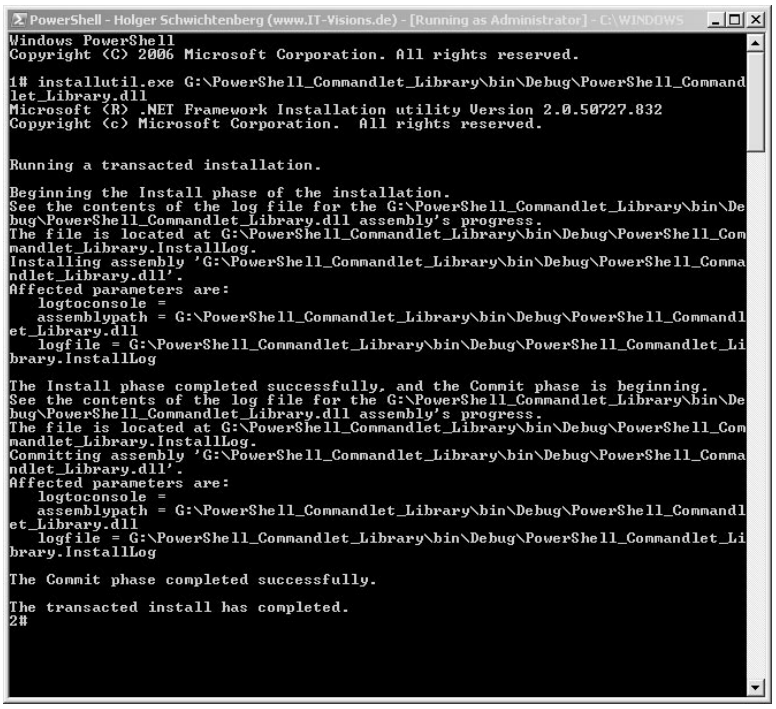
installutil.exe

Bei *installutil.exe* anzugeben ist der Dateiname zu der Erweiterungs-DLL, inklusive Pfad (sofern die PowerShell-Konsole genau diesen Pfad nicht schon als aktuellen Pfad hat).

```
installutil.exe G:\PowerShell_Commandlet_Library\bin\Debug\
PowerShell_Command
```

Die folgende Bildschirmabbildung zeigt, wie das Werkzeug die erfolgreiche Installation quittiert.

Abbildung 24.1
Ausgabe von InstallUtil.exe



Das Registrieren führt dazu, dass die DLL in der Registrierungsdatenbank unter *HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\PowerShell\1\PowerShellSnapIns* eingetragen wird.

Hinzufügen des Snap-In zur PowerShell-Konsole

Zum Aktivieren des Snap-In muss man in der PowerShell-Konsole das Commandlet `Add-PSSnapin` verwenden. Dieses Commandlet lädt die Erweiterung:

```
Add-PSSnapin PowerShell_Commandlet_Library
```

Laden des Snap-In

Während das Registrieren der DLL nur einmal notwendig ist, verwirft die PowerShell-Konsole bei jedem Beenden alle geladenen Snap-Ins. Wenn man möchte, dass die PowerShell immer mit bestimmten Erweiterungen geladen wird, gibt es zwei Möglichkeiten:

- Aufnehmen der entsprechenden `Add-PSSnapIn`-Anweisungen in Ihre systemweite oder benutzerspezifische Profildatei.

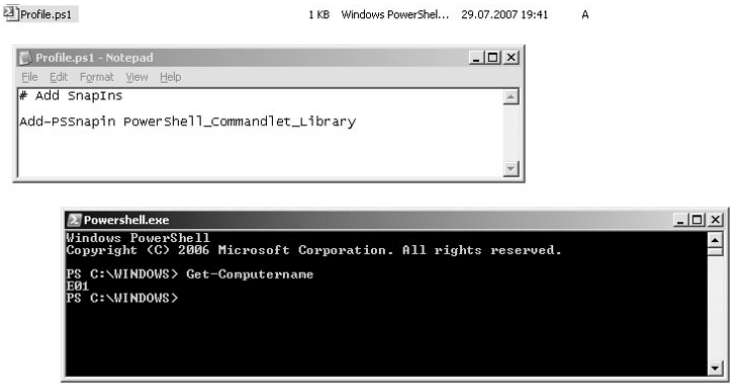


Abbildung 24.2
Laden eines Snap-In in der Profildatei

- Exportieren einer Konsolenkonfigurationsdatei mit `Export-Console`. Sie müssen allerdings vorher erst das Snap-In in der aktuellen Konsole hinzufügen und dann diese aktuelle Konsole exportieren. Dabei entsteht eine XML-Datei mit der Dateinamens-erweiterung *.psc1*. Diese *.psc1*-Datei muss dann beim Starten der PowerShell mit dem Kommandozeilenparameter *–PSConsoleFile* übergeben werden.

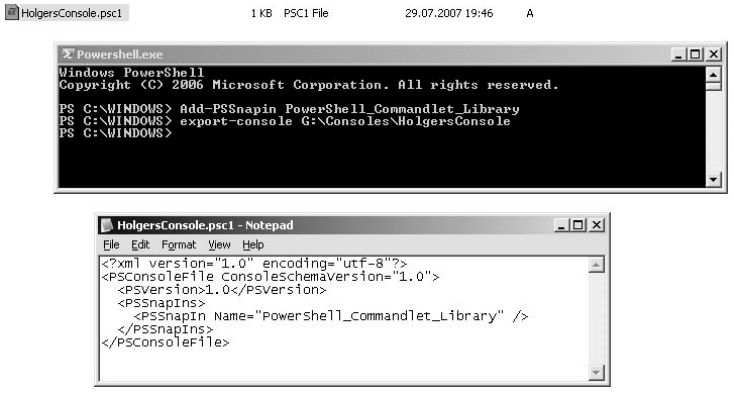


Abbildung 24.3
Exportieren einer Konsolenkonfigurationsdatei

Am besten legt man sich eine Verknüpfung im Dateisystem mit folgendem Ziel an:

```
%SystemRoot%\system32\WindowsPowerShell\v1.0\powershell.exe -
PSConsoleFile "G:\Consoles\HolgersConsole.psc1"
```

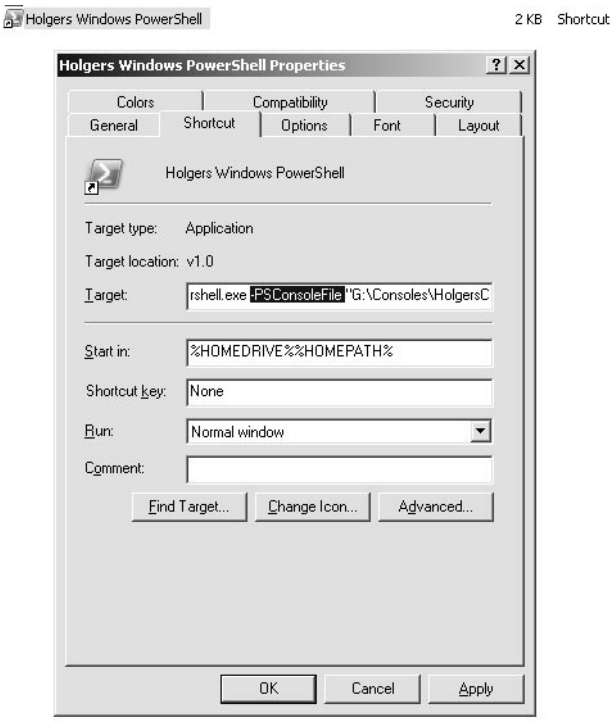


Abbildung 24.4 *Anlegen einer Verknüpfung zur PowerShell-Konsole, die automatisch eine bestimmte Konsolenkonfigurationsdatei mitlädt*

Liste der Snap-Ins



Das Commandlet `Get-PSSnapIn` listet im Normalfall nur die Snap-Ins auf, die bereits der PowerShell mit `Add-PSSnapIn` hinzugefügt wurden. Darunter findet man auch die Standard-Commandlet-Pakete, die mit `Microsoft.PowerShell.*` beginnen (siehe *Abbildung 24.5*).

Mit `Get-PSSnapin -registered` listet man hingegen alle registrierten Snap-Ins auf, unabhängig davon, ob diese in der aktuellen Konsole aktiv sind. In der folgenden Bildschirmabbildung sieht man die WorldWideWings_PowerShell_Extensions, die nicht in der Konsole aktiv sind (siehe *Abbildung 24.6*).

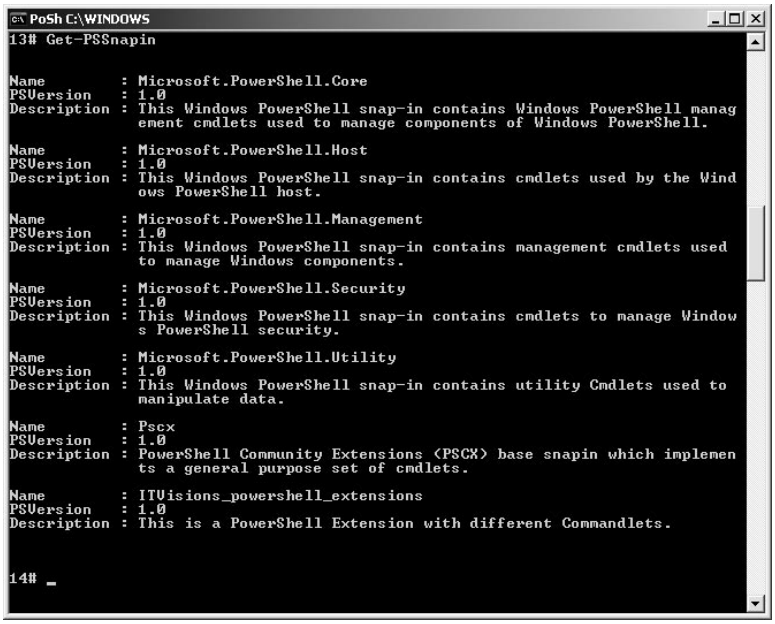


Abbildung 24.5
Aktive PowerShell-Snap-Ins

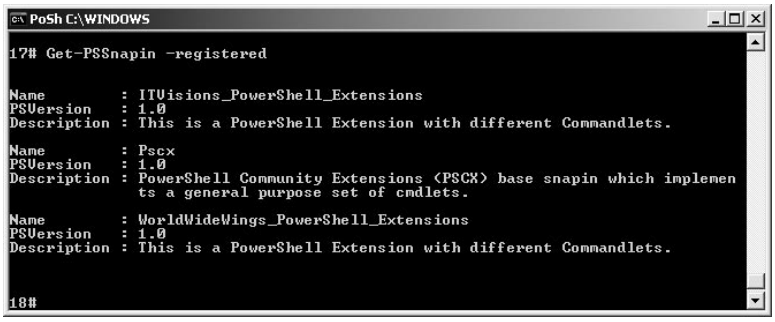


Abbildung 24.6
Alle auf dem System registrierten Commandlets

24.2 Liste der Commandlets

Um sich alle Commandlets einer bestimmten Erweiterung auflisten zu lassen, können Sie nach dem Attribut `PSSnapIn` in der Klasse `CmdletInfo` filtern, z.B.:

```
Get-command | where { $_.pssnapin -like „Pscx" }
```

oder

```
Get-command | where { $_.pssnapin -like "ITVisions_PowerShell_
Extensions" }
```

oder

```
Get-command | where { $_.pssnapin -like
"quest.activeroles.admanagement" }
```

24.3 Doppeldeutige Namen

Es ist möglich, dass Sie verschiedene Commandlet-Erweiterungen aktivieren, die gleichnamige Commandlets definieren, denn es gibt keine zentrale Registrierungsstelle für Commandlets. Wenn Sie auf dieses Problem stoßen, wird die PowerShell einen Aufruf eines doppeldeutigen Commandlets mit einem Fehler quittieren (siehe *Abbildung 24.7*).



Bitte beachten Sie, dass der Fehler tatsächlich erst im Betrieb auftritt, nicht schon beim Start der PowerShell-Konsole.

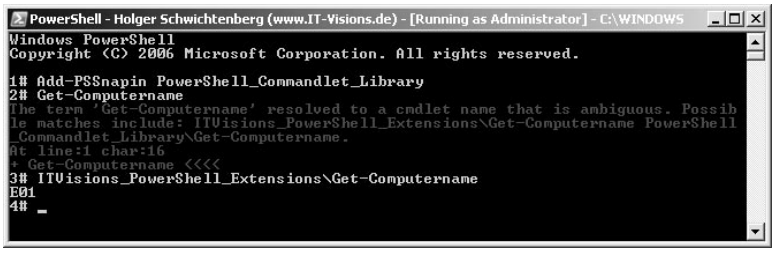


Abbildung 24.7
Ein Commandlet-Name ist doppelt vergeben.

Die Lösung zur Unterscheidung zwischen den beiden gleichnamigen Commandlets besteht darin, den Namen des Snap-In dem Commandlet voranzustellen (getrennt durch einen Schrägstrich), z.B.:

```
ITVisions_PowerShell_Extensions\Get-Computername
```

25 PowerShell-Module

Module versus Snap-Ins

Module sind Pakete, die die PowerShell durch Funktionalität erweitern. Module können aus Navigationsprovidern, Commandlets, Funktionen, vordefinierten Variablen und Aliasen bestehen. Auch Typformatierungsdateien (.ps1xml) und Dateidateien (z.B. Textdateien, XML-Datei, Datenbankdateien) können enthalten sein.

Durch Module können Entwickler bzw. Administratoren Funktionalität zur Wiederverwendung an andere Personen weitergeben. Der Empfänger muss das Modul importieren und kann die darin enthaltene Funktionalität dann nutzen wie die Kernfunktionalität der PowerShell.

Die bereits in PowerShell 1.0 eingeführten Snap-Ins sind .NET-Komponenten (Assemblies), die nur Navigationsprovider und Commandlets enthalten können. Module sind ein übergeordnetes Konzept, in dem neben Snap-Ins auch PowerShell-Skripte liegen können.

25.1 Überblick über die Commandlets

Folgende Commandlets sind für PowerShell-Module wichtig:

- `Get-Module` liefert eine Liste der installierten bzw. importierten Module.
- `New-Module` erzeugt ein neues sogenanntes dynamisches Modul im Hauptspeicher.
- `Export-ModuleMember` legt innerhalb eines Moduls fest, welche Funktionen von außen verfügbar sein sollen.
- `Import-Module` lädt ein Modul in die aktuelle PowerShell-Sitzung. Nach dem Ende der Sitzung ist das Modul nicht mehr verfügbar. Das Commandlet kann im Profilskript verwendet werden.
- `Remove-Module` entfernt ein Modul aus der aktuellen Sitzung.

25.2 Modularchitektur

Modulverzeichnisse

Ein Modul ist im installierten Zustand ein Dateisystemverzeichnis, das die Moduldateien enthält. Systemmodule werden vom Betriebssystem installiert unter *\Windows\System32\WindowsPowerShell\v1.0\Modules*. Benutzer können Module installieren unter *$home\Documents\WindowsPowerShell\Modules*.

Tipp: Die Umgebungsvariable `PSModulePath` liefert den Pfad zu den Benutzermodulen. Man kann diesen Pfad ändern oder auch weitere Pfade hinzufügen, z.B.:

```
$env:psmodulepath = $env:psmodulepath + ";h:\WPSModules"
```

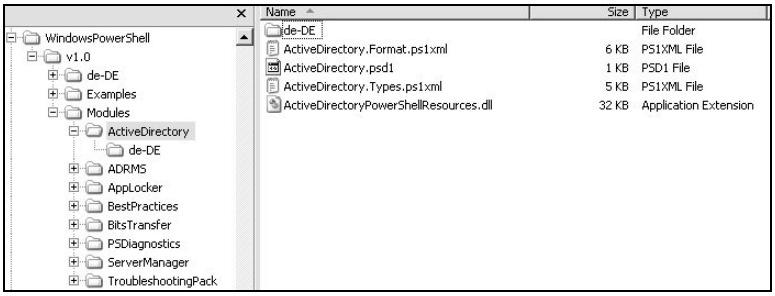


Abbildung 25.1 *Systemmodulverzeichnis und Inhalt des Moduls „ActiveDirectory" in Windows Server 2008*

Ein Modul besitzt ein Modul-Manifest (.psd1-Datei), in dem die zu dem Modul gehörenden Dateien festgelegt sind.

Listing 25.1 *Modul-Manifest für das Modul "ActiveDirectory"*

```
@{
GUID="{43c15630-959c-49e4-a977-758c5cc93408}"
Author="Microsoft Corporation"
CompanyName="Microsoft"
ModuleVersion="1.0.0.0"
Description="Active Directory Module"
PowerShellVersion="2.0"
CLRVersion="2.0.50727"
NestedModules="Microsoft.ActiveDirectory.Management"
RequiredAssemblies="Microsoft.ActiveDirectory.Management"
TypesToProcess="ActiveDirectory.Types.ps1xml"
FormatsToProcess="ActiveDirectory.Format.ps1xml"
}
```

Mit dem Commandlet `Test-ModuleManifest` kann man prüfen, ob ein Manifest korrekt ist und alle dort genannten Dateien in dem Modulverzeichnis existieren.

Beispiel:

```
Test-ModuleManifest C:\windows\system32\WindowsPowerShell\v1.0\
Modules\ActiveDirectory\ActiveDirectory.psd1
```

Der in der PowerShell-Hilfe angegebene Link zum Artikel „Writing a Windows PowerShell Module“ (*http://go.microsoft.com/fwlink? LinkId =144916*) führt (bis zum Redaktionsschluss dieses Buchs) leider nicht zum gewünschten Ziel. Unter diesem Link finden Sie aber trotzdem die notwendigen Informationen: *http://msdn.microsoft.com/en-us/library/dd878310(VS.85).aspx*.

25.3 Module installieren

Zum Installieren eines PowerShell-Moduls kopiert man alle Dateien des Moduls in eines der PowerShell-Modulverzeichnisse. Wichtig ist, dass die Dateien nicht direkt in dem PowerShell-Modulverzeichnis liegen, sondern in einem seiner Unterordner.

Falsch:

```
C:\windows\system32\WindowsPowerShell\v1.0\Modules\
ActiveDirectory.psd1
```

Richtig:

```
C:\windows\system32\WindowsPowerShell\v1.0\Modules\ActiveDirectory\
ActiveDirectory.psd1
```

25.4 Importieren von Modulen

Der Befehl

Get-Module

```
Get-Module -listAvailable
```

zeigt alle auf dem System installierten Module. Zur Verwendung eines Moduls muss dieses aber nicht nur installiert, sondern auch in der aktuellen PowerShell-Sitzung importiert sein.

```
PS C:\Users\hs.ITV> Get-Module -ListAvailable | ft name, ModuleType, Description, Version

Name                                ModuleType Description                    Version
----                                ---------- -----------                    -------
DesktopSearch                          Script                                0.0
DotNet                                 Script                                0.0
FileSystem                          Manifest Windows PowerShell module ...   1.0
IsePack                             Manifest A Collection of Tools for ...   1.1
LDAPCommandlets                        Script                                0.0
PowerShellPack                      Manifest Windows 7 Resource Kit Pow...   1.0
PSCodeGen                           Manifest Scripts to help with manag...   1.0
PSImageTools                        Manifest A powershell module to wor...   1.0
PSRSS                               Manifest Powershell functions to ge...   1.1
PSSystemTools                       Manifest A module to get and modify...   1.0
PSUserTools                         Manifest A Windows PowerShell Modul...   1.0
TaskScheduler                       Manifest A set of scripts to work w...   1.0
WPK                                 Manifest The WPF Powershell Kit is ...   1.0
WWWings_Commandlets                    Binary                                0.0
WWWings_ProviderAndCommand...       Manifest Modul für World Wide Wings...   1.0
ActiveDirectory                     Manifest Active Directory Module         1.0.0.0
AppLocker                           Manifest Powershell AppLocker Module     1.0.0.0
BitsTransfer                        Manifest                                 1.0.0.0
GroupPolicy                         Manifest                                 1.0.0.0
PSDiagnostics                       Manifest                                 1.0.0.0
TroubleshootingPack                 Manifest Microsoft Windows Troubles...   1.0.0.0
WebAdministration                   Manifest                                 1.0.0.0
```

Abbildung 25.2 *Liste der Module auf Windows Server 2008 R2 (abhängig von den gewählten Serverrollen und Serverfeatures)*

`Get-Module` liefert eine Liste der aktuell importierten Module.

Zum Importieren eines Moduls nutzt man `import-module` gefolgt von dem Modulnamen (aus der Liste der installierten Module), z.B.:

```
Import-Module activedirectory
```

Danach hat sich die Anzahl der Commandlets von 236 auf 315 (Zahl abhängig von den installierten Rollen/Features) erhöht. Eine Liste der neuen Commandlets kann man auf einfache Weise erhalten, da der Typ `System.Management.Automation.CmdletInfo`, den `Get-Command` liefert, in PowerShell 2.0 ein neues Attribut `ModuleName` besitzt:

```
(Get-Command) | where { $_.ModuleName -eq "ActiveDirectory" } | ft
name, modulename, pssnapin
```

`Get-Command` hat auch ein Filterattribut für Module, so dass man alternativ schreiben kann:

```
Get-Command -module ServerManager | ft name, modulename, pssnapin
```

Abbildung 25.3 *Eine Auswahl der Commandlets aus dem Active-Directory-Modul im Windows Server 2008 R2*

```
PS C:\Users\hs> (get-command) | where { $_.ModuleName -eq "ActiveDirectory" } | ft name, modulename, pssnapin

Name                                ModuleName                          PSSnapIn
----                                ----------                          --------
Add-ADComputerServiceAccount        activeDirectory
Add-ADDomainControllerPasswordReplic... activeDirectory
Add-ADFineGrainedPasswordPolicySubject activeDirectory
Add-ADGroupMember                   activeDirectory
Add-ADPrincipalGroupMembership      activeDirectory
Clear-ADAccountExpiration           activeDirectory
Disable-ADAccount                   activeDirectory
Disable-ADOptionalFeature           activeDirectory
Enable-ADAccount                    activeDirectory
Enable-ADOptionalFeature            activeDirectory
Get-ADAccountAuthorizationGroup     activeDirectory
Get-ADAccountResultantPasswordReplic... activeDirectory
Get-ADComputer                      activeDirectory
Get-ADComputerServiceAccount        activeDirectory
Get-ADDefaultDomainPasswordPolicy   activeDirectory
Get-ADDomain                        activeDirectory
Get-ADDomainController              activeDirectory
Get-ADDomainControllerPasswordReplic... activeDirectory
Get-ADDomainControllerPasswordReplic... activeDirectory
Get-ADFineGrainedPasswordPolicy     activeDirectory
Get-ADFineGrainedPasswordPolicySubject activeDirectory
Get-ADForest                        activeDirectory
Get-ADGroup                         activeDirectory
Get-ADGroupMember                   activeDirectory
Get-ADObject                        activeDirectory
Get-ADOptionalFeature               activeDirectory
Get-ADOrganizationalUnit            activeDirectory
Get-ADPrincipalGroupMembership      activeDirectory
```

Abbildung 25.4 *Das Modul „ServerManager“ umfasst nur drei Commandlets.*

```
PS C:\Users\hs> import-module servermanager
PS C:\Users\hs> (get-command) | where { $_.ModuleName -eq "servermanager" } | ft name, modulename, pssnapin

Name                                ModuleName                          PSSnapIn
----                                ----------                          --------
Add-WindowsFeature                  servermanager
Get-WindowsFeature                  servermanager
Remove-WindowsFeature               servermanager
```



Durch den Zusatz `-verbose` erhält man eine genaue Liste der Auswirkungen des Moduls.

Abbildung 25.5 *Parameter –verbose für Import-Module*

```
PS IIS:\Sites> import-module servermanager -verbose
VERBOSE: Loading module from path
'C:\Windows\system32\WindowsPowerShell\v1.0\Modules\servermanager\servermanager.psd1'.
VERBOSE: Loading 'FormatsToProcess' from path
'C:\Windows\system32\WindowsPowerShell\v1.0\Modules\servermanager\Feature.format.ps1xml'.
VERBOSE: Importing cmdlet 'Get-WindowsFeature'.
VERBOSE: Importing cmdlet 'Add-WindowsFeature'.
VERBOSE: Importing cmdlet 'Remove-WindowsFeature'.
VERBOSE: Exporting cmdlet 'Get-WindowsFeature'.
VERBOSE: Exporting cmdlet 'Add-WindowsFeature'.
VERBOSE: Exporting cmdlet 'Remove-WindowsFeature'.
VERBOSE: Importing cmdlet 'Add-WindowsFeature'.
VERBOSE: Importing cmdlet 'Get-WindowsFeature'.
VERBOSE: Importing cmdlet 'Remove-WindowsFeature'.
PS IIS:\Sites> _
```



Direktimport

Ein Modul, das sich nicht in einem der Standardverzeichnisse befindet, muss man unter Angabe des ganzen Pfads importieren, z.B.:

```
Import-Module c:\ITVModule\Basismodul
```

Einen Modulimport kann man auch beim Start der PowerShell an der Kommandozeile angeben, z.B.:

```
PowerShell.exe -noexit -command import-module ActiveDirectory
```

Man kann auch anordnen, alle Systemmodule zu laden:

```
PowerShell.exe -NoExit -ImportSystemModules
```

In Windows Server 2008 R2 gibt es in der Benutzerschnittstelle schon vorgefertigte Verknüpfungen im Startmenü, z.B. „Active Directory Module for Windows PowerShell“ (lädt das Active Directory-Modul) und „Windows PowerShell Modules“ (lädt alle Systemmodule). Letzterer Befehl ist auch über die Kontextmenüs in der neuen Taskleiste von Windows 7 und Windows Server 2008 R2 verfügbar.



Abbildung 25.6
Befehle der Kontextmenüs in der Taskleiste in Windows Server 2008 R2



Da die Namen für Commandlets nicht weltweit eindeutig vergeben werden, kann es doppelte Namen in zwei Modulen geben. PowerShell 2.0 verwendet dann immer das zuletzt importierte Commandlet. PowerShell 1.0 gab beim Import von Snap-Ins in dieser Situation einen Fehler aus. In PowerShell 2.0 ist auch dieses Verhalten geändert auf die Verwendung des zuletzt importierten Commandlets.



Zu beachten ist, dass einige der von Microsoft gelieferten Module auf 64-Bit-Systemen nur im 64-Bit-Modus der PowerShell verfügbar sind. Die nachfolgende Bildschirmabbildung beweist, dass einige der Module in der 32-Bit-PowerShell nicht verfügbar sind.

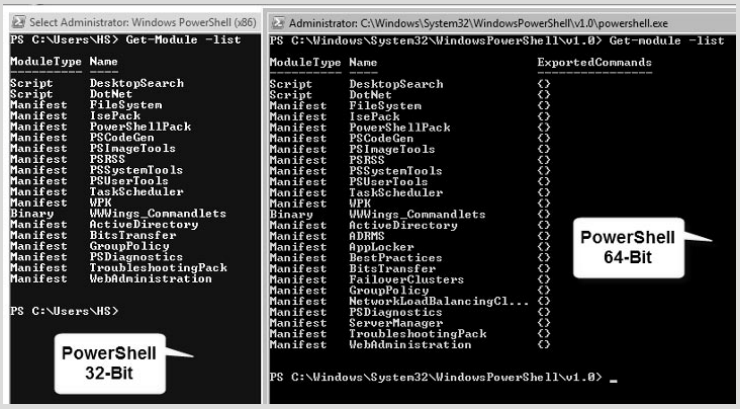


***Abbildung 25.7**: Verfügbare Module auf einem Windows Server 2008 R2 mit installiertem PowerShellPack und WWWings_Commandlets (links: 32-Bit-PowerShell, rechts: 64-Bit-PowerShell auf dem gleichen System)*

25.5 Entfernen von Modulen

Zum Entfernen eines Moduls aus der aktuellen PowerShell-Sitzung kann man das Modul mit `remove-module` entfernen, z.B.:

Remove-Module

```
Remove-Module activedirectory
```



Hinweise zum Erstellen von eigenen Modulen finden Sie in *Teil 4 „Profiwissen – Erweitern der PowerShell“*.

26 Ausgewählte PowerShell-Erweiterungen

Einige wichtige am Markt verfügbare kostenlose und kostenpflichtige Commandlet-Erweiterungen sind:

1. PowerShell-Module in Windows 7 und Windows Server 2008 R2
2. PowerShell Community Extensions von Microsoft
3. PowerShell Extensions von *www.IT-Visions.de*
4. Quest bietet Commandlets zum Active Directory Scripting.
5. Gruppenrichtlinienverwaltung mit der PowerShell ermöglicht die Firma FullArmor.
6. Commandlets zum Netzwerkmanagement mit der PowerShell bietet die Firma /n Software.
7. Die Firma PowerGadget bietet unter dem gleichen Namen eine Sammlung von zusätzlichen Commandlets zur Visualisierung des Inhalts der PowerShell-Pipeline.

26.1 PowerShell-Module in Windows 7 und Windows Server 2008 R2

Enthaltene Module

Windows 7 und Windows Server 2008 Release 2 enthalten Erweiterungsmodule für die PowerShell, die Sammlungen von Commandlets für verschiedene Funktionsbereiche darstellen (vgl. allgemeine Einführung in PowerShell-Module in diesem Buch). In Windows Server 2008 Release 2 sind die Module zum Teil erst nach dem Installieren der entsprechenden Serverrollen bzw. Serverfeatures verfügbar.

PowerShell-Modul	Windows 7	Windows Server 2008 R2	In diesem Buch beschrieben in Kapitel …
Active Directory	Durch Installation der Microsoft Remote Server Administration Tools (RSAT)	X	Verzeichnisdienste
ADRMS		X	
AppLocker	X	X	Softwareverwaltung
BestPractices		X	Optimierungen und Problemlösungen
BitsTransfer	X	X	
FailoverClusters		X	
GroupPolicy	Durch Installation der Microsoft Remote Server Administration Tools (RSAT)	X	Gruppenrichtlinien
NetworkLoadBalancingClusters		X	
PSDiagnostics	X	X	
RemoteDesktopServices		X	
ServerManager		X	Softwareverwaltung
TroubleshootingPack	X	X	Optimierungen und Problemlösungen
WebAdministration	X (nach Installieren des IIS 7.0 als optionales Windows-Feature)	X	Internet Information Services

***Tabelle 26.1**: PowerShell-Module in Windows 7 und Windows Server 2008 R2*



Die Aufarbeitung dieser Module für dieses Buch wurde durch die Tatsache, dass es bis zum Redaktionsschluss noch keine hinreichende Dokumentation von Microsoft gab, erheblich erschwert. Es gibt nur zu wenigen Modulen Überblicksdokumente („about_...“). Bei den meisten Modulen musste man mühselig die einzelnen Hilfedokumente zu den einzelnen Commandlets durcharbeiten. Eine TechNet-Dokumentation gab es noch nicht. Die Arbeit in diesem Buch basiert also zum Großteil auf mühseligen Selbsterforschungsarbeiten.



***Abbildung 26.1**: Die Dokumentation zu den im TechNet angepriesenen Modulen war bis zum Redaktionsschluss immer noch auf dem Stand der Beta-Phase.*

26.2 BITS-Transfer

Background Intelligent Transfer Service

Das Modul „BITSTransfer“ in Windows 7 und Windows Server 2008 R2 unterstützt mit acht Commandlets das Herunterladen von Dateien über den Background Intelligent Transfer Service-(BITS-)Dienst, den es seit Windows XP gibt.

Hinweis: BITS transferiert Daten über HTTP und HTTPS in Zeiten, in denen es freie Netzwerkleitungskapazitäten gibt. Die Übertragung erfolgt asynchron, d.h. ohne Blockierung der den Download initiierenden Anwendung.

Die Commandlets in diesem Modul sind:

- `Add-BitsFile`
- `Complete-BitsTransfer`
- `Get-BitsTransfer`
- `Remove-BitsTransfer`
- `Resume-BitsTransfer`
- `Set-BitsTransfer`
- `Start-BitsTransfer`
- `Suspend-BitsTransfer`

26.3 Windows Server Backup

Windows.Serverbackup

Die Unterstützung für Windows Server Backup ist nicht in Form eines Moduls, sondern eines Snap-In „Windows.Serverbackup“ in Windows Server 2008 R2 enthalten. Dort gibt es folgende Commandlets:

- `Add-WBBackupTarget`
- `Add-WBVolume`
- `Get-WBBackupTarget`
- `Get-WBDisk`
- `Get-WBPolicy`
- `Get-WBSchedule`
- `Get-WBSummary`
- `Get-WBVolume`
- `New-WBBackupTarget`
- `New-WBPolicy`
- `Remove-WBBackupTarget`
- `Remove-WBPolicy`
- `Remove-WBVolume`
- `Set-WBPolicy`
- `Set-WBSchedule`

Das Snap-In aktiviert man mit dem Befehl:

```
Add-Pssnapin Windows.serverbackup
```

Hinweis: In dem Commandlet werden die Backup-Aufträge kurioserweise als „Policy“ bezeichnet. `WBPolicy` listet also die geplanten Aufträge.

26.4 PowerShell Diagnostics

Das Modul „PSDiagnostics“ in Windows 7 und Windows Server 2008 R2 enthält Commandlets zur Ablaufverfolgung der internen Vorgänge der PowerShell.

- `Disable-PSTrace`
- `Disable-PSWSManCombinedTrace`
- `Disable-WSManTrace`
- `Enable-PSTrace`
- `Enable-PSWSManCombinedTrace`
- `Enable-WSManTrace`
- `Get-LogProperties`

- `Set-LogProperties`
- `Start-Trace`
- `Stop-Trace`

Leider war zu diesen Commandlets in dem zum Redaktionsschluss vorliegenden RC1 von Windows 7 und Windows Server 2008 R2 keine Dokumentation verfügbar, auch nicht mit `Get-Help` (Beweis: siehe nachstehende Abbildung).



Abbildung 26.2 *Absolut keine Hilfe verfügbar für die PSDiagnostics-Commandlets*

26.5 Windows PowerShell Community Extensions

Die PowerShell Community Extensions (PSCX) mit zusätzlichen Commandlets und Providern gibt es schon seit PowerShell 1.0. Zum Redaktionsschluss dieses Buchs gibt es die Version 1.2, die aber laut Dokumentation die neuen Funktionen der PowerShell 2.0 erst auf „Alpha-Level" (d.h. also nur rudimentär) unterstützt.

IT-Visions.de PowerShell Extensions	
Hersteller:	Microsoft/Open-Source-Community-Projekt
Preis:	Kostenlos
URL:	*http://www.codeplex.com/PowerShellCX*
Enthalten auf Buch-CD?	Ja, Version 1.2

PSCX 1.2 enthalten folgende Commandlets:

- `Add-PathVariable`
- `Clear-MSMQueue`
- `ConvertFrom-Base64`
- `ConvertTo-Base64`
- `ConvertTo-MacOs9LineEnding`
- `ConvertTo-UnixLineEnding`
- `ConvertTo-WindowsLineEnding`
- `Convert-Xml`
- `Disconnect-TerminalSession`
- `Expand-Archive`
- `Export-Bitmap`
- `Format-Byte`
- `Format-Hex`
- `Format-Xml`

- Get-ADObject
- Get-AdoConnection
- Get-AdoDataProvider
- Get-Clipboard
- Get-DhcpServer
- Get-DomainController
- Get-DriveInfo
- Get-EnvironmentBlock
- Get-FileVersionInfo
- Get-ForegroundWindow
- Get-Hash
- Get-HttpResource
- Get-MountPoint
- Get-MSMQueue
- Get-OpticalDriveInfo
- Get-PathVariable
- Get-PEHeader
- Get-Privilege
- Get-PSSnapinHelp
- Get-Random
- Get-ReparsePoint
- Get-ShortPath
- Get-TabExpansion
- Get-TerminalSession
- Import-Bitmap
- Invoke-AdoCommand
- Invoke-Apartment
- Join-String
- New-GenericObject
- New-Hardlink
- New-Junction
- New-MSMQueue
- New-Shortcut
- New-Symlink
- Out-Clipboard
- Ping-Host
- Pop-EnvironmentBlock
- Push-EnvironmentBlock
- Read-Archive
- Receive-MSMQMessage
- Remove-MountPoint
- Remove-ReparsePoint

- Resize-Bitmap
- Resolve-Assembly
- Resolve-Host
- Scroll-Table
- Scroll-Text
- Select-Xml
- Send-MSMQMessage
- Send-SmtpMail
- Set-Clipboard
- Set-FileTime
- Set-ForegroundWindow
- Set-PathVariable
- Set-Privilege
- Set-VolumeLabel
- Skip-Object
- Split-String
- Start-Process
- Start-TabExpansion
- Stop-TerminalSession
- Tail-File
- Test-Assembly
- Test-MSMQueue
- Test-Scroll
- Test-UserGroupMembership
- Test-Xml
- Write-BZip2
- Write-Clipboard
- Write-GZip
- Write-Tar
- Write-Zip

In PSCX 1.2 wurden ergänzt:

- Arbeit mit Archivdateien: Expand-Archive, Read-Archive
- Dateisystemoperationen: Get-DriveInfo, Tail-File, Get-DiskUsage, Get-OpticalDriveInfo
- Datenbankzugriffe: Invoke-AdoCommand, Get-AdoConnection, Get-AdoDataProvider
- Unterstützung für Microsoft Message Queing (MSMQ): Receive-MSMQMessage, Send-MSMQMessage, Test-MSMQueue, Clear-MSMQueue, Get-MSMQueue, New-MSMQueue
- Unterstützung für die Umgebungsvariable „Path“: Get-PathVariable, Add-PathVariable, Set-PathVariable

- Netzwerkfunktionen: `Get-HttpResource`
- Allgemeine Pipeline-Funktionen: `Skip-Object` (Überspringen der ersten oder letzten n Objekte in der Pipeline)
- Benutzerverwaltung: `Test-UserGroupMembership`
- Weitere Commandlets:

```
Invoke-Apartment
Push-EnvironmentBlock
Pop-EnvironmentBlock
New-GenericObject
Test-Scroll
Scroll-Table
Scroll-Text
```

Installation der PSCX

Die PowerShell Community Extensions besitzen eine eigene Installationsroutine. Während der Installation wird man gefragt, ob man eine Profildatei anlegen möchte, die das PSCX-Snap-In einbindet und zahlreiche Variablen und Funktionen anlegt. Wenn man dies nicht möchte (weil man bereits eine eigene Profildatei besitzt), muss man die PowerShell Community Extensions durch folgende Anweisung manuell in die eigene Profildatei einbinden oder bei jedem Start der Konsole ausführen:

```
Add-PSSnapin PSCX
```

26.6 PowerShellPack

Das PowerShellPack ist eine massive Erweiterung zur PowerShell 2.0, die Microsoft im Rahmen des Windows 7 Ressource Kit und als freien Download unter *http://code.msdn.microsoft.com/PowerShellPack* anbietet.

PowerShellPack	
Hersteller:	Microsoft
Preis:	Kostenlos
URL:	http://code.msdn.microsoft.com/PowerShellPack
Enthalten auf Buch-CD?	Ja, Version vom 16.10.2009

PowerShellPack beinhaltet rund 800 neue Befehle, aufgeteilt in 10 Module.

- Modul „WPK“: Das größte Modul im PowerShellPack ermöglicht die Erstellung von grafischen Benutzeroberflächen mit der Windows Presentation Foundation (WPF). WPK steht für WPF PowerShell Kit.
- IsePack: Mehr als 35 Erweiterungen für die ISE, hauptsächlich Shortcuts
- TaskScheduler: geplante Vorgänge einrichten und löschen
- FileSystem: Handhabung von ZIP-Dateien (z.B. `New-Zip`, `Copy-ToZip`), Laufwerken (`Rename-Drive`), Dateisystemüberwachung (`Start-FileSystemWatcher`) und Suche nach doppelten Dateien (`Get-DuplicateFile`)
- DotNet: Zugriff auf die geladenen Typen, herausfinden, welche Befehle auf die Typen angewendet werden können, sowie erarbeiten, wie PowerShell, DotNet und COM zusammenarbeiten können
- PSImageTools: Umwandeln, Drehen, Beschneiden bzw. Vergrößern oder Verkleinern von Bildern sowie Zugriff auf die Metadaten
- PSRSS: Befehle, um mit dem FeedStore für RSS-Feeds zu arbeiten
- PSSystemTools: Auslesen von Betriebssystem- und Hardwareinformationen
- PSUserTools: Zugriff auf die Benutzerkonten des lokalen Systems, Prüfung auf „elevation“ und Starten von Prozessen als Administrator
- PSCodeGen: Erzeugen von PowerShell-Skripten sowie C#-Code und PInvoke-Aufrufe



Diese Module kann man einzeln importieren mit Import-Module oder man kann alle Module zusammen importieren mit Import-Module PowerShellPack.

Zahlreiche der PowerShellPack-Commandlets werden in *Teil 3 „PowerShell im Praxiseinsatz“* besprochen.



Die Module des PowerShellPack sind als Skript-Module implementiert, d.h., man kann sich die Implementierung ansehen und davon lernen. Das PowerShellPack installiert sich in das benutzerspezifische Modulverzeichnis.

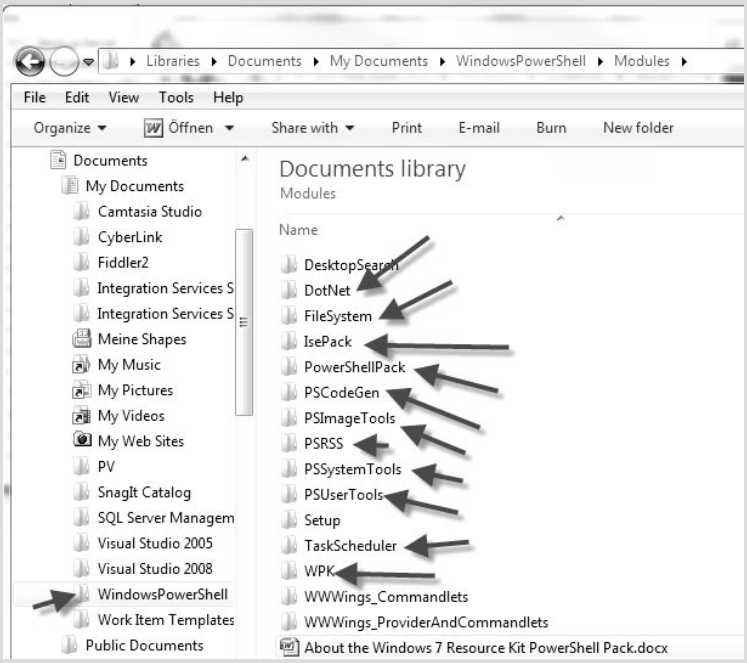


***Abbildung 26.3**: Module des PowerShellPack im benutzerspezifischen Modulverzeichnis in Windows 7*



Wenn Sie beim Importieren eines PowerShellPack-Moduls oder bei einzelnen Commandlets dieser Erweiterung auf Fehlermeldungen stoßen, könnte das daran liegen, dass Sie die PowerShell Community Extensions ebenfalls geladen haben. Leider sind diese beiden Erweiterungen nicht vollkompatibel zueinander, weil sie gleichnamige Commandlets definieren.

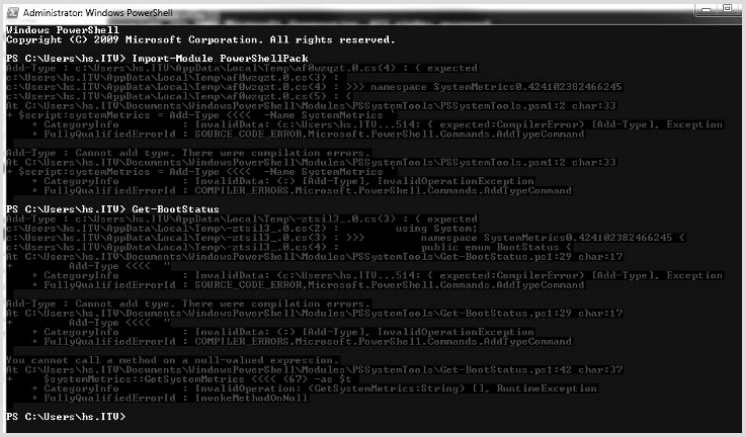


***Abbildung 26.4**: Diese Fehler basieren auf der Inkompatibilität zu PSCX*

26.7 www.IT-Visions.de PowerShell Extensions

Die PowerShell-Erweiterungen, die von der Firma des Buchautors kostenlos bereitgestellt werden, bieten Funktionen aus den Bereichen

- Verzeichnisdienstverwaltung (`Get-DirectoryEntry`, `Get-DirectoryChildren`, `Add-DirectoryEntry`, `Remove-DirectoryEntry` ..)
- Hardwareinformationen (`Get-Processor`, `Get-Memorydevice`, `Get-NetworkAdapter`, `Get-CDRomDrive`, `Get-Videocontroller`, `Get-USBController` ..)
- Datenbankzugriff (`Get-DBTable`, `Get-DBRow`, `Set-DBTable`, `Invoke-DBCommand`, `Get-DBConnection`, ..)

www.IT-Visions.de PowerShell Extensions	
Hersteller:	www.IT-Visions.de
Preis:	Kostenlos
URL:	*http://www.dotnetframework.de/scripting/powershell/PowerShellCommandletExtensions.aspx*
Enthalten auf Buch-CD?	Ja, Version 2.0

Die Erweiterung muss manuell mit `installutil.exe` installiert werden:

```
installutil.exe ITVisions_PowerShell_Extensions.dll
```

Anschließend muss die Erweiterung in die Konsole geladen werden (dies am besten in Profil.ps1 eintragen):

```
Add-PSSnapin ITVisions_PowerShell_Extensions
```

26.8 Quest Management Shell for Active Directory

Die Firma Quest stellt Commandlets zur Active Directory-Verwaltung sowie eine angepasste PowerShell-Konsole (Quest Management Shell for Active Directory) zur Verfügung.

Quest Management Shell for Active Directory	
Hersteller:	Quest
Preis:	Kostenlos
URL:	*http://www.quest.com/activeroles-server/arms.aspx*
Enthalten auf Buch-CD?	Ja, Beta-Version

Die Quest Commandlets können auch unabhängig von der Quest-Management-Konsole in die normale PowerShell durch den Befehl `Add-Pssnapin quest.activeroles.admanagement` eingebunden werden.



Abbildung 26.5 *Quest Management Shell for Active Directory*

Die Quest-Erweiterungen enthalten in der aktuellen Version folgende Commandlets:

- `Add-QADGroupMember`
- `Connect-QADService`
- `Disconnect-QADService`
- `Get-QADComputer`
- `Get-QADGroup`
- `Get-QADGroupMember`
- `Get-QADObject`
- `Get-QADUser`
- `New-QADGroup`
- `New-QADObject`
- `New-QADUser`
- `Remove-QADGroupMember`
- `Set-QADObject`
- `Set-QADUser`

26.9 Microsoft Exchange Server 2007

Der Microsoft Exchange Server 2007 war das erste Microsoft-Produkt, das die PowerShell zur Administration einsetzt. Die mit dem Exchange Server mitgelieferte Exchange Management Shell (eine angepasste Version der PowerShell) sowie zahlreiche zugehörige Commandlets ermöglichen es, alle administrativen Aufgaben des Exchange Servers von der Kommandozeile auszuführen.

U.a. folgende Commandlets werden dort bereitgestellt:

- `Get-ExchangeServer`
- `Enable-Mailcontact`
- `Enable-Mailbox`
- `Disable-Mailbox`
- `Get-Mailbox`
- `Get-MailboxStatistics`
- `New-SystemMessage`
- `Get-Recipient`
- `Get-UMMailbox`
- `New-MailboxDatabase`
- `New-StorageGroup`
- `New-SendConnector`

- `Suspend-Queue`
- `Resume-Queue`
- `Set-RecipientFilterConfig`
- `New-JournalRule`

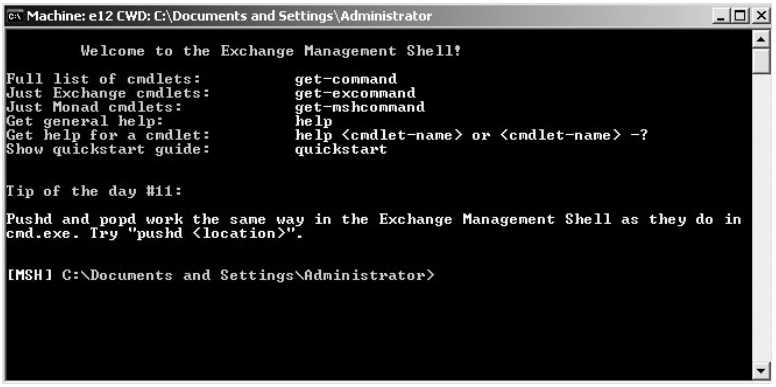


Abbildung 26.6
Exchange Server 2007 Management Shell



Weitere Informationen finden Sie unter [TNET01] und [TNET02].

26.10 System Center Virtual Machine Manager 2007

System Center Virtual Machine Manager (SCVMM) 2007 ist ein Verwaltungswerkzeug für virtuelle Systeme auf Basis von Virtual Server. Auch diese Verwaltungskonsole basiert im Hintergrund komplett auf PowerShell-Commandlets, so dass alle Aktionen des SCVMM auch per Skript ausgeführt werden können.

U.a. werden dort folgende Commandlets bereitgestellt:

- `New-VirtualNetworkAdapter`
- `New-VirtualDVDDrive`
- `New-HardwareProfile`
- `Get-VirtualHardDisk`
- `Add-VirtualHardDisk`
- `New-VM`
- `Get-VMHost`
- `Get-FloppyDrive`
- `Get-DVDDrive`
- u.a.

26.11 PowerShell Management Library for Hyper-V (pshyperv)

Diese bei Codeplex.com verfügbare Commandlets-Bibliothek bietet Commandlets zur Verwaltung von Virtuellen Maschinen (VM) im Virtualisierungsdienst Hyper-V in Windows Server 2008.

PowerShell Management Library for Hyper-V (pshyperv)	
Hersteller:	Open Source-Projekt
Preis:	Kostenfrei
URL:	*http://www.primalscript.com/*
Enthalten auf Buch-CD?	Ja

Die folgende Liste zeigt die verfügbaren Commandlets (Quelle: *http://pshyperv.codeplex.com/*)

- Auffinden einer Virtuellen Maschine
  - Get-VM, Choose-VM, Get-VMHost
- Zu einer Virtuellen Maschine verbinden
  - New-VMConnectSession
- Verändern einer Virtuellen Maschine
  - Get-VMState, Set-VMState, Convert-VmState,
  - Ping-VM , Test-VMHeartBeat, Shutdown-VM , Start-VM, Stop-VM, Suspend-VM
  - Get-VMKVP, Add-KVP, Remove-KVP, Get-VMJPEG
- Sichern und Exportieren einer Virtuellen Maschine, Erstellen von Momentaufnahmen (Snapshots)
  - Export-VM, Import-VM, Get-VMSnapshot, Choose-VMSnapshot, Apply-VMSnapshot, New-VMSnapshot, Remove-VMSnapshot, Rename-VMSnapShot, Update-VMSnapshot, Get-VMSnapshotTree, Get-VmBackupScript
- Virtuelle Maschine erstellen und löschen
  - New-VM, Remove-VM, Set-VM, Get-VMCPUCount, Set-VMCPUCount, Get-VMMemory, Set-VMMemory, Set-VMSerialPort
- Laufwerk erstellen
  - Get-VMDiskController
  - Add-VMSCSIController, Remove-VMSCSIcontroller
  - Get-VMDriveByController, Add-VMDRIVE, Remove-VMdrive
  - Get-VMDiskByDrive, Add-VMDISK, Set-VMDisk, Get-VMDisk
  - Get-VMFloppyDisk, Add-VMFloppyDisk
  - Add-VMNewHardDisk

- Network Interface Cards erstellen
  - Get-VMNic, List-VMNic, Choose-VMNIC, Add-VMNIC, Remove-VMNIC, Set-VMNICAddress, Set-VMNICConnection, Get-VMNicport
  - Get-VMnicSwitch, Choose-VMSwitch, New-VMSwitchPort, Get-VMByMACaddress, Choose-VMExternalEthernet
  - New-VMExternalSwitch, New-VMInternalSwitch, New-Vm PrivateSwitch
- Mit VHD-Dateien arbeiten
  - Get-VHDDefaultPath, Get-VHDInfo, New-VHD, Compact-VHD, Test-VHD, Convert-VHD, Merge-VHD, Mount-VHD, Unmount-VHD

26.12 Powershell Outlook Account Manager

Diese Erweiterung bietet drei Commandlets zur Verwaltung von E-Mail-Konten in Microsoft Outlook: `Get-MAPIProfile`, `Get-MAPIAccount`, `Set-MAPIAccount`. Gleichzeitig ist diese Erweiterung auch eine Klassenbibliothek für .NET (OutlookAccountManager.dll), die die Account Management API als .NET-Komponenten verpackt.

Powershell Outlook Account Manager	
Hersteller:	Open Source-Projekt
Preis:	Kostenfrei
URL:	*http://psoutlookmanager.codeplex.com/*
Enthalten auf Buch-CD?	Ja

26.13 PowerShell Configurator (PSConfig)

PSConfig bietet eine Sammlung von Commandlets in Form eines PowerShell-Moduls für verschiedene Konfigurationsaufgaben in Windows. Die Bibliothek ist laut der Website insbesondere gedacht für die Verwaltung von Windows Server 2008 R2 Core-Installationen. Die Commandlets sind aber auch nützlich auf vollständigen Windows-Server-Systemen.

- Software, Updates und Treiber
  - Add-Driver, Get-Driver
  - Add-HotFix
  - Add-InstalledProduct, Get-InstalledProduct, Remove-InstalledProduct
  - Add-WindowsFeature, Get-WindowsFeature, Remove-WindowsFeature
  - Add-WindowsUpdate, Get-WindowsUpdateConfig, Set-WindowsUpdateConfig
- Windows Firewall
  - Get-FirewallConfig, Set-FirewallConfig, Get-FirewallProfile, Get-FireWallRule
- Netzwerkkonfiguration
  - Get-NetworkAdapter, Get-IpConfig, New-IpConfig, Remove-IpConfig, Set-IpConfig
- Lizensierung/Aktivierung
  - Get-Registration, Register-Computer
- Auslagerungsdatei (Page File)
  - Get-ShutDownTracker, Set-ShutDownTracker
- Remote Desktop
  - Get-RemoteDesktopConfig, Set-RemoteDesktop
- Sonstige
  - Get-WinRMConfig
  - Rename-Computer
  - Set-iSCSIConfig
  - Set-RegionalConfig
  - Out-Tree
  - Test-Admin
  - Get-FirstAvailableDriveLetter

PowerShell Configurator (PSConfig)	
Hersteller:	Microsoft/James O'Neill
Preis:	Kostenfrei
URL:	*http://psconfig.codeplex.com/*
Enthalten auf Buch-CD?	Ja

26.14 Weitere Erweiterungen

Die Tabelle listet einige weitere Commandlet-Bibliotheken auf.

Tabelle 26.2 *Commandlet-Erweiterungen*

Name	URL	Auf der Buch-CD
PowerShell Provider BizTalk	*http://psbiztalk.codeplex.com/*	Ja
PowerShell Cmdlets for Visual Studio Team System	*http://pscmdlets4teamsystem.codeplex.com/*	Ja
SharePoint PowerShell Module (SPoshMod)	*http://sposhmod.codeplex.com/*	Ja
PowerShell VMWare Toolkit	*http://vmware.com/go/powershell*	Nein

27 Tipps und Tricks zur PowerShell

27.1 Befehlsgeschichte

Get-History

Die PowerShell-Konsole speichert im Standard die letzten 64 eingegebenen Befehle in einer Befehlsgeschichte (History). Diese lassen sich mit `Get-History` auflisten.

Durch den Parameter `Count` kann man eine bestimmte Anzahl von Befehlen ansehen (jeweils die letzten n Befehlen werden gezeigt).

```
Get-History -count 10
```

Invoke-History

Einen Befehl aus der Befehlsgeschichte kann man gezielt über die Position aufrufen:

```
Invoke-History 9
```

Die Anzahl der gespeicherten Befehle kann durch die Variable `$MaximumHistoryCount` erhöht werden.

Die Befehlsgeschichte kann man exportieren, entweder als Skriptdatei oder als XML-Datei. Eine Skriptdatei verwendet man, wenn man die eingegebenen Befehle wieder automatisch in der Reihenfolge der Eingabe ablaufen lassen will. Das XML-Dateiformat verwendet man, wenn man die Befehlsgeschichte einer früheren Sitzung wiederherstellen will, ohne die Befehle gleichzeitig auch alle auszuführen.

	Skriptdatei (.ps1)	XML-Format
Exportieren	`Get-History -Count 10 | format-table commandline -HideTable-Header | Out-file "c:\mein-Skript.ps1"`	`Get-History | Export-CliXml "b:\Skripte\geschichte.xml"`
Importieren (bzw. Ausführen)	`. "c:\meinSkript.ps1"`	`Import-CliXml "b:\Skripte\ geschichte.xml" | Add-History`

Mit `Clear-History` (ab PowerShell 2.0) kann man die Befehlsliste löschen.

27.2 System- und Hostinformationen

Get-Host

Das Commandlet `Get-Host` und die eingebaute Variable `$Host` liefern Informationen über die aktuelle PowerShell-Umgebung. Sowohl das Commandlet als auch die Variable liefern die gleiche Instanz der Klasse `System.Management.Automation.Internal.Host.InternalHost.InternalHost` enthält Informationen und erlaubt über sein Unterobjekt `UI.RawUI` auch Modifikationen, z.B.:

$Host

- `$Host.Name`: Name des Hosts (damit ist eine Unterscheidung der Umgebung möglich, z.B. liefert hier der PowerShellPlus Host einen anderen Wert als die normale PowerShell-Konsole)
- `$Host.Version`: Versionsnummer der Ablaufumgebung
- `$Host.UI.RawUI.WindowTitle = "Titel"`: Setzen der Titelzeile des Fensters
- `$Host.UI.RawUI.ForeGroundColor = [System.ConsoleColor]::White`: Setzen der Textfarbe
- `$Host.UI.RawUI.BackgroundColor = [System.ConsoleColor]::DarkBlue`: Setzen der Texthintergrundfarbe

Beispiel

Das folgende Beispiel erzeugt eine Kopfzeile, in der neben dem Namen des aktuellen Benutzers auch gezeigt wird, ob dieser ein Administrator ist oder nicht. Der Code eignet sich hervorragend zur Verwendung im Profilskript.

Listing 27.1 *Beispiel für ein Profilskript für eine aussagekräftige Titelzeile [Aufbauwissen/Profile/Profile_Title.ps1]*

```
# PowerShell Profile Script
# Holger Schwichtenberg 2007

# ------------- Window Title

$WI = [System.Security.Principal.WindowsIdentity]::GetCurrent()
$WP = New-Object System.Security.Principal.WindowsPrincipal($wi)
if
($WP.IsInRole([System.Security.Principal.WindowsBuiltInRole]::Admin
istrator))
{
 $Status = "[elevated user]"
}
else
{
 $Status = "[normal User]"
}

$Host.UI.RawUI.WindowTitle  = "PowerShell - " +
[System.Environment]::UserName  + " " + $Status
```

Get-Culture

`Get-Culture` (oder `$Host.CurrentCulture`) und `Get-UICulture` (oder `$Host.CurrentUICulture`) liefern die Informationen über die aktuelle Sprache auch einzeln in Form von Instanzen der .NET-Klasse `System.Globalization.CultureInfo`. `Get-Culture` bezieht sich auf die Ausgaben von Datum, Uhrzeit und Währungen (vgl. regionale Einstellungen der Windows-Systemsteuerung). `Get-UICulture` bezieht sich auf die Sprache der Benutzeroberfläche. In der Regel sind zwar beide Einstellungen gleich, ein Benutzer kann diese jedoch auch abweichend festlegen.

```
PS C:\Skripte> Get-Host


Name             : ConsoleHost
Version          : 2.0
InstanceId       : 9fef623f-0a96-45ca-90b3-48f9dd5c0457
UI               : System.Management.Automation.Internal.Host.InternalHostUserInterface
CurrentCulture   : de-DE
CurrentUICulture : de-DE
PrivateData      : Microsoft.PowerShell.ConsoleHost+ConsoleColorProxy
IsRunspacePushed : False
Runspace         : System.Management.Automation.Runspaces.LocalRunspace


PS C:\Skripte> _
```

Abbildung 27.1 *Ausführung von Get-Host*

27.3 Alle Anzeigen löschen

Clear-Host

`Clear-Host` (Alias `clear`) löscht die Anzeige in der PowerShell-Konsole, löscht aber nicht die Befehlsgeschichte.

27.4 Anpassen der Eingabeaufforderung (Prompt)

Prompt

Die Anzeige, mit der die PowerShell zur Eingabe von Befehlen auffordert, z.B. „PS C:\users\hs>“, wird als „Prompt“ bezeichnet. Der Aufbau des Prompts ist nicht in der PowerShell-Konsole starr festgelegt, sondern wird durch die eingebaute Funktion `Prompt` erzeugt. Diese Funktion kann man abändern, z.B. so, dass die aktuelle Uhrzeit als Eingabeaufforderung erscheint.

```
Function Prompt { [System.DateTime]::Now.ToShortTimeString()  +">"}
```

Die folgende Bildschirmabbildung zeigt die Standardimplementierung der Funktion `Prompt` und ihre Abänderungen.



Abbildung 27.2 *Änderung der Eingabeaufforderung*



Eine Funktion zum Zurücksetzen auf den alten Wert gibt es nicht. Innerhalb einer PowerShell-Sitzung bekommen Sie die normale Eingabeaufforderung nur dann wieder, wenn Sie sich die alte Implementierung in einer Variablen merken. Mit dem Ende der PowerShell-Sitzung (Schließen der Konsole) ist das Überschreiben der Prompt-Funktion aber wieder gelöscht. Wenn Sie die Eingabeaufforderung für alle PowerShell-Sitzungen ändern wollen, müssen Sie die eigene `Prompt`-Funktion in die Profildatei eintragen (siehe nächstes Kapitel).

27.5 Profileinstellungen für die PowerShell-Konsole

Profile.ps1

Beim Beenden der PowerShell-Konsole vergisst diese alle Einstellungen (z.B. geladene Snap-Ins, definierte Aliase, definierte Funktionen, eingebundene PowerShell-Provider und die Befehlsgeschichte). Mit Hilfe sogenannter Profildateien kann man der PowerShell-Konsole beim Start ihr Gedächtnis zurückgeben. Profile sind PowerShell-Skripte mit dem Namen „Profile“ und mit der Dateinamenserweiterung *.ps1*.

Eine *Profile.ps1* kann es auf zwei Ebenen geben:

- Global für alle Benutzer im PowerShell-Installationsordner (i.d.R. *C:\WINDOWS\system32\windowspowershell\v1.0*)
- Benutzerbezogen im Dateisystemverzeichnis (ab Windows Vista normalerweise in *c:\Benutzer\(Benutzername)\Dokumente\Windows PowerShell*, auf älteren Systemen unter *c:\Dokumente und Einstellungen\(Benutzername)\Eigene Dateien\WindowsPowerShell)*



Die PowerShell Extensions (PSCX) legen bei Ihrer Installation eine solche Profildatei mit zahlreichen Einstellungen (u.a. Ändern der Titelzeile und der Eingabeaufforderung) benutzerbezogen ab, siehe nachfolgendes Listing.

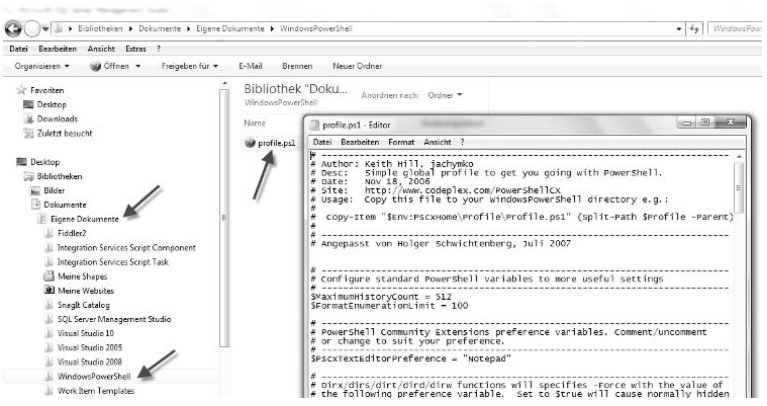


Abbildung 27.3 *Ablage der Profildatei (hier in Windows Vista)*

Listing 27.2
Leicht angepasste Version der Profildatei aus den PowerShell Extensions (PSCX) [Aufbauwissen/Profile/Profile.ps1]

```
# -------------------------------------------------------------------
----------
# Author: Keith Hill, jachymko
# Desc:   Simple global profile to get you going with PowerShell.
# Date:   Nov 18, 2006
# Site:   http://www.codeplex.com/PowerShellCX
# Usage:  Copy this file to your WindowsPowerShell directory e.g.:
#
#  Copy-Item "$Env:PscxHome\Profile\Profile.ps1" (Split-Path
$Profile -Parent)
#
# -------------------------------------------------------------------
----------
# Angepasst von Holger Schwichtenberg, Juli 2007

# -------------------------------------------------------------------
----------
# Configure standard PowerShell variables to more useful settings
# -------------------------------------------------------------------
----------
$MaximumHistoryCount = 512
$FormatEnumerationLimit = 100

# -------------------------------------------------------------------
----------
# PowerShell Community Extensions preference variables. Comment/
uncomment
# or change to suit your preference.
# -------------------------------------------------------------------
----------
$PscxTextEditorPreference = "Notepad"

# -------------------------------------------------------------------
----------
# Dirx/dirs/dirt/dird/dirw functions will specifies -Force with the
value of
# the following preference variable.  Set to $true will cause
normally hidden
# items to be returned.
# -------------------------------------------------------------------
----------
$PscxDirForcePreference = $true

# -------------------------------------------------------------------
----------
# Dirx/dirs/dirt/dird/dirw functions filter out files with system
attribute set.
# The performance may suffer on high latency networks or in folders
with
```

```
# many files.
# ------------------------------------------------------------------
----------
## $PscxDirHideSystemPreference = $true

# ------------------------------------------------------------------
----------
# Display file sizes in KB, MB, GB multiples.
# ------------------------------------------------------------------
----------
$PscxFileSizeInUnitsPreference = $false

# ------------------------------------------------------------------
----------
# The Send-SmtpMail default settings.
# ------------------------------------------------------------------
----------
## $PscxSmtpFromPreference = 'john_doe@example.net'
## $PscxSmtpHostPreference = 'smtp.example.net'
## $PscxSmtpPortPreference = 25

# ------------------------------------------------------------------
----------
# Uncomment this to create a transcript of the entire PowerShell
session.
# ------------------------------------------------------------------
----------
## $PscxTranscribeSessionPreference = $true

# ------------------------------------------------------------------
----------
# You can modify every aspect of the PSCX prompt appearance by
# creating your own eye-candy script.
# ------------------------------------------------------------------
----------
## $PscxEyeCandyScriptPreference = '.\EyeCandy.Jachym.ps1'
$PscxEyeCandyScriptPreference = '.\EyeCandy.Keith.ps1'

# ------------------------------------------------------------------
----------
# The following functions are used during processing of the PSCX
profile
# and are deleted at the end of loading this profile.
# !! Do not modify or remove the functions below !!
# ------------------------------------------------------------------
----------
function Set-PscxVariable($name, $value)
{
Set-Variable $name $value -Scope Global -Option AllScope,ReadOnly -
```

```
Description "PSCX variable"
}
function Set-PscxAlias($name, $value, $type = 'cmdlet',
[switch]$force)
{
Set-Alias $name $value -Scope Global -Option AllScope -Force:$force
-Description "PSCX $type alias"
}

function Test-PscxPreference($name)
{
if (Test-Path "Variable:$name")
{
(Get-Variable $name).Value
}
else
{
$false
}
}
# -------------------------------------------------------------------
----------
# !! Do not modify or remove the functions above !!
# -------------------------------------------------------------------
----------

if (!(Test-Path Variable:__PscxProfileRanOnce))
{
# -------------------------------------------------------------------
----------
# This should only be run once per PowerShell session
# -------------------------------------------------------------------
----------
Add-PSSnapin Pscx
Start-TabExpansion

# -------------------------------------------------------------------
----------
# Load ps1xml files which override built-in PowerShell defaults.
# -------------------------------------------------------------------
----------
Update-FormatData -PrependPath "$Env:PscxHome\FormatData\
FileSystem.ps1xml"
Update-FormatData -PrependPath "$Env:PscxHome\FormatData\
Reflection.ps1xml"

# -------------------------------------------------------------------
----------
# Create $UserProfile to point to the user's non-host specific
```

```
profile script
# -------------------------------------------------------------------
----------
Set-PscxVariable ProfileDir  (split-path
$MyInvocation.MyCommand.Path -Parent)
Set-PscxVariable UserProfile (join-path $ProfileDir 'Profile.ps1')

# -------------------------------------------------------------------
----------
# Create PSCX convenience variables, identity variables used by
EyeCandy.*.ps1
# -------------------------------------------------------------------
----------
Set-PscxVariable PscxHome    ($env:PscxHome)
Set-PscxVariable PscxVersion ([Version](Get-FileVersionInfo (Get-
PSSnapin Pscx).ModuleName).ProductVersion)
Set-PscxVariable Shell       (New-Object -com Shell.Application)
Set-PscxVariable NTIdentity
([Security.Principal.WindowsIdentity]::GetCurrent())
Set-PscxVariable NTAccount
($NTIdentity.User.Translate([Security.Principal.NTAccount]))
Set-PscxVariable NTPrincipal (New-Object
Security.Principal.WindowsPrincipal $NTIdentity)
Set-PscxVariable IsAdmin
($NTPrincipal.IsInRole([Security.Principal.WindowsBuiltInRole]::Adm
inistrator))
}
else
{
# -------------------------------------------------------------------
----------
# This should be run every time you want apply changes to your type
and format
# files.
# -------------------------------------------------------------------
----------
Update-FormatData
Update-TypeData
}

# -------------------------------------------------------------------
----------
# PowerShell Community Extensions utility functions and filters.
# Comment out or remove any dot sourced functionality that you
don't want.
# -------------------------------------------------------------------
----------
Push-Location (Join-Path $Env:PscxHome 'Profile')
. '.\TabExpansion.ps1'
```

```
. '.\GenericAliases.ps1'
. '.\GenericFilters.ps1'
. '.\GenericFunctions.ps1'
. '.\PscxAliases.ps1'
. '.\Debug.ps1'
. '.\Environment.VirtualServer.ps1'
. '.\Environment.VisualStudio2005.ps1'
. '.\Cd.ps1'
. '.\Dir.ps1'
. '.\TranscribeSession.ps1'
. $PscxEyeCandyScriptPreference
Pop-Location

# ------------------------------------------------------------------
----------
# Add PSCX Scripts dir to Path environment variable to allow
scripts to be executed.
# ------------------------------------------------------------------
----------
Add-PathVariable Path $env:PscxHome,$env:PscxHome\Scripts

# ------------------------------------------------------------------
----------
# Remove functions only required for the processing of the PSCX
profile.
# ------------------------------------------------------------------
----------
Remove-Item Function:Set-PscxAlias
Remove-Item Function:Set-PscxVariable
# ------------------------------------------------------------------
----------
# Keep track of whether or not this profile has ran already and
remove the
# temporary functions
# ------------------------------------------------------------------
----------
Set-Variable __PscxProfileRanOnce


# ------------------------------------------------------------------
----------
# Ergänzungen von Dr. Holger Schwichtenberg
# ------------------------------------------------------------------
----------

# Snap-Ins laden
Add-PSSnapin ITVisions_PowerShell_Extensions

# Title
```

```
$Wi = [System.Security.Principal.WindowsIdentity]::GetCurrent()
$wp = New-Object System.Security.Principal.WindowsPrincipal($wi)
if
($wp.IsInRole([System.Security.Principal.WindowsBuiltInRole]::Admin
istrator))
{
 $Status = "[elevated user]"
}
else
{
 $Status = "[normal User]"
}
$PscxWindowTitlePrefix  = "PowerShell - " +
[System.Environment]::UserName  + " " + $Status + " - "
```



Um die Profildateien übersichtlicher zu gestalten, darf man sie auf mehrere Dateien aufteilen und die einzelnen Dateien dann mit dem Punkt-Operator über das sogenannte „Dot Sourcing" aus dem Hauptskript (das immer *Profile.ps1* heißen muss) aufrufen.

27.6 Einblicke in die Interna der Pipeline-Verarbeitung

Trace-Command

Wenn Sie genauer wissen wollen, wie der Pipeline Processor der PowerShell arbeitet, weil etwas nicht so funktioniert, wie Sie es erwarten, oder weil Sie einfach „wissbegierig" sind, dann können Sie `Trace-Command` nutzen. Bei `Trace-Command` können Sie nach `-expression` eine Pipeline angeben und dann sehen Sie sehr genau (oft viel zu genau), was passiert. Zu beachten sind insbesondere die Stellen, wo eine Bindung von Parametern als erfolgreich „Successful" dokumentiert wird.

An dem folgenden Beispiel erkennt man, dass in der Pipeline

```
Get-Childitem c:\temp -filter *.txt | select -First 1| Get-Content
```

die Übergabe zwischen `Get-ChildItem` und `Select-First` über den Parameter „InputObject“ erfolgt, während `Get-Content` sich nur für die Eigenschaft „LiteralPath“ interessiert und den Rest des Objekts gar nicht erhält.

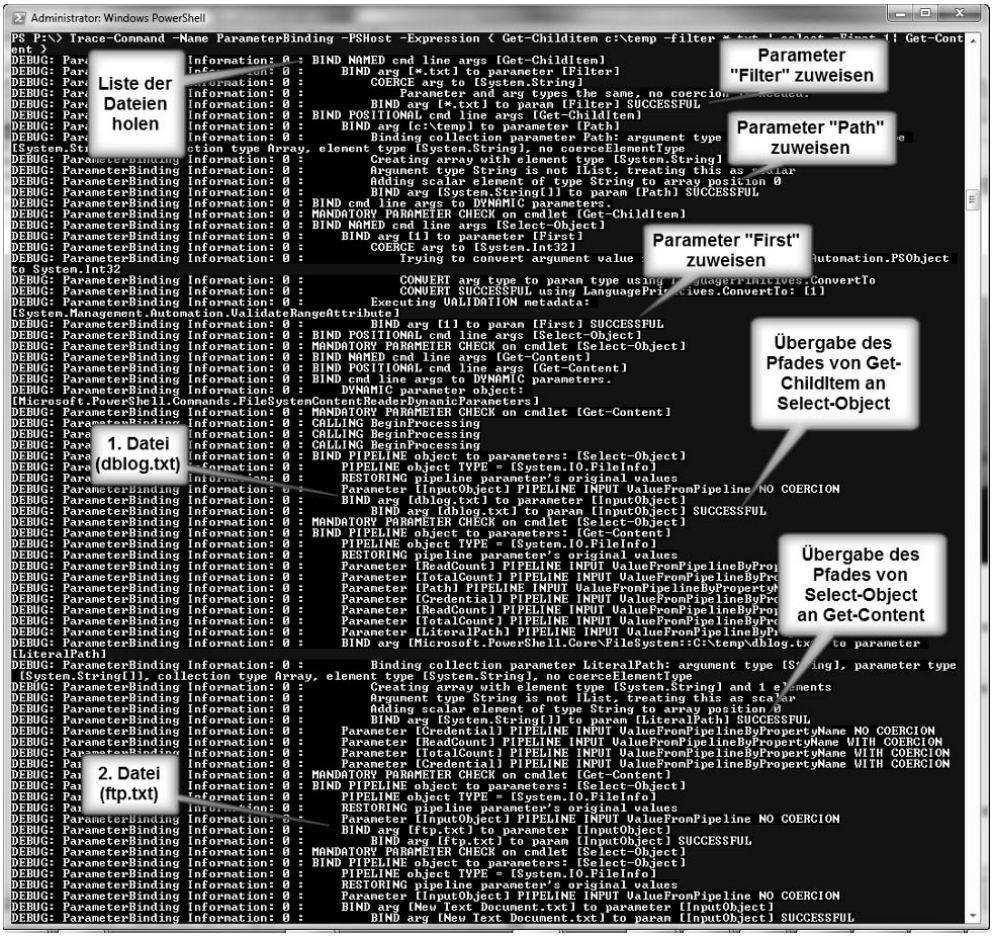


***Abbildung 27.4**: Beispiel für den Einsatz von TraceCommand*



Teil 3

PowerShell im Praxiseinsatz

Dieser Teil enthält zahlreiche praktische Anwendungsbeispiele für die Windows PowerShell.

28 Dateisystem

Das Dateisystem kann über das PowerShell-Navigationsmodell mit dem Provider „Filesystem“ angesprochen werden. Die folgende Tabelle zeigt wichtige Commandlets zur Arbeit mit dem Dateisystem sowie ähnliche Befehle in der klassischen Windows-Konsole und der Unix-Shell „sh“.

PowerShell-Commandlet	PowerShell-Alias	Befehl Klassische Windows-Kommandozeile	Befehl Unix „sh“	Beschreibung
Clear-Item	Cli	-	-	Inhalt leeren
Copy-Item	Cpi, cpp, cp, copy	Copy	Cp	Kopieren von Elementen
Get-Content	Gc	Type	Cat	Holt den Inhalt
Get-Location	Gl, pwd	Pwd	Pwd	Holt das aktuelle Verzeichnis
Move-Item	Mi, move, mv, mi	Move	Mv	Bewegen von Elementen
New-Item	Ni (Funktion md)	-	-	Element anlegen
Remove-Item	Ri, rp, rm, rmdir, del, erase, rd	del, rd	rm, rmdir	Löschen von Elementen
Rename-Item	Rni, ren	Rn	Ren	Umbenennen eines Elements
Set-Content	Sc	(Umleitungen >)	(Umleitungen >)	Festlegen des Inhalts
Set-Item	Si	-	-	Inhalt festlegen
Set-Location	Sl, cd, chdir	cd, chdir	cd, chdir	Setzt das aktuelle Verzeichnis

***Tabelle 28.1**: Wichtige Commandlets für die Arbeit mit dem Dateisystem*

28.1 Laufwerke

Dieses Kapitel behandelt den Umgang mit Dateisystemlaufwerken.

Auflisten der Laufwerke

Für den Zugriff auf die Dateisystemlaufwerke hat man fünf Möglichkeiten:

- Verwendung des Commandlets `Get-PSDrive` (Commandlet der PowerShell 1.0)
- Statische Methode `GetDrives()` der .NET-Klasse `System.IO.DriveInfo`
- Ermitteln der Instanzen der WMI-Klasse `Win32_LogicalDisk`
- Verwendung des Commandlets `Get-Disk` (Commandlet von *www.IT-Visions.de*)
- `Get-LogicalDiskInventory` (aus dem Modul „PSSystemTools", enthalten im PowerShellPack). Dieses Commandlet verwendet `Win32_LogicalDisk`, beschränkt sich aber auf den Laufwerkstyp 3 (lokale Laufwerke).

Get-PSDrive

Eine Liste der Dateisystemlaufwerke erhält man von `Get-PSDrive` durch Einschränkung auf den Provider „Filesystem":

```
Get-PSDrive -psprovider filesystem
```

Das Ergebnis sind Objekte des Typs `System.Management.Automation.PSDriveInfo`. Zu den Eigenschaften dieser Klasse gehört auch `Root`, welches das Wurzelverzeichnis zu jedem Laufwerk enthält.



Die PowerShell-Klasse `PSDriveInfo` enthält keine Informationen über Größe und Füllstand der Laufwerke, da es sich um ein generisches Konzept für alle Arten von Mengen handelt und solche Werte für einige Laufwerke (z.B. Umgebungsvariablen) keinen Sinn machen würden.

Abbildung 28.1
Einsatz der Methode GetDrives()

```
32# [System.IO.DriveInfo]::GetDrives()

VolumeLabel          Name          UsedSpace          FreeSpace   TotalSize %Free
-----------          ----          ---------          ---------   --------- -----
Local Disk           A:\                   0                                {} ... %
DATEN                B:\      41.798.037.504        142.630.912    39,06 GB   0 %
SYSTEM               C:\      35.511.558.144      6.429.110.272    39,06 GB  15 %
Local Disk           D:\                   0                                {} ... %
Buch                 G:\      33.978.740.736      7.961.927.680    39,06 GB  19 %
DATEN                H:\      41.798.037.504        142.630.912    39,06 GB   0 %
Install              I:\      93.895.868.416     10.959.966.208    97,655 GB  10 %
VM                   J:\       8.127.950.848      9.646.845.952    16,554 GB  54 %
Local Disk           L:\                   0                                {} ... %
ARCHIV               M:\     104.412.012.544        443.822.080    97,655 GB   0 %
Sicherheit           S:\      66.515.734.528     38.340.100.096    97,655 GB  37 %
MEDIEN               U:\      29.233.065.984     12.707.602.432    39,06 GB  30 %
WEB.DE SmartDrive    W:\       7.428.905.984      1.161.028.608        8 GB  14 %
```

Laufwerk anlegen

Die Windows PowerShell ermöglicht das Anlegen logischer Laufwerke im Dateisystem, die auf vorhandene Pfade abgebildet werden. Dazu kann `New-PSDrive` mit `-PSProvider FileSystem` eingesetzt werden. Der Laufwerksname darf mehr als einen Buchstaben umfassen.

Beispiel: `New-PSDrive -PSProvider FileSystem -name M -Root t:\data\marketing`

```
PS C:\Users\hs.ITV> New-PSDrive -PSProvider FileSystem -name M -Root t:\data\marketing

Name           Used (GB)     Free (GB) Provider      Root
----           ---------     --------- --------      ----
M                                11,75 FileSystem    T:\data\marketing
```

***Abbildung 28.2**: Ausführung des obigen Beispiels*



Zu beachten ist, dass die logischen Laufwerke nur innerhalb der PowerShell sichtbar sind und an die PowerShell-Sitzung gebunden sind. Sie werden nach dem Beenden der PowerShell-Konsole also wieder entfernt.

Das Commandlet `Mount-SpecialFolder` erstellt logische Laufwerke für alle Windows-Sonderordner (System, Programs, MyComputer, MyMusic, Desktop etc.). Die Liste aller Sonderordner erhält man mit

```
[Enum]::GetValues([Environment+SpecialFolder]) | foreach { "$_  = "
+ [System.Environment]::GetFolderPath($_) }
```

```
PS C:\Users\hs.ITV>  [Enum]::GetValues([Environment+SpecialFolder]) | foreach { "$_  = " + [System.Environment]::GetFolderPath($_) }
Desktop  = C:\Users\hs.ITV\Desktop
Programs  = C:\Users\hs.ITV\AppData\Roaming\Microsoft\Windows\Start Menu\Programs
Personal  = C:\Users\hs.ITV\Documents
Personal  = C:\Users\hs.ITV\Documents
Favorites  = h:\dok\links
Startup  = C:\Users\hs.ITV\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup
Recent  = C:\Users\hs.ITV\AppData\Roaming\Microsoft\Windows\Recent
SendTo  = C:\Users\hs.ITV\AppData\Roaming\Microsoft\Windows\SendTo
StartMenu  = C:\Users\hs.ITV\AppData\Roaming\Microsoft\Windows\Start Menu
MyMusic  = C:\Users\hs.ITV\Music
DesktopDirectory  = C:\Users\hs.ITV\Desktop
MyComputer  =
Templates  = C:\Users\hs.ITV\AppData\Roaming\Microsoft\Windows\Templates
ApplicationData  = C:\Users\hs.ITV\AppData\Roaming
LocalApplicationData  = C:\Users\hs.ITV\AppData\Local
InternetCache  = C:\Users\hs.ITV\AppData\Local\Microsoft\Windows\Temporary Internet Files
Cookies  = C:\Users\hs.ITV\AppData\Roaming\Microsoft\Windows\Cookies
History  = C:\Users\hs.ITV\AppData\Local\Microsoft\Windows\History
CommonApplicationData  = C:\ProgramData
System  = C:\Windows\system32
ProgramFiles  = C:\Program Files
MyPictures  = C:\Users\hs.ITV\Pictures
CommonProgramFiles  = C:\Program Files\Common Files
```

***Abbildung 28.3**: Liste der Sonderordner*

Die nächste Bildschirmabbildung zeigt die verfügbaren Laufwerke nach der Ausführung von `Mount-SpecialFolder`.

```
PS C:\Users\hs.ITV> Mount-SpecialFolder
PS C:\Users\hs.ITV> Get-PSDrive

Name           Used (GB)     Free (GB) Provider      Root
----           ---------     --------- --------      ----
A                                      FileSystem    A:\
Alias                                  Alias
Applica...                     55,85 FileSystem    C:\Users\hs.ITV\AppData\Roaming
B                                      FileSystem    B:\
C                44,28         55,85 FileSystem    C:\
cert                                   Certificate   \
CommonA...                     55,85 FileSystem    C:\ProgramData
CommonP...                     55,85 FileSystem    C:\Program Files\Common Files
Cookies                        55,85 FileSystem    C:\Users\hs.ITV\AppData\Roaming\...
D                                      FileSystem    D:\
Desktop                        55,85 FileSystem    C:\Users\hs.ITV\Desktop
Desktop...                     55,85 FileSystem    C:\Users\hs.ITV\Desktop
E                                      FileSystem    E:\
Env                                    Environment
F                                      FileSystem    F:\
Favorites                      20,74 FileSystem    H:\dok\links
Function                               Function
G                 ,53           1,43 FileSystem    G:\
H                28,09         20,74 FileSystem    H:\
History                        55,85 FileSystem    C:\Users\hs.ITV\AppData\Local\Mi...
HKCU                                   Registry      HKEY_CURRENT_USER
HKLM                                   Registry      HKEY_LOCAL_MACHINE
I                                      FileSystem    I:\
Interne...                     55,85 FileSystem    C:\Users\hs.ITV\AppData\Local\Mi...
J                                      FileSystem    J:\
LocalAp...                     55,85 FileSystem    C:\Users\hs.ITV\AppData\Local
MyComputer                     55,85 FileSystem    C:\Users\hs.ITV
MyMusic                        55,85 FileSystem    C:\Users\hs.ITV\Music
MyPictures                     55,85 FileSystem    C:\Users\hs.ITV\Pictures
Personal                       55,85 FileSystem    C:\Users\hs.ITV\Documents
Program...                     55,85 FileSystem    C:\Program Files
Programs                       55,85 FileSystem    C:\Users\hs.ITV\AppData\Roaming\...
Recent                         55,85 FileSystem    C:\Users\hs.ITV\AppData\Roaming\...
SendTo                         55,85 FileSystem    C:\Users\hs.ITV\AppData\Roaming\...
StartMenu                      55,85 FileSystem    C:\Users\hs.ITV\AppData\Roaming\...
Startup                        55,85 FileSystem    C:\Users\hs.ITV\AppData\Roaming\...
System                         55,85 FileSystem    C:\Windows\system32
T                 7,79         11,75 FileSystem    T:\
Templates                      55,85 FileSystem    C:\Users\hs.ITV\AppData\Roaming\...
Variable                               Variable
W                74,89          3,24 FileSystem    W:\
WSMan                                  WSMan
X              1175,18        575,98 FileSystem    X:\
Y               107,87       1657,12 FileSystem    Y:\
Z              1175,18        575,98 FileSystem    Z:\
```

Abbildung 28.4 *Mount-SpecialFolder hat zahlreiche Laufwerke für Sonderordner angelegt, z.B. Desktop: und Recent:.*

Füllstand

Um den Füllstand der Dateisystemlaufwerke auszugeben, hat man vier Möglichkeiten:

- Attribut `TotalFreeSpace` in der .NET-Klasse `System.IO.DriveInfo`
- Attribut `Freespace` in der WMI-Klasse `Win32_LogicalDisk`
- Verwendung des Commandlets `Get-Disk` (Commandlet von *www.IT-Visions.de*), das intern WMI verwendet
- Verwendung des Commandlets `Get-FreeDiskSpace` aus dem Modul „FileSystem“ in dem PowerShellPack. Das Commandlet verwendet intern die Leistungsindikatoren von Windows.

```
$drive = New-Object System.IO.DriveInfo("C")
$drive.TotalFreeSpace
```

Listing 28.1*: Auslesen des Füllstands von C unter Einsatz der .NET-Klasse System.IO.DriveInfo*

```
Get-Wmiobject Win32_Logicaldisk -Filter "DeviceID = 'c:'" | Select
FreeSpace
```

Listing 28.2*: Auslesen des Füllstands von C unter Einsatz der WMI-Klasse Win32_LogicalDisk*

```
Get-Wmiobject Win32_Logicaldisk | Select-Object
deviceid,size,freespace
```

Listing 28.3*: Auslesen des Füllstands aller Laufwerke unter Einsatz der WMI-Klasse Win32_LogicalDisk*

Das folgende Skript zeigt eine Möglichkeit, diese Daten besser formatiert auszugeben:

```
$Computer = "localhost"
$laufwerke = Get-wmiobject Win32_LogicalDisk -computer $computer
" Laufwerk        Groesse(MB)    Freier Platz(MB)"
ForEach ($laufwerk in $laufwerke)
{
"    {0}       {1,15:n}    {2,15:n}" -f $laufwerk.DeviceID,
   ($laufwerk.Size/1m), $($laufwerk.freespace/1m)
}
```

Listing 28.4 *Ausgabe des Füllstands der Laufwerke [Freespace.ps1]*

Der Einsatz der WMI-Klasse `Win32_LocigalDisk` bietet zwei Vorteile:

- Man kann auch unter PowerShell 1.0 entfernte Systeme abfragen (siehe Beispiel).
- Man kann auch gezielt mit Hilfe einer WQL-Abfrage filtern (siehe Beispiel).

```
Get-Wmiobject Win32_logicaldisk -Filter "DeviceID = 'c:'" -Computer
E02 | Select DeviceID, FreeSpace
```

Listing 28.5*: Auslesen des Füllstands von Laufwerk C von einem entfernten Computer unter Einsatz der WMI-Klasse Win32_LogicalDisk*

```
([WMISearcher] "Select * from Win32_LogicalDisk where Freespace <
1000000000").Get()  | Select DeviceID, FreeSpace
```

Listing 28.6*: Ermitteln der Laufwerke mit wenig freiem Speicherplatz unter Einsatz einer WQL-Abfrage über die WMI-Klasse Win32_LogicalDisk*

Mit `Get-FreeDiskSpace` kann man alle Laufwerke und einzelne Laufwerke auf dem lokalen oder entfernten Computern abfragen. Eine Besonderheit ist, dass man eine periodische Abfrage einrichten kann.

```
Get-FreeDiskSpace -drive c: -con -sampleinterval 30 -computername
E02
```

Listing 28.7*: Periodische Abfrage (alle 30 Sekunden) des freien Speichers auf Laufwerk C auf dem System E02*

```
PS C:\Users\hs.ITV> Get-FreeDiskSpace -drive c: -con -sampleinterval 30 -computername E02

Timestamp                 CounterSamples
---------                 --------------
19.01.2010 14:30:30       \\e02\\logicaldisk(c:)\% free space :
                          6,88758501133484

                          \\e02\\logicaldisk(c:)\free megabytes :
                          2066


19.01.2010 14:31:00       \\e02\\logicaldisk(c:)\% free space :
                          6,88758501133484

                          \\e02\\logicaldisk(c:)\free megabytes :
                          2066


19.01.2010 14:31:30       \\e02\\logicaldisk(c:)\% free space :
                          6,88758501133484

                          \\e02\\logicaldisk(c:)\free megabytes :
                          2066
```

Abbildung 28.5 *Ausgabe von GetFreediskSpace*

Laufwerksbezeichnungen

Zum Auslesen und Verändern der Laufwerksbezeichnungen kann man `VolumeLabel` aus der Klasse `DriveInfo` verwenden.

Listing 28.8 Ändern der Laufwerksbezeichnung [ChangeVolumeLabel.ps1]

```
$drive = New-Object System.IO.DriveInfo("C")
"Alte Bezeichnung:"
$drive.VolumeLabel
"Neue Bezeichnung:"
$drive.VolumeLabel = "SYSTEM"
$drive.VolumeLabel
```

Alternativ kann man das Commandlet `Set-Volumelabel` aus den PSCX einsetzen (wobei es noch kein Gegenstück „Get-VolumeLabel“ gibt).

```
Set-VolumeLabel c: "Systemlaufwerk"
```

Ebenfalls kann man `Rename-Drive` aus dem FileSystem-Modul des PowerShellPack einsetzen:

```
Rename-Drive t: "Systemlaufwerk"
```

Netzlaufwerke

Informationen über die verbundenen Netzwerklaufwerke des angemeldeten Benutzers liefert die WMI-Klasse `Win32_MappedLogicalDisk`:

```
Get-Wmiobject Win32_MappedLogicalDisk | select caption,
providername
```

28.2 Ordnerinhalte

Get-ChildItem

Den Inhalt eines Dateisystemordners listet man mit `Get-ChildItem` (Alias: `dir`) auf.

Ohne Parameter listet `Get-ChildItem` den aktuellen Pfad auf. Man kann auch einen Pfad explizit angeben:

```
Get-ChildItem c:\temp\Scripte
```

Die Ergebnismenge besteht aus .NET-Objekten der Typen `System.IO.DirectoryInfo` (für Unterverzeichnisse) und `System.IO.FileInfo` (für Dateien).

Der Parameter `-Filter` beschränkt die Ausgabemenge auf Dateien mit einem bestimmten Namensmuster:

```
Get-ChildItem c:\temp\Scripte -filter "*.ps1"
```

Alternativ kann man auch `-include` zum Filtern verwenden und dabei auch mehrere Dateiextensionen angeben:

```
Get-childitem c:\temp\Scripte -include *.ps1,*.vbs
```

Das Commandlet arbeitet normalerweise nur auf der angegebenen Ebene. Es kann auch rekursiv die Unterordner durchsuchen:

```
Get-ChildItem c:\temp\Scripte -filter "*.ps1" -recurse
```

Mit `Measure-Object` kann man Berechnungen über eine Objektmenge ausführen. Der folgende Befehl zeigt die Anzahl der Dateien in *c:\Windows* sowie die Gesamtgröße aller Dateien, die Größe der größten und kleinsten Datei sowie die Durchschnittsgröße.

```
Get-ChildItem c:\windows | Measure-Object -Property length -min -
max -average -sum
```

Mit dem folgenden Befehl erzeugt man eine Liste von großen Word-Dateien auf dem Laufwerk „H“ und seiner Unterverzeichnisse und exportiert eine Liste der Namen und Größe sortiert nach Größe in eine CSV-Datei:

```
Get-ChildItem h:\ -filter *.doc | Where-Object { $_.Length -gt
40000 } | Select-Object Name, Length | Sort-Object Length  |
export-csv p:\GrosseWordDateien.csv -notype
```

Das `-notype` am Ende sorgt dafür, dass der Name der .NET-Klasse nicht exportiert wird. Würde man den Namen exportieren, so könnte man später mit einem `Import-CSV` die Daten wieder als Objekt-Pipeline weiterverarbeiten.



Den Kurznamen einer Datei oder eines Ordners gemäß der alten 8+3-Notation kann man mit dem Commandlet `Get-ShortPath` aus den PSCX ermitteln.

28.3 Dateisystemoperationen

Copy, Move, Rename, Remove

Zum Kopieren von Dateien und Ordnern verwendet man `Copy-Item` (Aliase `copy` oder `cp`):

```
Copy-Item j:\demo\dokumente\profil.pdf c:\temp\profil_
HSchwichtenberg.pdf
```

Zum Bewegen (Verschieben) von Dateisystemobjekten kommt `Move-Item` (Alias `move`) zum Einsatz:

```
Move-Item j:\demo\dokumente\profil.pdf c:\temp\profil_
HSchwichtenberg.pdf
```

Das Commandlet `Rename-Item` (Alias `Rename`) benennt ein Dateisystemobjekt um:

```
Rename-Item profil.pdf profil_HS.pdf
```

Zum Löschen einer Datei verwendet man das Commandlet `Remove-Item` (Alias `del`):

```
Remove-Item j:\demo\profil_HSchwichtenberg.pdf
```



Für `Remove-Item` ist die Simulation des Verhaltens mit `–whatif` eine sehr nützliche Funktion.

Abbildung 28.6
Einsatz von –whatif bei Remove-Item



Der folgende Befehl löscht alle Dateien, die älter als 30 Tage sind:

```
Get-ChildItem c:\temp -recurse | Where-Object {($now - $_
.LastWriteTime).Days -gt 30} | remove-Item
```

28.4 Dateieigenschaften lesen

Get-Item

Informationen über ein Dateisystemobjekt (z.B. Name, Größe, letzte Veränderung, Attribute) erhält man mit `Get-Item`:

```
Get-Item j:\demo\profil_HSchwichtenberg.pdf
```

Für eine Datei erhält man damit eine Instanz von `System.IO.FileInfo`.

Den gleichen Effekt hat auch

```
Get-ItemProperty j:\demo\profil_HSchwichtenberg.pdf
```

Get-ItemProperty

Einzelne Daten kann man so abfragen:

```
Get-ItemProperty daten.txt -name length
Get-ItemProperty daten.txt -name attributes
```

28.5 Datei-Hash

Einen Hash-Wert (alias Fingerabdruck) zur Feststellung, ob der Inhalt einer Datei verändert wurde, kann man mit dem Commandlet `Get-SHA1` aus dem Modul „FileSystem“ (PowerShellPack) ermitteln.

```
PS C:\Users\hs.ITV> Get-SHA1 t:\data\marketing\Notes.jnt
File                                                    SHA1
----                                                    ----
T:\data\marketing\Notes.jnt                             BvfndkuG+6uWOP
```

***Abbildung 28.7**: Beispiel zum Einsatz von Get-SHA1*

28.6 Finden von Duplikaten

`Get-DuplicateFile` setzt auf der zuvor beschriebenen Funktion `Get-SHA1` auf, um Dateien gleichen Inhalts (Duplikate, Dubletten) zu finden.

Beispiel: Der folgende Befehl sucht rekursiv im Verzeichnis „t:\Data“ nach Duplikaten.

```
Get-DuplicateFile t:\data -Recurse
```



***Abbildung 28.8**: Ergebnis der Duplikatsuche*

Die Implementierung dieses Commandlets ist so schön kurz, dass man sie als ein gutes Anschauungsbeispiel für die Prägnanz der PowerShell hier abdrucken und erläutern sollte.

- Im `Begin`-Block wird eine leere Hash-Tabelle angelegt.
- Im `Process`-Block wird eine Liste aller Dateien mit `Get-ChildItem` erzeugt. Für jede Datei wird der Hashwert ermittelt mit `Get-SHA1`. Dann wird jeweils geschaut, ob der Hashwert bereits in der Hashtabelle existiert. Wenn nicht, wird er angelegt. Anschließend wird der Dateipfad in der Hashtabelle dem Hashwert zugeordnet.
- Im `End`-Block muss man dann nur noch prüfen, welche Einträge in der Hashtabelle mehr als einen Wert (also mehr als einen Pfad) besitzen.

***Listing 28.9** Quellcode von Get-DuplicateFile (Quelle: Module "FileSystem" aus dem PowerShell-Pack)*

```
function Get-DuplicateFile {
    <#
    .Synopsis
    Finds files that have identical file contents.

    .Description
    The Get-DuplicateFile function detects files in a drive
    or directory that have identical file content by comparing
    an SHA1 cryptographic hash of the file contents.

    .Outputs
    Returns a System.Collections.DictionaryEntry for each
    collection
    of duplicate files. The key is the hash and the value is
    the fully
    qualified path to the duplicate files.

    .Notes
    To get the cryptographic hash of each file, Get-
    DuplicateFile uses
    the Get-SHA1 functions.
```

```
.Parameter Directory

Enter the path to a file system drive or directory, such as
"C:" or "d:\test". The default is the local directory.

You can enter only one directory with the Directory
parameter. To submit multiple directories, pipe the
directories
to the Get-DuplicateFile function.

.Parameter Recurse

Searches recursively for duplicate files in all
subdirectories
of the specified directory.

.Parameter HideProgress

Hides the progress bar that Get-DuplicateFile displays by
default.

.Example
Get-DuplicateFile

.Example
Get-DuplicateFile c:\users -recurse

.Example
get-item c:\ps-test, d:\testing | get-duplicateFile -
hideprogress

.Link
Get-SHA1
#>

param(
[Parameter(ValueFromPipelineByPropertyName=$true)]
[Alias("FullName")]
[String]
$Directory,

[Switch]
$Recurse,

[Switch]
$HideProgress
)

Begin {
    $duplicateFinder = @{}
```

```
    }

    Process {
        Get-ChildItem $directory -Recurse:$recurse |
            Get-SHA1 |
            ForEach-Object {
                $file = $_.File
                $sha1 = $_.SHA1
                if (-not $hideProgress) {
                    Write-Progress "Finding Duplicate Files"
$file
                }
                if (-not $duplicateFinder.$sha1) {
                    $duplicateFinder.$sha1 = @()
                }
                $duplicateFinder.$sha1 += $file
            }
    }

    end {
        $duplicateFinder.GetEnumerator() |
            Where-Object {
                $_.Value.Count -gt 1
            }
    }
}
```

28.7 Dateieigenschaften verändern

Set-ItemProperty

Zum Ändern von Eigenschaften von Dateisystemeinträgen kann man auf die Attribute der `FileInfo`- und `DirectoryInfo`-Objekte direkt schreibend zugreifen. Ein expliziter Speichervorgang ist nicht vorgesehen.

Das folgende Beispiel zeigt, wie man das Erzeugungsdatum eines Dateisystemordners nachträglich ändert und nachher überprüft, ob es auch wirklich geändert wurde.

```
PS IIS:\sites> $v = Get-Item c:\data\projects
PS IIS:\sites> $v.CreationTime

Montag, 3. August 2009 19:17:20


PS IIS:\sites> $v.CreationTime = new-Object DateTime(2009,3,20)
PS IIS:\sites> $v.CreationTime

Freitag, 20. März 2009 00:00:00


PS IIS:\sites> $v = Get-Item c:\data\projects
PS IIS:\sites> $v.CreationTime

Freitag, 20. März 2009 00:00:00


PS IIS:\sites> _
```

Abbildung 28.9 *Beim Ändern von Attributen in Dateisystemobjekten werden die Änderungen sofort wirksam.*

Eine Alternative ist die Veränderungen von Eigenschaften mit `Set-ItemProperty`. Mit dem folgenden Befehl werden die in `Attributes` gespeicherten Bitflags gesetzt. Die .NET-Klassenbibliothek definiert die möglichen Flags in der Aufzählung `System.IO.FileAttributes`. Wichtig ist, dass die Elemente der Aufzählung wie statische Mitglieder angesprochen (also mit dem `::`-Operator) und mit einem binären exklusiven Oder (`-bxor`) verknüpft werden.

```
Set-ItemProperty daten.txt -name attributes -value
([System.IO.FileAttributes]::ReadOnly -bxor
[System.IO.FileAttributes]::Archive)
```

Zeiten

Die `FileInfo`-Klasse bietet Informationen über das Erstellungsdatum und das Datum des letzten Zugriffs.

```
dir $dir | select name, creationtime, lastaccesstime, lastwritetime
```

Mit `Set-FileTime` (enthalten in den PSCX) kann man diese Daten manipulieren, z.B. wenn man nicht möchte, dass jemand sieht, wie alt eine Datei wirklich ist.

Listing 28.10 *Setzen aller Zeiten aller Dateien in einem Verzeichnis auf das aktuelle Datum und die aktuelle Uhrzeit [/Dateisystem/Filetime.ps1]*

```
$dir = "c:\temp"
$time = [DateTime]::Now
dir $dir | Set-FileTime -Time $time -SetCreatedTime -
SetModifiedTime
dir $dir | select name, creationtime, lastaccesstime, lastwritetime
```

Eigenschaften ausführbarer Dateien

.EXE und .DLL

Für ausführbare Dateien bieten die PSCX einige spezielle Commandlets an:

- `Test-Assembly`: liefert `true`, wenn die Datei eine .NET-Assembly ist (nur anwendbar auf Datei des Typs *.dll*)
- `Get-FileVersionInfo`: liefert Informationen über die Produkt- und Dateiversion
- `Get-PEHeader`: liefert die Kopfinformationen des Portable-Executable-Formats (PE) für beliebige ausführbare Dateien
- `Get-ExportedType`: liefert für eine .NET-Assembly die Liste der von außen instanziierbaren Klassen

Das folgende PowerShell-Skript ermittelt alle mit .NET geschriebenen ausführbaren DLLs im Windows-Verzeichnis und zeigt zu diesen DLLs die Versionsinformationen an.

```
"Suche .NET-Assemblies"

foreach ( $d in (Get-childitem c:\Windows\ -include "*.dll" -
recurse))
{
$a = $d.Fullname | Test-assembly -ErrorAction SilentlyContinue
if ($a) { Get-FileVersionInfo $d.Fullname }
}
```

Listing 28.11
Suche nach .NET-Assemblies [/Dateisystem/Assembly-Search.ps1]

Das folgende Beispiel liefert die PE-Kopfinformationen über den Windows Editor:

```
Get-PEHeader  C:\windows\system32\notepad.exe
```

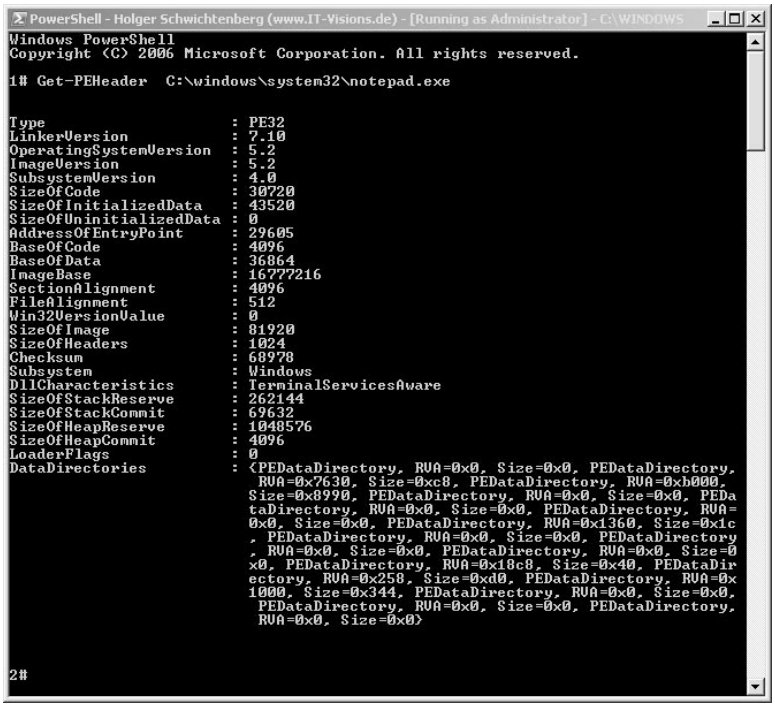


Abbildung 28.10
Ausgabe der PE-Kopfinformationen

Mit dem Commandlet `Resolve-Assembly` kann man prüfen, welche Versionen einer .NET-Softwarekomponente vorliegen bzw. ob eine bestimmte Version vorliegt.

```
# Zeige alle Versionen dieser Assembly
Resolve-Assembly System.Windows.Forms
# Prüfe, ob Version 3.0 verfügbar ist
Resolve-Assembly System.Windows.Forms -Version 3.0.0.0
```

28.8 Verknüpfungen im Dateisystem

Commandlets zur Erstellung von Verknüpfungen findet man in den PSCX.

Explorer-Verknüpfungen

New-Shortcut

Seit Windows 95 unterstützt der Windows Explorer Verknüpfungen im Dateisystem durch *.lnk*-Dateien. *.lnk*-Dateien enthalten als Verknüpfungsziel entweder eine Datei oder ein Verzeichnis. Sie werden erstellt im Windows Explorer durch die Kontextmenüfunktionen „Verknüpfung erstellen" oder „Neu/Verknüpfung". Windows zeigt die Dateinamenserweiterung von *.lnk*-Dateien nicht an. Stattdessen sieht man im Windows Explorer das Symbol des Zielobjekts mit einem Pfeil. Ein Doppelklick leitet den Windows Explorer oder einen Dateidialog, der *.lnk*-Dateien unterstützt, zum Ziel.

Diese Explorer-Verknüpfungen erstellt man mit dem Commandlet `New-Shortcut`, wobei der erste Parameter der Pfad zu der zu erstellenden .lnk-Datei ist und der zweite Parameter der Zielpfad:

```
New-Shortcut "c:\Kundendaten" "g:\Daten\Kunden"
```



Falls die Verknüpfung bereits existiert, wird sie ohne Vorwarnung überschrieben. `New-Shortcut` kann keine Verknüpfungen zu HTTP-Adressen (.url-Dateien) herstellen.

URL-Verknüpfungen

URL-Verknüpfungen legt Windows in .url-Dateien ab.

Abbildung 28.11 *Innereien einer .url-Datei*

```
[{000214A0-0000-0000-C000-000000000046}]
Prop3=19,2
[InternetShortcut]
URL=http://www.it-visions.de/
IDList=
```

Das Commandlet `Resolve-ShortcutFile` (Modul „FileSystem" im PowerShellPack) extrahiert die URL aus einer solchen .url-Datei.

```
Resolve-ShortcutFile t:\ITVWebsite.url
```

Abbildung 28.12 *Ausgabe von Resolve-ShortcutFile*

```
PS C:\Users\hs.ITV> Resolve-ShortcutFile t:\ITVWebsite.url | fl

ShortcutFile : T:\ITVWebsite.url
Url          : http://www.it-visions.de/
```

Zum Erstellen von .url-Dateien gibt es noch kein Commandlet. Ein Commandlet `New-UrlShortcut` zu erstellen, ist aber nicht schwer, denn es gibt im Windows Script Host (WSH) ein COM-Objekt „WScript.Shell„, das eine Methode `CreateShortcut()` anbietet, die wahlweise .url- oder .lnk-Dateien erstellt.

Listing 28.12
Implementierung und Test von New-UrlShortcut

```
function New-UrlShortcut
{ param(
    [Parameter(
        ValueFromPipeline=$true,
        ValueFromPipelineByPropertyName=$true)]
    [Alias("FullName")]
    [string] $Filename,
[string] $URL)

process {
$shell = New-Object -ComObject WScript.Shell
$us = $shell.CreateShortcut($Filename)
$us.TargetPath = $URL
$us.Save()
}
}

# Zum Testen
New-UrlShortcut "t:\PowerShellWebsite.url" "http://www.powershell-
doktor.de
```

Hardlinks

New-Hardlink

Leider gibt es drei gravierende Nachteile bei auf *.lnk*-Dateien basierenden Explorer-Verknüpfungen:

- Der Windows Explorer zeigt Verknüpfungen zu Ordnern nicht in der Ordnerhierarchie (links) an, sondern sortiert sie in die Dateiliste (rechts) ein.
- Die Verknüpfungen funktionieren nicht an der Kommandozeilenebene.
- Windows verfolgt das Ziel nicht beim Umbenennen/Verschieben, sondern sucht stets erst danach, wenn das Ziel nicht mehr auffindbar ist, wobei nicht immer das richtige Ziel gefunden wird.

Nutzer von Unix kennen hingegen bessere Verknüpfungsarten in Form von Hardlinks und symbolischen Links (Symbolic Links/Symlinks). Unter Windows können Nutzer von NTFS-Dateisystemen ähnliche Konzepte nutzen. Das NTFS-Dateisystem unterstützt feste Verknüpfungen zu Dateien in Form sogenannter Hardlinks und zu Ordnern in Form von Junction Points. Leider werden beide Funktionen nicht direkt im Windows Explorer, sondern nur durch Kommandozeilen- oder Drittanbieterwerkzeuge unterstützt.

Ein Hardlink ist eine feste Verknüpfung zu einer Datei. Microsoft liefert dazu in Windows XP und Windows Server 2003 das Kommandozeilenwerkzeug *fsutil.exe*. In den PowerShell Extensions findet man das Commandlet `New-Hardlink`.

Die Syntax zum Erstellen eines Hardlinks lautet:

```
New-Hardlink <neuer Dateiname> <vorhandener Dateiname>
```

Beispiel:

```
New-Hardlink "g:\Kunden.csv" "g:\Daten\Kunden\Kundenliste.csv"
```

Danach erscheint die Datei in beiden Verzeichnissen – ohne dass ein Verknüpfungspfeil angezeigt würde. Es handelt sich dennoch nicht um eine Kopie; beide Einträge im Verzeichnisbaum weisen auf die gleiche Stelle auf der Festplatte und daher kann die Datei nun an beiden Stellen manipuliert werden. Ein Verschieben der Datei macht überhaupt keine Probleme. Der Dateiinhalt ist erst dann verloren, wenn beide Einträge im Verzeichnisbaum gelöscht wurden.

Zwei Wermutstropfen:

- Es können keine Ordnerverknüpfungen erstellt werden.
- Es können nur Verknüpfungen zu Dateien auf dem gleichen Laufwerk erstellt werden.



Einen Hardlink löscht man, indem man die Linkdatei entfernt. Die Zieldatei bleibt dabei unangetastet:

```
Remove-Item "g:\Kunden.csv"
```

Junction Points

NewJunction

Junction Points sind das Äquivalent zu Hardlinks für Ordner. Im Gegensatz zu Hardlinks funktionieren Junction Points auch laufwerkübergreifend. Als Commandlet kommt hier das Commandlet `New-Junction` zum Einsatz, das jedoch leider nur mit den Ressource Kits der verschiedenen Windows-Versionen ausgeliefert wird. Bei *linkd.exe* ist im Gegensatz zu *fsutil.exe* erst die Quelle und dann das Ziel zu nennen.

Beispiel:

Die Anweisung

```
New-Junction "h:\Kunden" "g:\Daten\Kunden\"
```

erstellt folglich hier eine Verknüpfung, die das Verzeichnis *g:\Daten\Kunden* als Unterverzeichnis *Kunden* im Ordner *h:* einblendet. Junction Points funktionieren auch an der Kommandozeile. So zeigt der Befehl

```
dir h:\Kunden
```

den Inhalt von *g:\Daten\Kunden*.

Der Windows Explorer sortiert einen Junction Point wie einen Ordner in die Ordnerhierarchie auf der linken Seite ein.

Das Ziel eines Junction Point kann man mit dem Commandlet `Get-ReparsePoint` betrachten, z.B.:

```
Get-ReparsePoint c:\Kunden
```

Zum Löschen eines Junction Point verwendet man:

Remove-Reparse-Point

```
Remove-ReparsePoint "c:\Kunden"
```



Wird der eigentliche Zielordner vor dem Junction Point gelöscht, entsteht ein verwaister Junction Point. Leider bemerkt Windows das Verschieben eines Ordners nicht, so dass auch in diesem Fall der Junction Point ins Nirwana führt.

Symbolische Verknüpfungen ab Windows Vista

NewSymlink

Die neuen symbolischen Verknüpfungen (Symbolic Links), die Microsoft mit Windows Vista eingeführt hat, kann man mit den Commandlet `New-Symlink` erstellen.

28.9 Komprimierung

ZIP, GZIP, TAR und BZIP2

Commandlets zum Komprimieren von Dateien in Archive findet man in den PSCX und im Modul „FileSystem“ im PowerShellPack.

In den PSCX gibt es Commandlets für vier verschiedene Komprimierungsformate (ZIP, GZIP, TAR und BZIP2):

- `Write-Zip`
- `Write-GZip`
- `Write-Tar`
- `Write-BZip2`
- Das PowerShellPack bietet:
- `New-Zip`: Erstellen eines neuen ZIP-Archivs
- `Copy-ToZip`: Erstellen eines ZIP-Archivs oder Hinzufügen von Dateien zu einem Archiv

Im Folgenden finden Sie einige aussagekräftige Praxisbeispiele, welche die Syntax der Befehle erläutern. Alle Beispiele verwenden einheitlich das ZIP-Format. Alle anderen Formate funktionieren analog mit dem entsprechenden Commandlet aus den PSCX.

Tabelle 28.2 Anwendungsbeispiele für Write-Zip

`Write-zip Kundenliste.csv` Komprimiert die Datei Kundenliste.csv in das Archiv Kundenliste.csv.zip
`Write-zip Kundenliste.csv Kundenliste.zip` Komprimiert die Datei Kundenliste.csv zu Kundenliste.zip
`"Kundenliste.csv", "Preisliste.doc", "Projektrichtlinien.doc" | Write-Zip` Komprimiert die drei angegebenen Dateien einzeln in "Kundeniste.csv.zip", "Preisliste.doc.zip" und "Projektrichtlinien.doc.zip"
`"Kundenliste.csv", "Preisliste.doc", "Projektrichtlinien.doc" | Write-Zip -Outputpath G:\daten\kunden.zip` Komprimiert die drei angegebenen Dateien zusammen in Kunden.zip
`Write-Zip g:\daten\kunden -Outputpath G:\daten\kunden.zip` Komprimiert den ganzen Inhalt des Ordners g:\daten\kunden nach kunden.zip
`dir g:\daten -Filter *.doc -Recurse | Write-zip -Output g:\Daten\docs.zip` Sucht im Ordner g:\Daten und allen seinen Unterordnern nach Microsoft Word-Dateien und komprimiert diese zusammen in *g:\Daten\docs.zip*
`Copy-ToZip g:\daten\kunden G:\daten\kunden.zip` Komprimiert den Inhalt eines Verzeichnisses



Wenn die Zieldatei bereits existiert, werden die neuen Dateien mit in das Archiv aufgenommen. Die bestehenden Dateien werden nicht gelöscht.

Die Komprimierungs-Commandlets in den PSCX besitzen noch einige interessante Optionen, von denen beispielhaft zu nennen sind:

- `RemoveOriginal`: Löschen der Originaldatei nach Aufnahme der Datei in das Archiv
- `Level`: Komprimierungsrate von 1 bis 9 (Standard ist 5)
- `FlattenPaths`: In dem Archiv werden keine Pfadinformationen gespeichert.

Komprimierung ohne Zusatzkomponenten

Eine ZIP-Datei können Sie auch ganz ohne eine zusätzliche Commandlet-Bibliothek erstellen, indem man auf das COM-Objekt „Shell.Application“ zurückgreift (siehe folgendes Listing).

Listing 28.13 Komprimieren mit dem COM-Objekt "Shell.Application" (ZIPWithCOM.ps1)

```
$Path = "t:\markting.zip"
# Erzeuge leere ZIP-Datei
Set-Content $path ("PK" + [char]5 + [char]6 + ("$([char]0)" * 18))
# Shell-Objekt
$ShellApplication = New-Object -ComObject Shell.Application
# Zugriff auf Paket
$ZipPackage =$ShellApplication.Namespace($Path)
```

```
# Zwei Dateien kopieren
$ZipPackage.CopyHere("T:\Data\marketing\Website.doc", 0)
$ZipPackage.CopyHere("T:\Data\marketing\Notes.jnt", 0)
```

28.10 Dateisystemfreigaben

Win32_Share

Der Zugriff auf Dateisystemfreigaben (engl. Shares) erfolgt über die WMI-Klasse `Win32_Share`. Wichtige Mitglieder der Klasse sind:

- `Name`: Name der Freigabe
- `Path`: Pfad im Dateisystem, zu dem die Freigabe führt
- `Description`: Beschreibungstext zu der Freigabe
- `MaximumAllowed`: Maximalanzahl der gleichzeitigen Benutzer
- `SetShareInfo()`: Setzen der Eigenschaften `Description`, `MaxiumAllowed` und der Berechtigungen für die Freigabe
- `GetAccessMask()`: Auslesen der Berechtigungen für die Freigabe
- `Create()`: `Create` ist eine statische Methode der Klasse `Win32_Share` zum Anlegen neuer Freigaben



Das Attribut `AccessMask` ist immer leer (siehe Bildschirmabbildung), weil es von Microsoft als „veraltet“ deklariert wird. Das Setzen und Lesen der Berechtigungen erfolgt über die Methoden `Create()`, `SetShareInfo()` und `GetAccessMask()`. Diese Methoden legen entsprechende Assoziationen an.

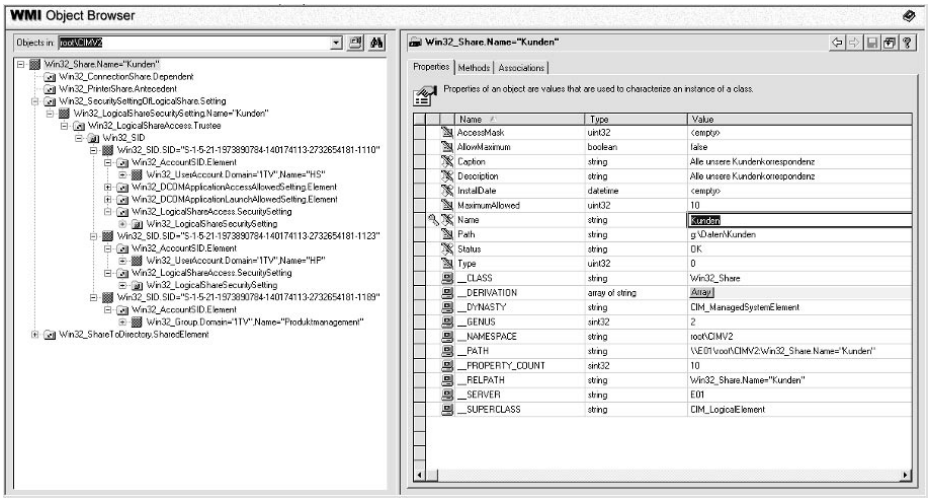


***Abbildung 28.13**: Darstellung einer Instanz der Klasse Win32_Share im WMI Object Browser*

Bei den Freigaben sind die Berechtigungen der komplizierteste Teil, wie schon die Assoziationen im WMI Object Browser andeuten.

Freigaben auflisten

Win32_Share Zum Auflisten der Freigaben muss man auf die Instanzen der WMI-Klasse `Win32_Share` zurückgreifen:

```
Get-Wmiobject Win32_Share
```

Abbildung 28.14 *Auflisten der freigegebenen Dateisystemverzeichnisse*



Über den Namen der Freigabe kann man eine Freigabe (auch auf entfernten Systemen) gezielt ansprechen:

```
Get-WmiObject Win32_Share -Filter "Name='C$'" -computer E02 |
Select Name, Path, Description, MaximumAllows | Format-List
```

Freigaben anlegen

Create() Das Anlegen einer Freigabe ist eine aufwändigere Angelegenheit – zumindest dann, wenn man auch die Zugriffsrechteliste setzen will. Leider kann man hier nicht auf die .NET-Klassen für die Berechtigungsvergabe zurückgreifen, sondern muss entsprechende WMI-Klassen verwenden.

Aus didaktischen Gründen folgt zunächst erst einmal ein Skript, bei dem die Berechtigungen nicht explizit gesetzt werden. Die Freigabe erhält dadurch die Standardrechte (Vollzugriff für jedermann). Zum Anlegen der Freigabe wird die statische Methode `Create()` der Klasse `Win32_Share` aufgerufen. Für `AccessMask` wird dabei `$null` übergeben. Das Skript prüft beim Start, ob es die Freigabe schon gibt, und löscht diese gegebenenfalls, damit eine Neuanlage möglich ist.

Listing 28.14 *Anlegen einer Freigabe mit Standardberechtigungen [New-Share-withoutPermissions.ps1]*

```
##############################################
# New-Share (without Permissions)
# (C) Dr. Holger Schwichtenberg
##############################################

# Parameters
$Computer = "."
$ShareName = "Kunden"
$Pfad = "g:\Daten\Kunden"
$Comment = "Alle unsere Kundenkorrespondenz"

"Vorher:"
Get-WmiObject Win32_Share -Filter "Name='$ShareName'"

Get-WmiObject Win32_Share -Filter "Name='$ShareName'" | foreach-
Object { $_.Delete() }
```

```
# Win32_Share
$MC = [WMIClass] "ROOT\CIMV2:Win32_Share"
$Access = $Null
$R = $mc.Create($pfad, $Sharename, 0, 10, $Description, "",
$Access)

if ( $R.ReturnValue -ne 0) { Write-Error "Fehler beim Anlegen: "+
$R.ReturnValue; Exit}
"Freigabe wurde angelegt!"

"Nachher:"
Get-WmiObject Win32_Share -Filter "Name='$ShareName'"
```

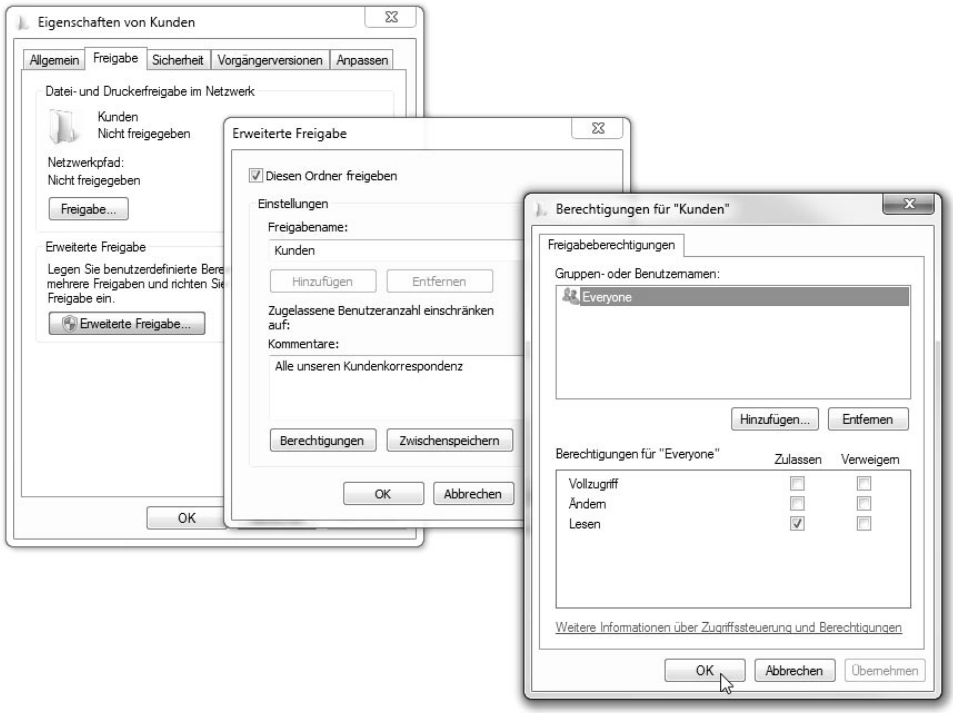


***Abbildung 28.15**: Eine mit Standardrechten angelegte Freigabe*

Berechtigungen auf Freigaben setzen

Um beim Anlegen einer Dateisystemfreigabe die Zugriffsrechte zu setzen, sind folgende Schritte zusätzlich notwendig:

- Ermitteln des Security Identifier für jeden Benutzer/jede Gruppe, die Rechte erhalten soll
- Erstellen einer Instanz von `Win32_Trustee` für jeden Benutzer/jede Gruppe, die Rechte erhalten soll
- Instanziieren der Klasse `Win32_ACE` für jeden Rechteeintrag
- Befüllen von `Win32_ACE` mit dem `Win32_Trustee`-Objekt, den Rechten und den Rechteeigenschaften
- Erstellen einer Instanz von `Win32_SecurityDescriptor`
- Befüllen des `Win32_SecurityDescriptor`-Objekts mit einer Discretionary Access Control List (DACL)
- Zusammenbauen der DACL aus einzelnen Rechteeinträgen (Access Control Entries – ACE), also Instanzen von `Win32_ACE`

Listing 28.15 *Anlegen einer Freigabe mit expliziten Rechten [New-Share-withPermission.ps]*

```
##############################################
# New-Share (with Permissions)
# (C) Dr. Holger Schwichtenberg
##############################################

# Parameters
$Computer = "."
$ShareName = "Kunden"
$Pfad = "g:\Daten\Kunden"
$Comment = "Alle unsere Kundenkorrespondenz"

# Constants
$SHARE_READ = 1179817
$SHARE_CHANGE = 1245462
$SHARE_FULL = 2032127
$SHARE_NONE = 1

$ACETYPE_ACCESS_ALLOWED = 0
$ACETYPE_ACCESS_DENIED = 1
$ACETYPE_SYSTEM_AUDIT = 2

$ACEFLAG_INHERIT_ACE = 2
$ACEFLAG_NO_PROPAGATE_INHERIT_ACE = 4
$ACEFLAG_INHERIT_ONLY_ACE = 8
$ACEFLAG_INHERITED_ACE = 16
$ACEFLAG_VALID_INHERIT_FLAGS = 31
$ACEFLAG_SUCCESSFUL_ACCESS = 64
$ACEFLAG_FAILED_ACCESS = 128

# Get Trustee
function New-Trustee($Domain, $User)
{
```

```
$Account = New-Object system.security.principal.ntaccount("itv\
hs")
$SID =
$Account.Translate([system.security.principal.securityidentifie
r])
$useraccount = [ADSI] ("WinNT://" + $Domain + "/" + $User)
$mc = [WMIClass] "Win32_Trustee"
$t = $MC.CreateInstance()
$t.Domain = $Domain
$t.Name = $User
$t.SID = $useraccount.Get("ObjectSID")
return $t
}
# Create ACE
function New-ACE($Domain, $User, $Access, $Type, $Flags)
{
$mc = [WMIClass] "Win32_Ace"
$a = $MC.CreateInstance()
$a.AccessMask = $Access
$a.AceFlags = $Flags
$a.AceType = $Type
$a.Trustee = New-Trustee $Domain $User
return $a
}

# Create SD
function Get-SD
{
$mc = [WMIClass] "Win32_SecurityDescriptor"
$sd = $MC.CreateInstance()
$ACE1 = New-ACE "ITV" "HP" $SHARE_READ  $ACETYPE_ACCESS_ALLOWED
$ACEFLAG_INHERIT_ACE
$ACE2 = New-ACE "ITV" "HS" $SHARE_FULL $ACETYPE_ACCESS_ALLOWED
$ACEFLAG_INHERIT_ACE
$ACE3 = New-ACE "ITV" "Produktmanagement" $SHARE_FULL $ACETYPE_
ACCESS_ALLOWED $ACEFLAG_INHERIT_ACE
[System.Management.ManagementObject[]] $DACL = $ACE1 , $ACE2,
$ACE3

$sd.DACL = $DACL
return $sd
}

# before
"Vorher:"
Get-WmiObject Win32_Share -Filter "Name='$ShareName'"

Get-WmiObject Win32_Share -Filter "Name='$ShareName'" |
foreach-Object { $_.Delete() }

# Win32_Share anlegen
```

```
$MC = [WMIClass] "ROOT\CIMV2:Win32_Share"
$Access = Get-SD
$R = $mc.Create($pfad, $Sharename, 0, 10, $Comment, "",
$Access)

if ( $R.ReturnValue -ne 0) { Write-Error "Fehler beim Anlegen:
"+ $R.ReturnValue; Exit}
"Freigabe wurde angelegt!"

# after
"Nachher:"

Get-WmiObject Win32_Share -Filter "Name='$ShareName'" | foreach
{ $_.GetAccessMask()  } | gm
```

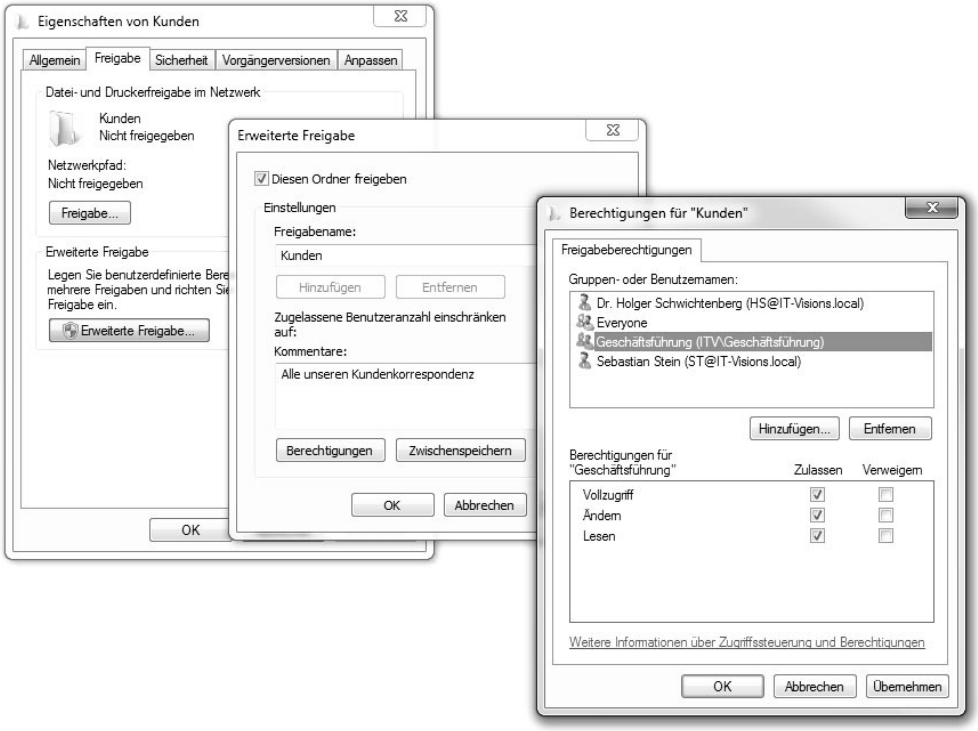


***Abbildung 28.16**: Ergebnis des obigen Skripts zum Anlegen einer Freigabe mit expliziten Rechten*



`Create()` besitzt einige Fehlercodes, z.B. 22 = Freigabename existiert bereits oder 21 = Falsche Parameter.

28.11 Überwachung des Dateisystems

Das .NET Framework stellt im Namensraum `System.IO` eine Klasse `FileSystemWatcher` bereit, die beim Eintreten von Veränderungen im Dateisystem ein Ereignis auslösen kann. Überwachbare Veränderungen sind das Anlegen, Umbenennen und Löschen von Dateien und Ordnern sowie die Änderungen von Eigenschaften an diesen Objekten.



Man kann sowohl lokale Ordner als auch Netzlaufwerke und UNC-Pfade überwachen. In Tests hat die Überwachung sogar mit Freigaben auf Linux-Systemen, die SAMBA verwenden, funktioniert.

Das folgende Listing zeigt die Instanziierung und Konfiguration des `FileSystemWatcher` inklusive der Ereignisbindung mit `Register-ObjectEvent`.

Listing 28.16 *Einsatz des FileSystemWatcher zur Überwachung eines Verzeichnisses*

```
"Starte Überwachung der Word-Dateien in t:\Data ..."

$fsw = New-Object System.IO.FileSystemWatcher
$fsw.Path = "t:\data"
$fsw.IncludeSubdirectories = $true
$fsw.filter = "*.doc"

$aktion = {
[console]::beep(440,10)
Write-Host
Write-Warning "Dateisystemereignis: " $eventArgs.FullPath ": "
$eventArgs.ChangeType
    }

Register-ObjectEvent -InputObject $fsw -EventName Created -Action
$aktion
Register-ObjectEvent -InputObject $fsw -EventName Changed -Action
$aktion
Register-ObjectEvent -InputObject $fsw -EventName Deleted -Action
$aktion
Register-ObjectEvent -InputObject $fsw -EventName Renamed -Action
$aktion

"Überwachung läuft!"
```

Mit dem Commandlet `New-FileSystemWatcher` aus dem PowerShell-Pack (Modul „FileSystem“) kann man etwas prägnanter arbeiten.

Listing 28.17 *Einsatz von New-FileSystemWatcher*

```
Import-Module FileSystem
"Starte Überwachung der Word-Dateien in t:\Data ..."
$aktion = {
        [console]::beep(440,10)
        Write-Host
Write-Host "Dateisystemereignis: " $eventArgs.FullPath ": "
$eventArgs.ChangeType
    }
Start-FileSystemWatcher -Filter *.doc -file "t:\data" -do $aktion -
recurse
"Überwachung läuft!"
```

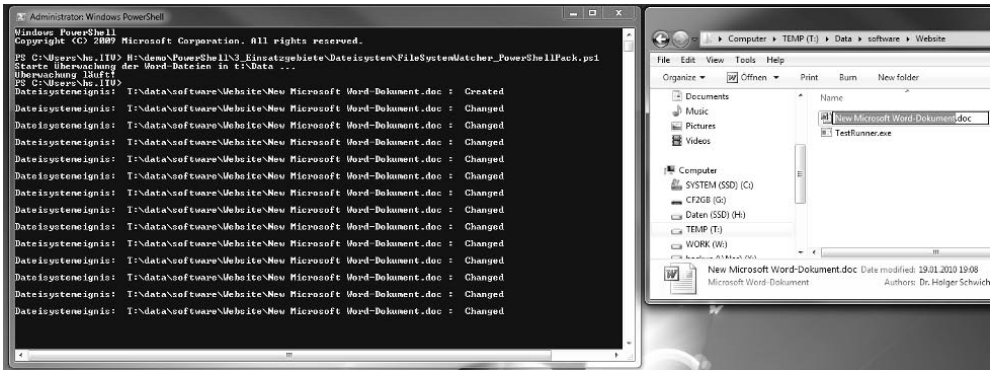


Abbildung 28.17*: Ereignisse nach dem Anlegen einer Word-Datei im Windows Explorer*

29 Dokumente

Dieses Kapitel behandelt die Erstellung und Nutzung verschiedener Dokumententypen: Textdateien, Binärdateien, CSV-Dateien und XML-Dateien.

29.1 Textdateien

Get-Content

Zum Einlesen von Dateien stellt die PowerShell das Commandlet `Get-Content` zur Verfügung. `Get-Content` liest im Standard die gesamte Datei ein.

Das folgende Listing zeigt das Einlesen einer Textdatei und die zeilenweise Ausgabe:

Listing 29.1 *Zeilenweises Einlesen einer Textdatei [Textfile_Read.ps1]*

```
$datei = Get-Content j:\demo\dokumente\benutzerliste.csv
$a = 0
$datei | Foreach-Object { $a++; "Zeile" + $a + ": " + $_ }
"Gesamtzahl der Zeilen: " + $a
```

Wenn es nur darum geht, die Anzahl der Zeilen zu ermitteln, dann geht das auch kürzer:

```
Get-Content j:\demo\dokumente\benutzerliste.csv | Measure-Object
```

Set-Content

Das Beschreiben einer Textdatei im Dateisystem erfolgt mit `Set-Content` und `Add-Content`. `Set-Content` tauscht den Inhalt aus, `Add-Content` ergänzt Inhalte.

Listing 29.2 *Erstellen und Ergänzen einer Textdatei [Textfile_Write.ps1]*

```
$datei = "j:\demo\dokumente\protokoll.txt"
"Neubeginn der Protokolldatei " | Set-Content $datei

"Neuer Eintrag " | Add-Content $datei
"Neuer Eintrag " | Add-Content $datei
"Neuer Eintrag " | Add-Content $datei

"Inhalt der Datei jetzt:"

Get-Content $datei
```

`Clear-Content` löscht den Inhalt einer Datei, belässt die Datei aber leer im Dateisystem.

New-Item Eine andere Möglichkeit zum Erstellen einer Textdatei ist die Verwendung von `New-Item`:

```
New-Item . -name Daten.txt -type "file" -value "Dies ist der
Inhalt!" -force
```

In diesem Fall gibt es aber nur die Option, die Datei neu anzulegen (ohne `-force`) oder eine bestehende Datei zu überschreiben (mit `-force`).

Eine dritte Möglichkeit zum Beschreiben einer Datei ist das Commandlet `Out-File`.

Suche in Textdateien Das Durchsuchen von Textdateien ist möglich mit dem Commandlet `Select-String`. Die folgende Anweisung liefert Informationen dazu, in welchen Skriptdateien in einer Verzeichnishierarchie das Wort „Where" vorkommt:

```
Get-ChildItem h:\demo\powershell -Filter *.ps1 -Recurse | Select-
String "Where"
```

29.2 Binärdateien

Auch Binärdateien kann man mit `Get-Content` auslesen und mit `Set-Content` bzw. `Add-Content` beschreiben. Jeweils ist als Parameter `-encoding Byte` anzugeben.

Listing 29.3 *Lesen und Schreiben einer Binärdatei [BinaryFile_Read-Write.ps1]*

```
# Binärdatei lesen
$a = Get-Content H:\demo\PowerShell\Registry\www.IT-Visions.de_
Logo.jpg -encoding byte

# Binärdatei schreiben
$a | Set-Content "g:\Daten\Logo.jpg" -encoding byte
```

29.3 CSV-Dateien

Export, Import Zum Importieren und Exportieren von Daten im CSV (Comma-Separated Values)-Format bietet die PowerShell die Commandlets `Import-CSV` und `Export-CSV`.



Im Standard verwenden diese Commandlets als Trennzeichen ein Komma („,"), nicht wie in Deutschland üblich ein Semikolon. Um das Semikolon zu nutzen, müssen Sie -Delimiter ";" als Parameter angeben. Um deutsche Umlaute zu exportieren, müssen Sie einen entsprechenden Zeichensatz angeben, z.B. `-Encoding UTF8`.

Neben der Frage des Trennzeichens ist noch zu entscheiden, ob man eine normale CSV-Datei oder eine CSV-Datei mit Metainformationen über den exportierten Typ erzeugen möchte.

Man kann eine normale CSV-Datei ohne Metadaten erstellen lassen:

```
Get-Service | Where-Object {$_.status -eq "running"} | export-csv
c:\temp\dienste.csv -NoTypeInformation
```

Alternativ dazu kann man eine CSV-Datei erzeugen, bei der in der ersten Zeile nach „#Type“ hinterlegt ist, welche Objekttypen in der Datei persistiert sind. Dies hat den Vorteil, dass man bei dem späteren Import wieder Objekte dieses Typs erhält. Allerdings versteht kein Werkzeug außer der PowerShell diese Syntax.

```
Get-Service | Where-Object {$_.status -eq "running"} | export-csv
c:\temp\dienste.csv
```

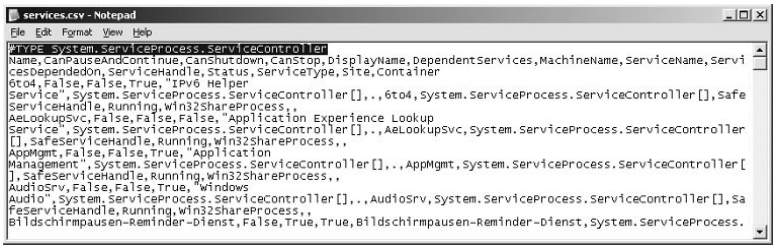


Abbildung 29.1
Exportierte CSV mit Typinformationen in der ersten Zeile

Beim Import einer CSV-Datei mit

```
Import-csv c:\temp\dienste.csv | where { $_.Status -eq "Running" }
```

entscheidet die Typinformation darüber, welcher Objekttyp konstruiert wird. Mit Typinformationen wird der entsprechende Typ erstellt, mit dem Zusatz „CSV“ (siehe Bildschirmabbildung).



Das Exportieren/Importieren entspricht einer Serialisierung/Deserialisierung von Objekten. Die deserialisierten Objekte haben zwar alle Attribute (also Daten) des Ursprungobjekts, nicht aber deren spezifische Methoden.

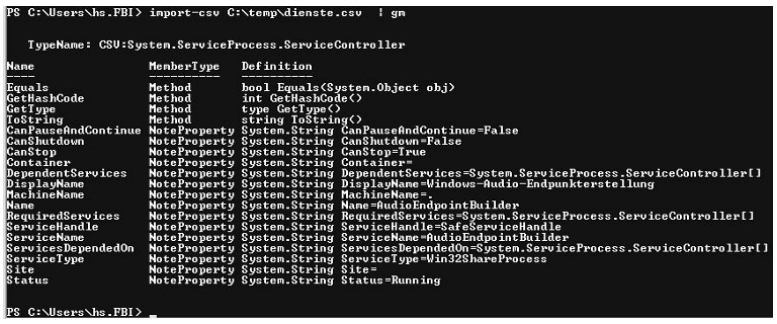


Abbildung 29.2
Deserialisierte Dienst-Objekte

Ohne Typinformationen entstehen Instanzen der Klasse `System.Management.Automation.PSCustomObject`.

ConvertTo-CSV

`ConvertTo-CSV` (ab PowerShell 2.0) erzeugt genau wie `Export-CSV` ein CSV-Dokument (Comma-Separated Values) aus einer Objektmenge. Der Unterschied ist aber, dass `ConvertTo-CSV` die Ausgabe nicht in ein Dokument schreibt, sondern innerhalb der PowerShell als Zeichenkette zur Weiterverarbeitung anbietet.

Beispiel:

```
Get-Service I* | ConvertTo-Csv
```



Abbildung 29.3 Ausgabe von ConvertTo-CSV nach dem Befüttern mit Instanzen von System.ServiceProcess.ServiceController

29.4 INI-Dateien

Zum Einlesen von „alten" INI-Dateien kann man auf `Import-INIFile` aus dem PowerShellPack (Modul `PSSystemTools`) zurückgreifen. `Import-INIFile` erzeugt aus der INI-Datei ein `PSCustomObject` mit den Einträgen der INI-Datei als Attribute (`NoteProperty`).



***Abbildung 29.4**: Zugriff auf INI-Dateien am Beispiel der Application.ini-Datei von Firefox*

29.5 XML-Dateien

[Xml]

Die PowerShell bietet eine sehr komfortable Möglichkeit, XML-Dokumente auszuwerten, denn die XML-Elementnamen können wie Attribute eines .NET-Objekts angesprochen werden. Wenn `$doc` das in der nachstehenden Abbildung gezeigte XML-Dokument enthält, dann liefert `$doc.Websites.Website` die Menge von XML-Knoten, die <*Website*> heißen.

Websites.xml

```
<?xml version="1.0" encoding="utf-8"?>
<Websites>
 <Website ID="1">
  <URL>www.IT-Visions.de</URL>
  <Beschreibung>Websites der Firma des Autors</Beschreibung>
 </Website>
 <Website ID="2">
  <URL>www.dotnetframework.de</URL>
  <Beschreibung>Community-Site für .NET-Entwickler</Beschreibung>
 </Website>
 <Website ID="3">
  <URL>www.Windows-Scripting.de</URL>
  <Beschreibung>Community-Site für WSH- und PowerShell-Entwickler</Beschreibung>
 </Website>
 <Website ID="4">
  <URL>www.powershell-doktor.de</URL>
  <Beschreibung>Community-Site zur Microsoft PowerShell</Beschreibung>
 </Website>
</Websites>
```

Abbildung 29.5
Beispiel für ein XML-Dokument

Das obige Dokument kann so ausgewertet werden:

```
$doc = [xml] (Get-Content -Path h:\demo\dokumente\websites.xml)
$Sites = $doc.Websites.Website
$Sites | select URL, Beschreibung
```

Listing 29.4
Auslesen einer XML-Datei [XML_Document.ps1]



Um die besondere XML-Unterstützung der PowerShell nutzen zu können, muss die PowerShell wissen, welche Variablen ein XML-Dokument enthält. Daher ist die Typzuweisung mit `[xml]` in der ersten Zeile sehr wichtig.

```
PS J:\Demo\Dokumente> $doc = [xml] (Get-Content -Path j:\demo\dokumente\websites
.xml)
PS J:\Demo\Dokumente> $doc.Websites.Website | select URL, Beschreibung

URL                                Beschreibung
---                                ------------
www.dotnetframework.de             Community-Site für .NET-Entwickler
www.Windows-Scripting.de           Community-Site für WSH- und PowerShe...
www.aspnetdev.de                   Community-Site für ASP.NET-Entwickler
www.komponenten.info               Softwarekomponentenkatalog
www.IT-Visions.de                  Websites des Autors


PS J:\Demo\Dokumente> _
```

Abbildung 29.6
Ergebnis der Auswertung des XML-Dokuments

Prüfung von XML-Dokumenten

Test-Xml

Den Versuch, ein nicht gültiges XML-Dokument (in dem z.B. ein schließendes Tag fehlt) in den Typ `[Xml]` zu konvertieren, quittiert die PowerShell mit einem Fehler (siehe *Abbildung 29.7*).

```
PowerShell - Holger Schwichtenberg (www.IT-Visions.de) - [Running as Administrator] - C:\WINDOWS
Windows PowerShell
Copyright (C) 2006 Microsoft Corporation. All rights reserved.

1# [XML] (Get-Content h:\demo\powershell\xml\websites_ungueltig.xml)
Cannot convert value "System.Object[]" to type "System.Xml.XmlDocument". Error:
 "The 'Website' start tag on line 19 does not match the end tag of 'Websites'.
Line 26, position 3."
At line:1 char:6
+ [XML]  <<<< (Get-Content h:\demo\powershell\xml\websites_ungueltig.xml)
2# _
```

Abbildung 29.7
Fehlermeldung, wenn ein schließendes Tag fehlt

Mit dem Commandlet `Test-Xml` (aus den PSCX) kann man vorher prüfen, ob ein Dokument gültig ist. `Test-Xml` liefert `True` oder `False`.

```
Test-Xml h:\demo\powershell\xml\websites_ungueltig.xml
```

`Test-Xml` prüft im Standard nur die Gültigkeit. Optional ist eine Validierung gegen ein XML-Schema möglich. Hierbei ist nach `-SchemaPath` der Pfad zu der XML-Schema-Datei (*.xsd*) anzugeben. Alternativ kann dort auch ein Array mit mehreren Pfaden angegeben werden.

```
Test-Xml h:\demo\powershell\xml\websites.xml -SchemaPath h:\demo\
powershell\xml\websites.xsd
```

Abbildung 29.8
XML-Schema für die Websites-Datei

Websites.xsd*

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified" xmlns:
   xs="http://www.w3.org/2001/XMLSchema">
3   <xs:element name="Websites">
4    <xs:complexType>
5     <xs:sequence>
6      <xs:element maxOccurs="unbounded" name="Website">
7       <xs:complexType>
8        <xs:sequence>
9         <xs:element name="URL" type="xs:string" />
10        <xs:element name="Beschreibung" type="xs:string" />
11       </xs:sequence>
12       <xs:attribute name="ID" type="xs:unsignedByte" use="required" />
13      </xs:complexType>
14     </xs:element>
15    </xs:sequence>
16   </xs:complexType>
17  </xs:element>
18 </xs:schema>
```

Formatierte Ausgabe

Format-Xml

XML-Dokumente müssen nicht formatiert sein, d.h., Einrückungen der XML-Elemente entsprechend der Ebene sind nicht notwendig. In den PSCX gibt es eine Möglichkeit, mit dem Commandlet `Format-Xml` nicht formatierte XML-Dokumente formatiert auszugeben bzw. die Formatierung der Ausgabe anzupassen.

PowerShell - Holger Schwichtenberg (www.IT-Visions.de) - [Running as Administrator] - C:\WINDOWS

```
8# Format-Xml h:\demo\powershell\xml\websites.xml -IndentString ".   "
<?xml version="1.0" encoding="utf-8"?>
<Websites>
.   <Website ID="1">
.   .   <URL>www.dotnetframework.de</URL>
.   .   <Beschreibung>Community-Site für .NET-Entwickler</Beschreibung>
.   </Website>
.   <Website ID="2">
.   .   <URL>www.Windows-Scripting.de</URL>
.   .   <Beschreibung>Community-Site für WSH- und PowerShell-Entwickler</Beschreibung>
.   </Website>
.   <Website ID="3">
.   .   <URL>www.powershell-doktor.de</URL>
.   .   <Beschreibung>Community-Site zur Microsoft PowerShell</Beschreibung>
.   </Website>
.   <Website ID="4">
.   .   <URL>www.aspnetdev.de</URL>
.   .   <Beschreibung>Community-Site für ASP.NET-Entwickler</Beschreibung>
.   </Website>
.   <Website ID="5">
.   .   <URL>www.komponenten.info</URL>
.   .   <Beschreibung>Softwarekomponentenkatalog</Beschreibung>
.   </Website>
.   <Website ID="6">
.   .   <URL>www.IT-Visions.de</URL>
.   .   <Beschreibung>Websites des Autors</Beschreibung>
.   </Website>
</Websites>
9#
```

***Abbildung 29.9**: Einsatz von Format-Xml*

Der folgende Befehl liefert eine formatierte Ausgabe eines XML-Dokuments, bei der jede Ebene mit einem Punkt und vier Leerzeichen eingerückt wird (siehe *Abbildung 29.9*).

```
Format-Xml h:\demo\powershell\xml\websites.xml -IndentString ".
"
```

XPath-Anweisungen

Select-Xml

Zur Suche in XML-Dokumenten mit Hilfe von XPath (XPath ist ein W3C-Standard, siehe [W3C01]) unterstützt die Klasse `XmlDocument` die Methoden `SelectNodes()` und `SelectSingleNode()`. In den PSCX gibt es das Commandlet `Select-Xml`.



`SelectNodes()` und `SelectSingleNode()` liefern Instanzen der Klassen `System.Xml.Xml-Element` und `System.Xml.XmlAttribute`. `Select-Xml` hingegen liefert Instanzen von `MS.Internal.Xml.Cache.XPathDocument-Navigator`. Die Ausgabe ist daher sehr verschieden. Um bei beiden Befehlen zur gleichen Ausgabe zu kommen, muss man das Ergebnis von `Select-Xml` an `Select-Object InnerXml` senden (siehe *Abbildung 29.10*).

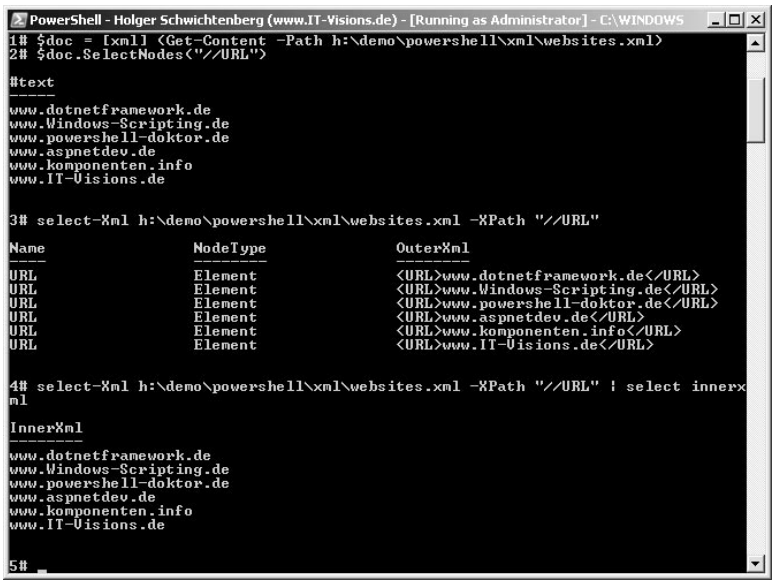


Abbildung 29.10
Ausgabe von SelectNodes() und Select-Xml im Vergleich

Es folgen einige Beispiele.

`$doc.SelectNodes("//URL")`
Oder
`select-Xml h:\demo\powershell\xml\websites.xml -XPath "//URL" | select innerxml`
Liefert alle <URL>-Elemente.

`$doc.SelectNodes("//Website/@ID")`
Oder
`select-Xml h:\demo\powershell\xml\websites.xml -XPath "//Website/@ID" | select innerxml`
Liefert alle ID-Attribute aller <Website>-Elemente.

`$doc.SelectSingleNode("//Website[@ID=3]/URL")`
Oder
`select-Xml h:\demo\powershell\xml\websites.xml -XPath "//Website[@ID=3]/URL" | select innerxml`
Liefert das <URL>-Element des <Website>-Elements mit dem Attributwert 3 im Attribut ID.

***Tabelle 29.1**: Beispiele zur Anwendung von XPath*

`Select-Xml` hat den Vorteil, dass dort auch eine einfache Unterstützung für XML-Namensräume geboten wird. Der folgende Befehl liest aus einer Visual-Studio-Projektdatei die Namen aller eingebundenen C#-Quellcodedateien aus. Dabei muss Bezug genommen werden auf den entsprechenden Namensraum des Kommandozeilenwerkzeugs MSBuild, das für die Übersetzung der Projekte zuständig ist.

```
Select-Xml "H:\demo\PowerShell\_Eigene Commandlets\PowerShell_
Commandlet_Library\PowerShell_Commandlet_Library.csproj"  -
Namespace 'dns=http://schemas.microsoft.com/developer/msbuild/2003'
-XPath "//dns:Compile/@Include"
```

Beispiel

***Abbildung 29.11** Dieses Fragment aus der Visual-Studio-Projektdatei zeigt die zu selektierenden Elemente und deren Namensraumdeklaration.*

```
<Project DefaultTargets="Build" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  ...
  <ItemGroup>
    <Compile Include="Test-Dauer.cs" />
    <Compile Include="Get-Disk3.cs" />
    <Compile Include="Get-Computername.cs" />
    <None Include="Get-Disk2.cs" />
    <None Include="Get-Disk1.cs" />
    <Compile Include="Properties\AssemblyInfo.cs" />
    <Compile Include="PSSnapin.cs">
      <SubType>Component</SubType>
    </Compile>
  </ItemGroup>
...
</Project>
```

XML-Dateien verändern

Das nächste Skript ergänzt einen Eintrag in einer XML-Datei unter Verwendung der Methoden `CreateElement()` und `AppendChild()`. Dieses Beispiel zeigt, dass es aber auch Ecken in der PowerShell gibt, die etwas komplizierter sein können. Weil die Unterelemente eines XML-Knotens als Attribute der .NET-Klasse, welche die PowerShell verarbeitet, dargestellt werden, können – zur Vermeidung von Namenskonflikten – die Attribute der Metaklasse `System.Xml.Node` (bzw. abgeleiteter Klassen) nicht mehr direkt dargestellt werden. Diese Attribute sind nur über ihre Getter und Setter verfügbar. Dies bedeutet, dass man mit dem PowerShell-Skript den Inhalt eines Knotens nicht über `$knoten.Innertext = "xyz"` setzen kann, sondern etwas umständlicher über `$knoten. _set_Innertext("xyz")` aufrufen muss.

Listing 29.5
Ergänzen einer XML-Datei [XML_Modify.ps1]

```
"Vorher"
$doc = [xml] (Get-Content -Path h:\demo\buch\websites.xml)
$doc.Websites.Website | select URL,Beschreibung
"Nachher"
$site = $doc.CreateElement("Website")
$url =  $doc.CreateElement("URL")
$url.set_Innertext("www.powershell-doktor.de")
$beschreibung =  $doc.CreateElement("Beschreibung")
$beschreibung.set_Innertext("Community-Website zur PowerShell")
$site.AppendChild($url)
$site.AppendChild($Beschreibung)
$doc.Websites.AppendChild($site)
$doc.Websites.Website | select URL,Beschreibung
$doc.Save("h:\demo\buch\websites_neu.xml")
"Dokument gespeichert!"
```

XML-Dateien aus Pipeline exportieren

Export-Datei für PowerShell-Pipeline-Objekt in XML-Form (Export-CliXml)

Die PowerShell verwendet ein eigenes XML-Format („CLIXML"), mit dem die Objekt-Pipeline in XML-Form (durch `Export-Cl iXml`) persistiert (serialisiert) werden kann, so dass diese später wiederhergestellt werden kann. Der folgende Befehl speichert die Objektliste der laufenden Systemdienste. Die Abbildung zeigt das Ergebnis.

```
Get-Service | Where-Object {$_.status -eq "running"} | Export-
CliXml j:\demo\dokumente\dienste.xml
```

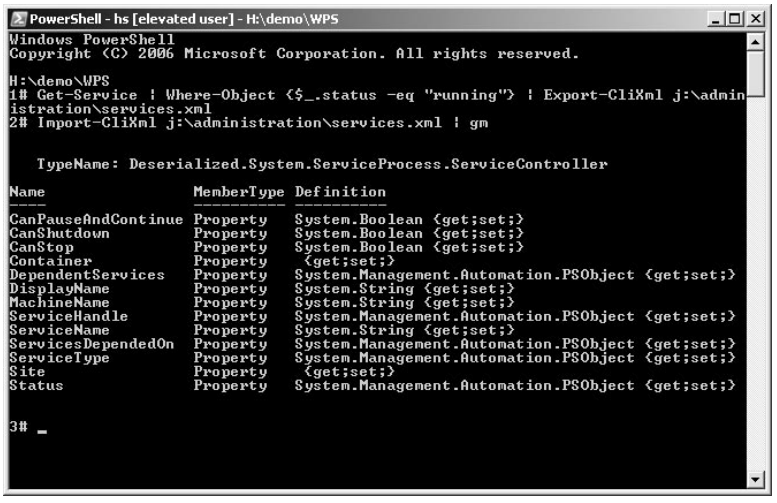
Abbildung 29.12
Ausschnitt aus der Serialisierung einer PowerShell-Pipeline



Das Gegenstück zur Wiederherstellung der Pipeline ist `Import-CliXml`.

```
Import-CliXml j:\demo\dokumente\dienste.xml | Get-Member
```

Abbildung 29.13
Deserialisierte Dienst-Objekte



Nach der Deserialisierung der Objekte können alle Attribute der Objekte wieder verwendet werden, nicht aber die Methoden der Objekte!

ConvertTo-XML

`ConvertTo-XML` (ab PowerShell 2.0) erzeugt genau wie `Export-CliXml` ein XML-Dokument aus einer Objektmenge. Der Unterschied ist aber, dass ConvertTo-XML die XML-Ausgabe nicht in ein Dokument schreibt, sondern innerhalb der PowerShell in drei Formen zur Weiterverarbeitung anbietet: Zeichenkette (`-as string`), Stream (`-as stream`) oder XML-Dokument (`-as document`) in Form einer Instanz von `System.Xml.XmlDocument`.

Hinweis: Im Test mit Windows 7 RC1 und Windows Server 2008 RC liefert die Angabe der Option „-as stream“ immer das gleiche Ergebnis wie „-as string“.

```
PS C:\Users\HS> get-service bits | convertto-XML -as stream
<?xml version="1.0"?>
<Objects>
<Object Type="System.ServiceProcess.ServiceController">
  <Property Name="Name" Type="System.String">bits</Property>
  <Property Name="RequiredServices" Type="System.ServiceProcess.ServiceController[]">
    <Property Type="System.ServiceProcess.ServiceController">System.ServiceProcess.ServiceController</Property>
    <Property Type="System.ServiceProcess.ServiceController">System.ServiceProcess.ServiceController</Property>
  </Property>
  <Property Name="CanPauseAndContinue" Type="System.Boolean">False</Property>
  <Property Name="CanShutdown" Type="System.Boolean">False</Property>
  <Property Name="CanStop" Type="System.Boolean">True</Property>
  <Property Name="DisplayName" Type="System.String">Background Intelligent Transfer Service</Property>
  <Property Name="DependentServices" Type="System.ServiceProcess.ServiceController[]" />
  <Property Name="MachineName" Type="System.String">.</Property>
  <Property Name="ServiceName" Type="System.String">bits</Property>
  <Property Name="ServicesDependedOn" Type="System.ServiceProcess.ServiceController[]">
    <Property Type="System.ServiceProcess.ServiceController">System.ServiceProcess.ServiceController</Property>
    <Property Type="System.ServiceProcess.ServiceController">System.ServiceProcess.ServiceController</Property>
  </Property>
  <Property Name="ServiceHandle" Type="SafeServiceHandle">SafeServiceHandle</Property>
  <Property Name="Status" Type="System.ServiceProcess.ServiceControllerStatus">Running</Property>
  <Property Name="ServiceType" Type="System.ServiceProcess.ServiceType">Win32ShareProcess</Property>
  <Property Name="Site" Type="System.ComponentModel.ISite" />
  <Property Name="Container" Type="System.ComponentModel.IContainer" />
</Object>
</Objects>
PS C:\Users\HS>
```

Abbildung 29.14 Ausgabe von ConvertTo-Xml bei Anwendung auf einer Instanz von System.ServiceProcess.ServiceController

XML-Dateien transformieren

Für die Anwendung des W3C-Standards XSLT (XML Stylesheet Transformation) steht in den PSCX das Commandlet `Convert-Xml` zur Verfügung. Alternativ kann man die .NET-Klasse `System.Xml.Xsl.XslCompiledTransform` verwenden.

Das folgende Beispiel zeigt, wie man die XML-Datei *Websites.xml* mit Hilfe der in der Abbildung gezeigten XSLT-Datei in eine XHTML-Datei konvertieren kann. Das Ergebnis wird gespeichert als *Websites.html*.

```
Convert-Xml h:\demo\powershell\xml\websites.xml -XsltPath H:\DEV\
ITVisions_PowerShell_CommandletLibrary\CommandletLibrary\Daten\
   WebsitesToHTML.xslt | Set-Content h:\demo\powershell\xml\
websites.html
```

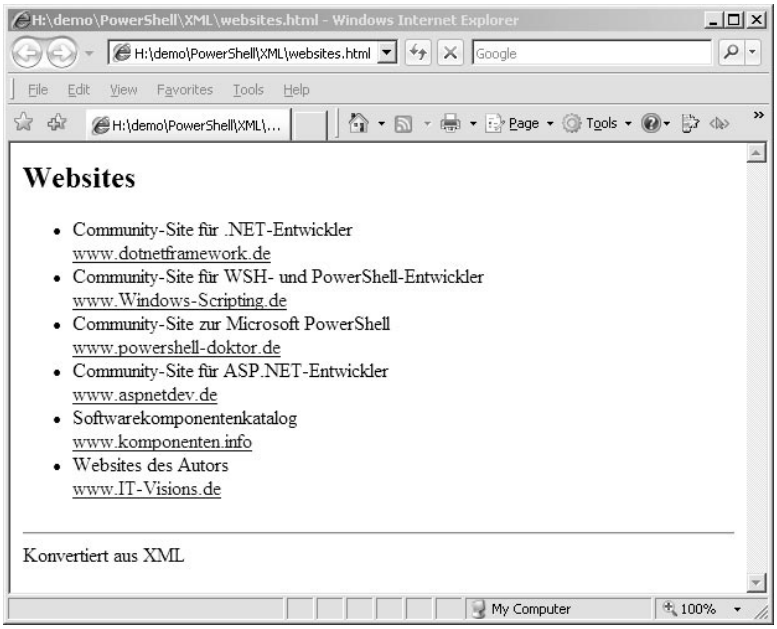
Hilfe beim Entwickeln und Testen von XSLT-Dateien bietet Ihnen Visual Studio 2005/2008/2010.

Abbildung 29.15
XSLT-Datei

WebsitesToHTML.xslt

```
<?xml version="1.0" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
 <!-- Transformation -->
 <xsl:template match="Websites">
  <HTML>
   <body>
    <h2>Websites von Dr. Holger Schwichtenberg</h2>
    <ul>
     <xsl:for-each select="/Websites/Website">
      <li>
       <xsl:value-of select='Beschreibung'/>
       <br>
        <a>
         <xsl:attribute name="href">
          <xsl:value-of select="URL"/>
         </xsl:attribute>
         <xsl:value-of select="URL"/>
        </a>
       </br>
      </li>
     </xsl:for-each>
    </ul>
    <hr></hr>
    Konvertiert aus XML
   </body>
  </HTML>
 </xsl:template>
</xsl:stylesheet>
```

Abbildung 29.16
Diese HTML-Datei wurde aus der XML-Datei generiert.



29.6 HTML-Dateien

Convert-Html

Das Commandlet `Convert-Html` konvertiert die Objekte der Pipeline in eine HTML-Tabelle.

Der folgende Befehl speichert die Liste der Windows-Systemdienste als eine HTML-Datei.

```
Get-Service | ConvertTo-Html name,status -title "Diensteliste" -
body "Liste der Dienste" | Set-Content j:\demo\dokumente\
dienste.htm
```

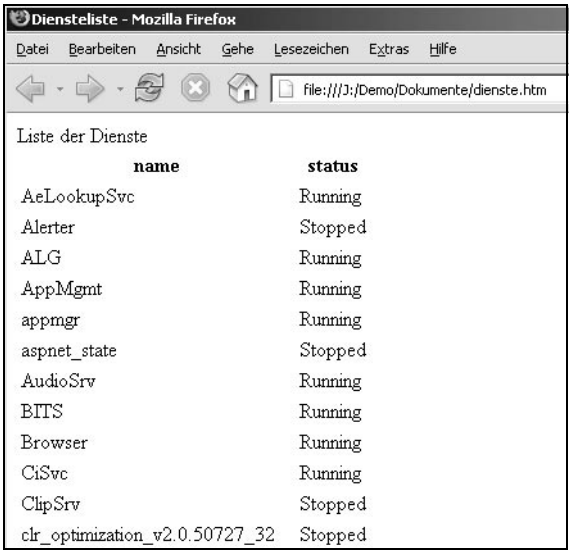


Abbildung 29.17
Ergebnis der Konvertierung in eine HTML-Tabelle

30 Datenbanken

ADO.NET

Die PowerShell besitzt keine Commandlets für den Datenbankzugriff und auch keinen Navigation Provider, obwohl es sich anbieten würde, auch Datenbanken ins Konzept der Navigation Provider einzubeziehen. Zum Datenbankzugriff kann man in der PowerShell auf ADO.NET zugreifen. Immerhin unterstützt die PowerShell beim Zugriff auf die einzelnen Tabellen, indem sie die Spaltennamen als Attribute des Tabellenobjekts anbietet (hier findet eine ähnliche automatische Abbildung statt wie bei WMI-Objekten).

Erst der SQL Server 2008 bringt einen eigenen Provider für die Arbeit mit dem Server und den Datenbanken mit.



ADO.NET ist die Weiterentwicklung der COM-Komponente ActiveX Data Objects (ADO).

Im Folgenden werden zwei Wege gezeigt:

- Zuerst der direkte Zugriff auf ADO.NET
- Anschließend die Nutzung der Commandlets aus den *www.IT-Visions.de-PowerShell Extensions*, die den Zugriff kapseln

30.1 ADO.NET-Grundlagen

Dieses Kapitel vermittelt einige notwendige Grundlagen zu ADO.NET.

Providerarchitektur

Provider

Genauso wie die Vorgängerkonzepte ODBC und OLEDB verwendet ADO.NET auch datenquellenspezifische Treiber, die ADO.NET Data Provider, .NET Data Provider oder Managed Provider genannt werden. Data Provider für OLEDB und ODBC stellen dabei die Abwärtskompatibilität von ADO.NET für Datenquellen her, für die (noch) keine spezifischen ADO.NET-Datenprovider existieren.

Datenprovider von Microsoft

Mitgelieferte Provider

ADO.NET 2.0 wird mit folgenden Datenprovidern (alias .NET Data Provider oder Managed Data Provider) ausgeliefert:

- `System.Data.SqlClient` (spezieller Treiber für Microsoft SQL Server 7.0/2000 und 2005; dieser Treiber wird auch innerhalb des SQL Server 2005 für Managed-Code-Anwendungen benutzt und löst `System.Data.SqlServer` der vorherigen Beta-Versionen ab)
- `System.Data.SqlServerCe` (spezieller Treiber für Microsoft SQL Server CE)
- `System.Data.OracleClient` (spezieller Treiber für Oracle-Datenbanken)
- `System.Data.OLEDB` (Brücke zu OLEDB-Providern)
- `System.Data.Odbc` (Brücke zu ODBC -Treibern)

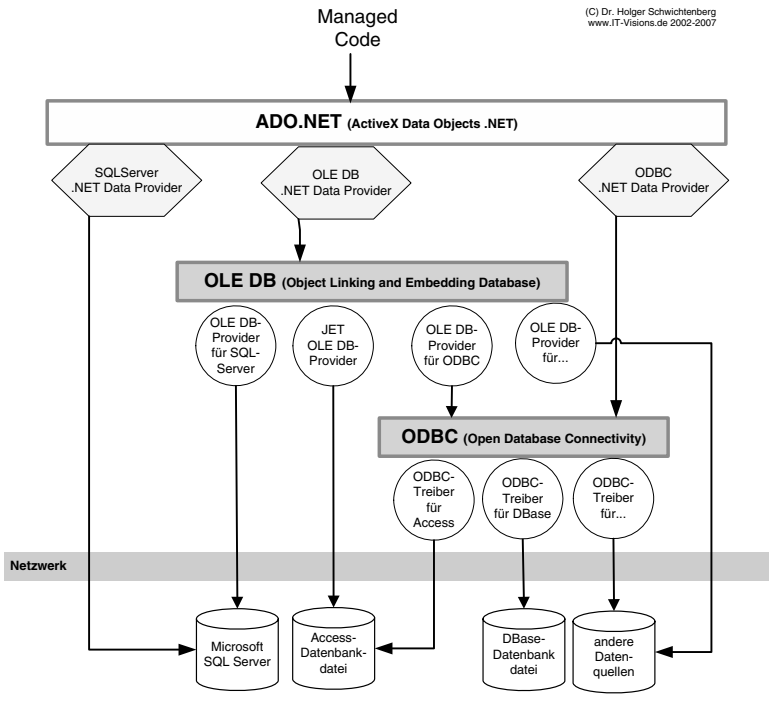


Abbildung 30.1 *ADO.NET-Treiberarchitektur*

Datenprovider von anderen Herstellern

Andere Anbieter

Weitere Provider (z.B. für MySQL, DB2, Sybase, Informix und Ingres) werden von anderen Herstellern geliefert, eine Liste finden Sie unter [DOTNET02].

Ermittlung der installierten Datenprovider

DbProviderFactories

Die auf einem System registrierten ADO.NET-Datenprovider können über die statische Methode `System.Data.Common.DbProviderFactories.GetFactoryClasses()` aufgelistet werden.

Der Zugriff in der PowerShell sieht so aus:

```
[System.Data.Common.DbProviderFactories]::GetFactoryClasses()
```



Die installierten Provider sind nicht in der Registrierungsdatenbank, sondern – wie es sich für eine .NET-Anwendung gehört – in der zentralen XML-Konfigurationsdatei des .NET Frameworks (`machine.config`) abgelegt (Sektion `<system.data> <DbProviderFactories>`).

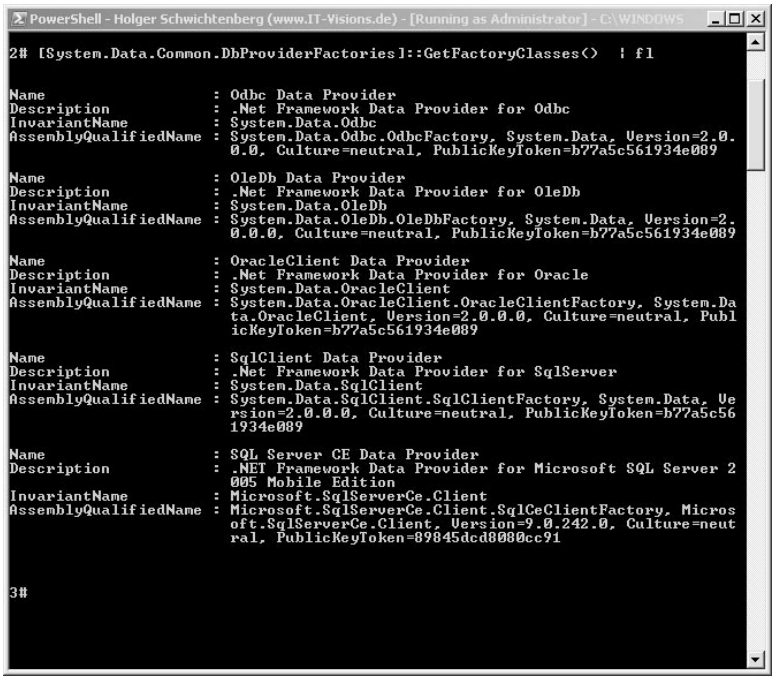


Abbildung 30.2
Auflisten der installierten ADO.NET-Treiber

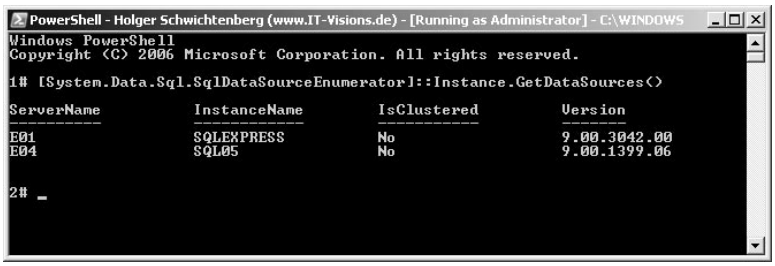
Liste der verfügbaren SQL Server

SqlDataSourceEnumerator

Wenn Sie wissen wollen, welche Instanzen von Microsoft SQL Server in Ihrer Domäne in Betrieb sind, können Sie die .NET-Klasse `SqlDataSourceEnumerator` verwenden:

```
[System.Data.Sql.SqlDataSourceEnumerator]::Instance.GetDataSources()
```

Abbildung 30.3
Liste der verfügbaren SQL Server



Datenwege

Architektur

Die nachstehende Abbildung zeigt die möglichen Datenwege in ADO.NET 2.0 von einer Datenquelle zu einem Datenverbraucher. Alle Zugriffe auf eine Datenquelle laufen auf jeden Fall über ein `Command`-Objekt, das datenproviderspezifisch ist. Zum Auslesen von Daten bietet das Modell zwei Wege: Daten können über ein providerspezifisches `DataReader`-Objekt oder über ein providerunabhängiges `DataSet`-Objekt zum Datenverbraucher gelangen. Das `DataSet`-Objekt benötigt zur Beschaffung der Daten ein `DataAdapter`-Objekt (nicht zu verwechseln mit einem PowerShell-Objektadapter), das wiederum in jedem Datenprovider separat zu implementieren ist.

Ab .NET 2.0 existieren Möglichkeiten, nachträglich noch von einem in das andere Zugriffsmodell zu wechseln. Datenänderungen erfolgen, indem der Datenverbraucher direkt Befehle an ein `Command`-Objekt sendet.

Ab .NET 2.0 stellt .NET sogenannte Datenquellensteuerelemente bereit, die dem Entwickler die Bindung von Daten an ein Steuerelement erleichtern. Diese Datenquellensteuerelemente sind Teil der Bibliotheken für grafische Benutzeroberflächen (Windows Forms und ASP.NET) und werden in diesem Buch nicht behandelt. Dazu sei auf [SCH02] verwiesen.



Es ist möglich, aber etwas aufwändiger, den Zugriff auf eine Datenquelle so zu programmieren, dass die Art der Datenbank ausgetauscht werden kann.

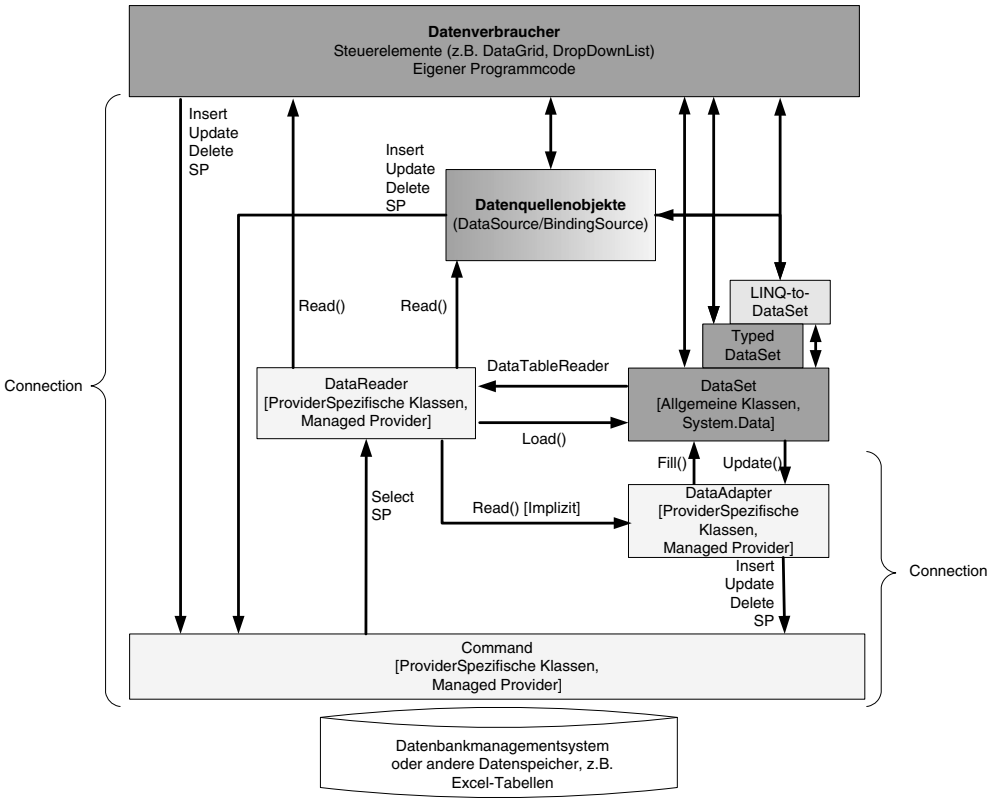


***Abbildung 30.4**: Datenwege in ADO.NET*

Datareader versus Dataset

Bei der Beschreibung der Datenwege wurde zwischen Datareader und Dataset unterschieden. Die folgende Tabelle und die Grafik vergleichen die beiden Zugriffsverfahren im Detail.

	Datareader	Dataset
Modell	Server Cursor	Client Cursor
Implementiert in	Jedem Datenprovider	`System.Data`
Basisklassen	`DbDataReader MarshalByRefObject Object`	`MarshalByValue-ComponentObject`
Schnittstellen	`IDataReader, IDisposable, IDataRecord, IEnumerable`	`IListSource, IXmlSerializable, ISupportInitialize, ISerializable`

***Tabelle 30.1**: Datareader vs. Dataset*

	Datareader	Dataset
Daten lesen	Ja	Ja
Daten vorwärts lesen	Ja	Ja
Daten rückwärts lesen	Nein	Ja
Direktzugriff auf beliebigen Datensatz	Nein	Ja
Direktzugriff auf beliebige Spalte in Datensatz	Ja	Ja
Daten verändern	Nein, nur über separate `Command`-Objekte	Ja (über Datenadapter)
Befehlserzeugung für Datenänderung	Komplett manuell	Teilweise automatisch (CommandBuilder)
Zwischenspeicher für Daten	Nein	Ja
Änderungshistorie	Nein	Ja
Speicherverbrauch	Niedrig	Hoch
Geeignet für Datentransport zwischen Schichten	Nein	Ja

***Tabelle 30.1**: Datareader vs. Dataset (Forts.)*



Es gibt im .NET Framework (bisher) keinen schreibenden Cursor.

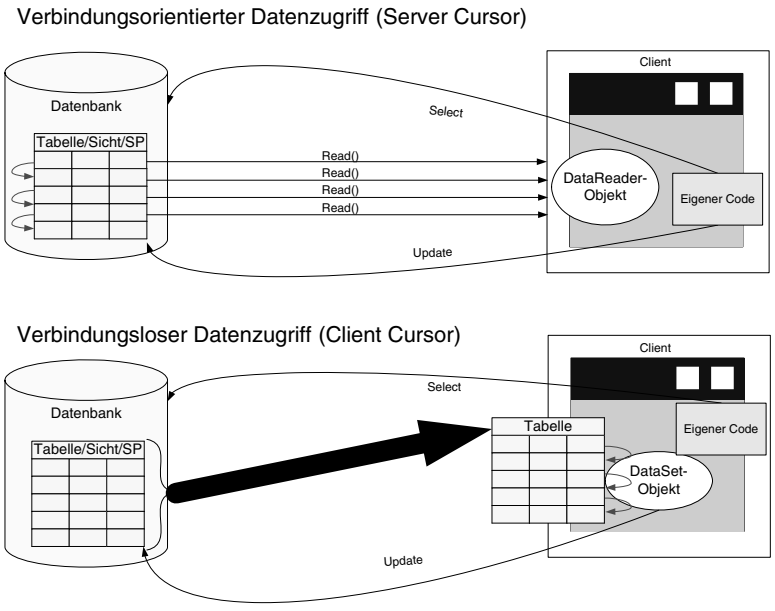


***Abbildung 30.5** Vergleich von Datareader und Dataset*

30.2 Beispieldatenbank

Benutzertabelle

Die Beispieldatenbank ist aus dem Leben der Systemadministration gegriffen, denn sie enthält eine Liste von Benutzerkonten, die entweder aus einem Windows-System exportiert wurde oder die dazu dienen kann, eine Reihe von Benutzern per Skript anzulegen. Hinweis: Der Inhalt der Datenbank ist eine Liste von (zum Teil schon etwas „angestaubten“) Politikern.

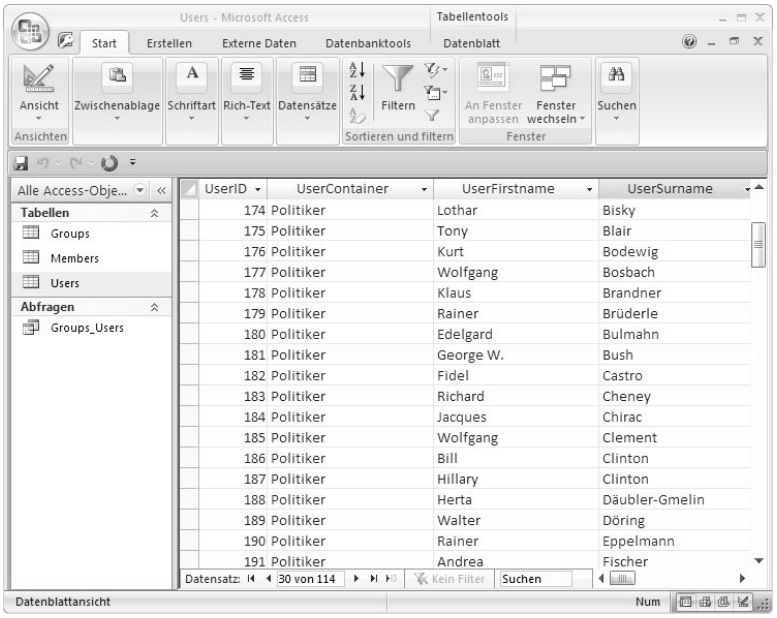


Abbildung 30.6
Datenbank mit Benutzerkonten



Da es keinen 64-Bit-Treiber für Access-Datenbanken gibt, müssen Sie alle PowerShell-Befehle, die auf Access-Datenbanken zugreifen, auf einem 64-Bit-System in der 32-Bit-Variante der PowerShell laufen lassen (siehe *%Systemroot\Syswow64\WindowsPowerShell\V1.0*).

30.3 Datenzugriff mit den Bordmitteln der PowerShell

Zunächst wird die Herstellung einer Verbindung behandelt. Danach folgt der Datenzugriff.

Datenbankverbindungen (Connection)

Verbindungsobjekt

Egal, welche Datenzugriffsform gewählt wird, und egal, welche Aktion ausgeführt werden soll: Für die Kommunikation mit dem Datenbankmanagementsystem ist immer eine Verbindung notwendig.

Jeder Datenprovider hat eine eigene Implementierung für die Verbindungsklasse: `Sql-Connection`, `OracleConnection`, `OleDbConnection` usw. Bei der Instanziierung dieser Objekte kann die Verbindungszeichenfolge übergeben werden. Danach erfolgt der Aufruf von `Open()`. Eine Verbindung muss geschlossen werden durch `Close()`.

Beispiele

Die folgenden drei Beispiele zeigen jeweils den Verbindungsaufbau und -abbau zu drei verschiedenen Arten von Datenbanken:

- Microsoft-Access-Datenbankdatei
- Statische eingebundene Microsoft-SQL-Server-Datenbank
- Microsoft-SQL-Server-Datenbankdatei (funktioniert nur mit Microsoft SQL Server Express)

Listing 30.1 *Verbindung zu einer Microsoft-Access-Datenbank aufbauen und schließen [Connection.ps1]*

```
# Eingabedaten
$Conn = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=H:\demo\
PowerShell\Datenbanken\users.mdb;"
$SQL = "Select * from users order by UserSurname"

# Datenbank oeffnen
"Oeffne die Datenbank..."
$conn = New-Object System.Data.OleDb.OleDbConnection($Conn)
$conn.open()
"Zustand der Datenbank: " + $conn.State

# Datenbank schließen
$Conn.Close()
"Zustand der Datenbank: " + $conn.State
```

Listing 30.2 *Verbindung zu einer statisch verbundenen Microsoft-SQL-Server-Datenbank aufbauen und schließen [Connection.ps1]*

```
# Eingabedaten
$Connstring = "Data Source=.\SQLEXPRESS;Initial
catalog=Users;Integrated Security=True;"
$SQL = "Select * from users order by UserSurname"

# Datenbank oeffnen
"Oeffne die Datenbank..."
$conn = New-Object
System.Data.SqlClient.SqlConnection($Connstring)
```

```
    $conn.open()
    "Zustand der Datenbank: " + $conn.State

    # Datenbank schließen
    $Conn.Close()
    "Zustand der Datenbank: " + $conn.State
```

Listing 30.3
Verbindung zu einer dynamisch verbundenen Microsoft-SQL-Server-Express-Datenbankdatei aufbauen und schließen [Connection.ps1]

```
# Eingabedaten
$Connstring = "Data Source=.\SQLEXPRESS;AttachDbFileName=H:\demo\
PowerShell\Datenbanken\users.mdf;Integrated Security=True;"
$SQL = "Select * from users order by UserSurname"

# Datenbank oeffnen
"Oeffne die Datenbank..."
$conn = New-Object System.Data.SqlClient.SqlConnection($Connstring)
$conn.open()
"Zustand der Datenbank: " + $conn.State

# Datenbank schließen
$Conn.Close()
"Zustand der Datenbank: " + $conn.State
```

Providerunabhängiger Zugriff

DbProviderFactories

In den bisherigen Beispielen kamen verschiedene Klassen vor in Abhängigkeit davon, welcher Datenbankprovider (Microsoft Access oder Microsoft SQL Server) verwendet wurde. Dies ist unschön, wenn man auf verschiedene Datenbanken zugreifen muss oder die Datenbank später einmal wechseln möchte. ADO.NET unterstützt auch den providerunabhängigen Datenzugriff.

Beim providerunabhängigen Datenzugriff instanziiert man die Verbindungsklasse nicht direkt, sondern über eine sogenannte Providerfabrik. Die Providerfabrik erhält man von der .NET-Klasse `System.Data.Common.DbProviderFactories` unter Angabe des sogenannten „Provider Invariant Name“ als Zeichenkette. Dieser ist:

- Für Microsoft Access: „System.Data.OleDb“
- Für Microsoft SQL Server: „System.Data.SqlClient“
- Für Oracle: „System.Data.OracleClient“



Bei dem providerunabhängigen Datenzugriff findet keine Übersetzung von SQL-Befehlen statt. Wenn Sie datenbankmanagementsystemspezifische Befehle nutzen, verlieren Sie die Providerunabhängigkeit.

Listing 30.4 Provider-unabhängiger Verbindungsaufbau [/Datenbanken/Connection.ps1]

```
# Eingabedaten
$PROVIDER = "System.Data.SqlClient"
$CONNSTRING = "Data Source=.\SQLEXPRESS;AttachDbFileName=
   H:\demo\PowerShell\Datenbanken\users.mdf;Integrated
Security=True;"
$SQL = "Select * from FL_Fluege"

# Fabrik erzeugen
$provider =
[System.Data.Common.DbProviderFactories]::GetFactory($PROVIDER)

# Verbindungsobjekt erstellen und befüllen
$conn = $provider.CreateConnection()
$conn.ConnectionString = $CONNSTRING;

# Verbindung aufbauen
$conn.Open();
"Zustand der Datenbank: " + $conn.State

# Datenbank schließen
$Conn.Close()
"Zustand der Datenbank: " + $conn.State
```

Befehle ausführen

Command

Jeder Datenbankprovider stellt ein providerspezifisches Befehlsobjekt (`SqlCommand`, `OracleCommand`, `OleDbCommand` usw.) zur Verfügung. Darüber hinaus gibt es ein providerneutrales Befehlsobjekt vom Typ `DbCommand`.

Das Befehlsobjekt bietet folgende Funktionen an:

- `ExecuteNonQuery()` zur Ausführung von DML- und DDL-Befehlen, die keine Datenmenge zurückliefern. Sofern die Befehle die Anzahl der betroffenen Zeilen zurückliefern, steht diese Zahl im Rückgabewert der Methode. Sonst ist der Wert –1.
- `ExecuteRow()` liefert die erste Zeile der Ergebnismenge in Form eines `SqlRecord`-Objekts (nur SQL Server).
- `ExecuteScalar()` liefert nur die erste Spalte der ersten Zeile der Ergebnismenge.
- `ExecuteReader()` liefert ein Datareader-Objekt (siehe nächster Abschnitt).

Über Providerfabriken kann mit dem Befehlsobjekt auch providerunabhängig gearbeitet werden, wie das nachstehende Beispiel zeigt. Hierbei ist das Befehlsobjekt von der Providerfabrik über `CreateCommand()` zu erzeugen.

Beispiel

In dem Beispiel wird erst die Anzahl der Benutzer gezählt, dann wird ein neuer Benutzer angelegt, dann wieder gezählt. Zum Schluss wird der angelegte Benutzer wieder gelöscht und erneut gezählt.

Listing 30.5
Ausführung von Befehlen mit providerunabhängigen Befehlsobjekten [/Datenbanken/Command.ps1]

```
# Parameters
$PROVIDER = "System.Data.SqlClient"
$CONNSTRING = "Data Source=.\SQLEXPRESS;AttachDbFileName=
   H:\demo\PowerShell\Datenbanken\users.mdf;Integrated
Security=True;"
$SQL1 = "Select count(*) from users"
$SQL2 = "insert into users ( UserFirstName, UserSurname) values
('Hans', 'Meier')"
$SQL3 = "delete from users where UserSurname='Meier'"

# Create factory
$provider =
[System.Data.Common.DbProviderFactories]::GetFactory($PROVIDER)

# Create connection object
$conn = $provider.CreateConnection()
$conn.ConnectionString = $CONNSTRING

# Open connection
$conn.Open();
"Database Connection State: " + $conn.State

# create command #1
[System.Data.Common.DbCommand] $cmd1 = $provider.CreateCommand()
$cmd1.CommandText = $SQL1
$cmd1.Connection = $conn
# execute command #1
$e = $counter = $cmd1.ExecuteScalar()
"Count before insert: " + $Counter

# create command #2 (INSERT)
[System.Data.Common.DbCommand] $cmd2 = $provider.CreateCommand()
$cmd2.CommandText = $SQL2
$cmd2.Connection = $conn
# execute command #2
$e = $cmd2.ExecuteNonQuery()
# execute command #1
$counter = $cmd1.ExecuteScalar()
"Count after insert: " + $Counter

# create command #3 (DELETE)
[System.Data.Common.DbCommand] $cmd3 = $provider.CreateCommand()
$cmd3.CommandText = $SQL3
$cmd3.Connection = $conn
# execute command #2
$e = $cmd3.ExecuteNonQuery()

# execute command #1
$counter = $cmd1.ExecuteScalar()
```

```
"Count after delete: " + $Counter

# Datenbank schließen
$Conn.Close()
"Database Connection State: " + $conn.State
```

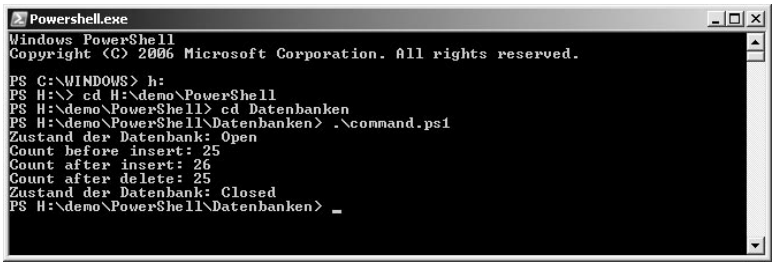


Abbildung 30.7 Ausführung des Skripts Command.ps1

Datenzugriff mit dem Datareader

DataReader

Bei einem `DataReader`-Objekt handelt es sich um einen serverseitigen Cursor, der unidirektionalen Lesezugriff (nur vorwärts) auf das Ergebnis einer `SELECT`-Anwendung (Resultset) erlaubt. Eine Veränderung der Daten ist nicht möglich. Im Gegensatz zum `DataSet` unterstützt der `DataReader` nur eine flache Darstellung der Daten. Die Datenrückgabe erfolgt immer zeilenweise, deshalb muss über die Ergebnismenge iteriert werden. Verglichen mit dem klassischen ADO entspricht ein ADO.NET-`DataReader` einem „read-only/forward-only Recordset" (zu Deutsch: „Vorwärtscursor").

Jeder ADO.NET-Datenprovider enthält seine eigene `DataReader`-Implementierung, so dass es zahlreiche verschiedene `DataReader`-Klassen im .NET Framework gibt (z.B. `Sql-DataReader` und `OLEDBDataReader`). Die `DataReader`-Klassen sind abgeleitet von `System.Data.ProviderBase.Db DataReaderBase` und implementieren `System.Data.IDataReader`.

Ein `DataReader` benötigt zur Beschaffung der Daten ein `Command`-Objekt, das ebenso providerspezifisch ist (z.B. `SqlCommand` und `OLEDBCommand`). Für die Verbindung zur Datenbank selbst wird ein providerspezifisches `Connection`-Objekt (z.B. `SqlConnection` oder `OleDbConnection`) benötigt. Die nachstehenden Abbildungen zeigen den Zusammenhang dieser Objekte am Beispiel der Datenprovider für OLEDB und SQL Server. Bei dem Provider für SQL Server (`SqlClient`) existiert ab .NET 2.0 eine zusätzliche Klasse `SqlRecord`, die einen einzigen Datensatz als Ergebnis eines Befehls repräsentiert.

Der DataReader kann auch providerunabhängig verwendet werden über eine Instanz der Klasse `System.Data.Common.DbDataReader`, die man aus einem providerunabhängigen Befehlsobjekt über `ExecuteReader()` gewinnt.

Beispiel

Das Beispiel liest alle Benutzer aus der Benutzertabelle aus.

Listing 30.6
Auslesen einer Datenbanktabelle mit einem providerunabhängigen Datareader [/Datenbanken/DataReader.ps1]

```
# Eingabedaten
$PROVIDER = "System.Data.SqlClient"
$CONNSTRING = "Data Source=.\SQLEXPRESS;AttachDbFileName=H:\demo\
PowerShell\Datenbanken\users.mdf;Integrated Security=True;"
$SQL = "Select * from users"

# Fabrik erzeugen
$provider =
[System.Data.Common.DbProviderFactories]::GetFactory($PROVIDER)

# Verbindungsobjekt erstellen und befüllen
$conn = $provider.CreateConnection()
$conn.ConnectionString = $CONNSTRING

# Verbindung aufbauen
$conn.Open();
"Zustand der Datenbank: " + $conn.State

# Befehl erzeugen
$cmd = $provider.CreateCommand()
$cmd.CommandText = $SQL
$cmd.Connection = $conn
# Befehl ausführen
$reader = $cmd.ExecuteReader()

# Schleife über alle Datensätze
while($reader.Read())
{
$reader.Item("UserID").ToString() + ": " +
$reader.Item("UserFirstName") + " " + $reader.Item("UserSurname")
}

# Datenbank schließen
$Conn.Close()
"Zustand der Datenbank: " + $conn.State
```

Datenzugriff mit dem Dataset

DataSet

Ein `DataSet` enthält eine Sammlung von Datentabellen, die durch einzelne `DataTable`-Objekte dargestellt werden. Die `DataTable`-Objekte können aus beliebigen Datenquellen gefüllt werden, ohne dass eine Beziehung zwischen dem Objekt und der Datenquelle existiert; das `DataTable`-Objekt weiß nicht, woher die Daten kommen. Die `DataTable`-Objekte können auch ohne Programmcode zeilenweise mit Daten befüllt werden; eine Datenbank ist nicht notwendig. Ein `DataSet` bietet – im Gegensatz zum `DataReader` – alle Zugriffsarten, also auch das Hinzufügen,

Löschen und Ändern von Datensätzen. Ebenfalls lassen sich hierarchische Beziehungen zwischen einzelnen Tabellen darstellen und im `DataSet` speichern. Dadurch ist eine Verarbeitung hierarchischer Datenmengen möglich. Im Untergrund verwendet ein `DataSet` übrigens einen `DatenReader` zum Einlesen der Daten.

Ein Dataset ist ein clientseitiger Datenzwischenspeicher, der die Änderung mitprotokolliert. Das `DataSet` nimmt keine Sperrung von Datensätzen auf der Datenquelle vor, sondern verwendet immer das sogenannte „optimistische Sperren", d.h., Änderungskonflikte treten erst auf, wenn man versucht, die Daten zurückzuschreiben. Das Konzept eines serverseitigen Cursors ist in ADO.NET nur durch die `DataReader`-Klasse realisiert. Einen serverseitigen Cursor mit Schreibfunktion und pessimistischem Sperren gibt es in ADO.NET bisher nicht.



Das Dataset verbraucht sehr viel mehr Speicher als eine selbst definierte Datenstruktur. Das Abholen von Daten mit einem Datareader, das Speichern in einer selbst definierten Datenstruktur und das Speichern von Änderungen mit direkten SQL-Befehlen machen zwar mehr Arbeit bei der Entwicklung, sind aber wesentlich effizienter bei der Ausführung. Dies ist insbesondere bei serverbasierten Anwendungen wichtig.

Objektmodell

DataSet-Objektmodell

Ein `DataSet`-Objekt besteht aus einer Menge von `DataTable`-Objekten (`DataTable-Collection`). Jedes `DataTable`-Objekt besitzt über das Attribut `DataSet` einen Verweis auf das Dataset, zu dem es gehört.

Während die `DataTable`-Objekte in ADO.NET 1.x dem `DataSet`-Objekt noch völlig untergeordnet waren, besitzt die `DataTable`-Klasse in ADO.NET 2.0 viele der Import- und Exportmöglichkeiten, über die auch die `DataSet`-Klasse verfügt.

Das `DataTable`-Objekt besitzt eine `DataColumnCollection` mit `DataColumn`-Objekten für jede einzelne Spalte in der Tabelle und eine `DataRowCollection` mit `DataRow`-Objekten für jede Zeile. Innerhalb eines `DataRow`-Objekts kann man die Inhalte der Zellen durch das indizierte Attribut `Item` abrufen. `Item` erwartet alternativ den Spaltennamen, den Spaltenindex oder ein `DataColumn`-Objekt.

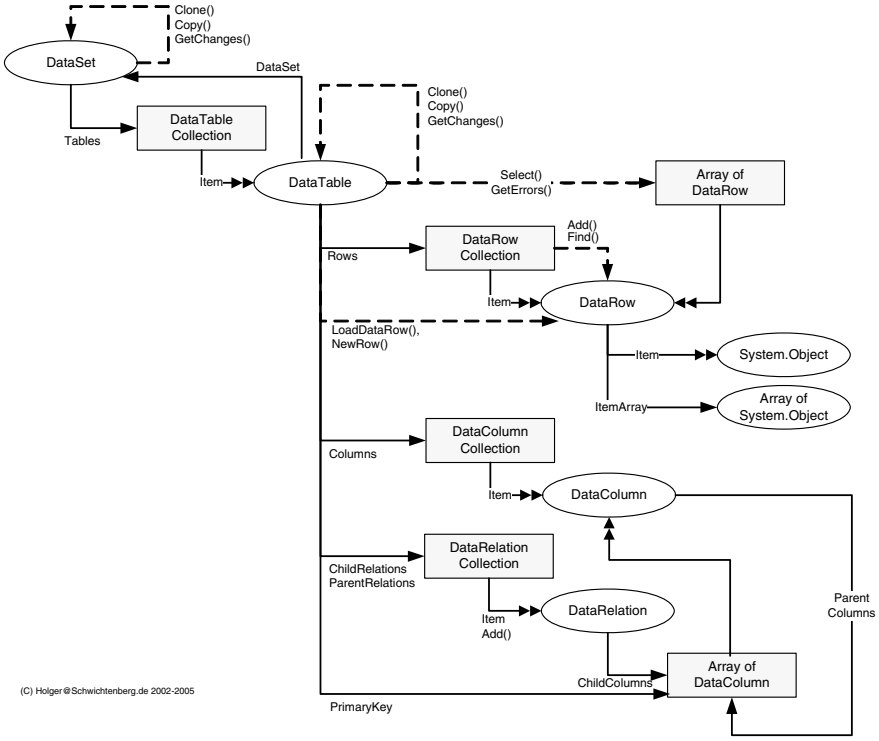


***Abbildung 30.8**: Objektmodell der DataSet-Klasse*

Datenadapter

Ein Dataset benötigt zum Einlesen von Daten einen Datenadapter. Das Lesen von Daten mit einem `DataSet` läuft in folgenden Schritten ab:

- Aufbau einer Verbindung zu der Datenbank mit einem `Connection`-Objekt. Bei der Instanziierung dieses Objekts kann die Verbindungszeichenfolge übergeben werden.
- Instanziierung der Befehlsklasse und Bindung dieses Objekts an das `Connection`-Objekt über die Eigenschaft `Connection`
- Festlegung eines SQL-Befehls, der Daten liefert (also z.B. `SELECT` oder eine Stored Procedure), im `OLEDBCommand`-Objekt in der Eigenschaft `CommandText`
- Instanziierung des Datenadapters auf Basis des `Command`-Objekts
- Instanziierung des `DataSet`-Objekts (ohne Parameter)
- Die Ausführung der Methode `Fill()` in dem `DataSet`-Objekt kopiert die kompletten Daten in Form eines `DataTable`-Objekts in das `DataSet`. Als zweiter Parameter kann bei `Fill()` der Aliasname für das `DataTable`-Objekt innerhalb des `DataSet` angegeben werden. Ohne diese Angabe erhält das `DataTable`-Objekt den Namen Table.

- Optional können weitere Tabellen eingelesen und im `DataSet` miteinander verknüpft werden.
- Danach kann die Verbindung sofort geschlossen werden.

Providerspezifisches Beispiel

Das folgende PowerShell-Skript gibt aus einer Microsoft Access-Datenbanktabelle alle Datensätze sortiert aus. Zum Einsatz kommt dabei der OLEDB-Provider für ADO.NET, der im .NET Framework ab Version 1.0 enthalten ist. Die Implementierung ist providerspezifisch.

Das Skript besteht aus folgenden Schritten:

- Festlegung der Verbindungszeichenfolge und der auszuführenden SQL-Anweisung
- Instanziierung eines Verbindungsobjekts (`OleDbConnection`) mit Hilfe der Verbindungszeichenfolge und Öffnen der Verbindung zur Datenbank
- Erstellen eines Befehlsobjekts (`OleDbCommand`) unter Angabe des Verbindungsobjekts und des SQL-Befehls
- Erstellen eines Datenadapters (`OleDbDataAdapter`) für den Befehl
- Instanziieren eines leeren Datencontainers (`DataSet`) zur Aufnahme der Daten
- Befüllen des Datencontainers durch den Datenadapter mit Hilfe der Methode `Fill()`
- Zugriff auf die erste Tabelle in dem Datencontainer (bitte beachten Sie, dass die Zählung bei 0 beginnt!)
- Ausgabe der Daten durch Pipelining der Tabelle



Es ist nicht möglich, analog zu XML-Dokumenten mit `$Tabelle.Spaltenname` auf die Inhalte der Tabelle zuzugreifen, weil gemäß dem ADO.NET-Objektmodell das `DataTable`-Objekt nicht direkt die Spalten enthält, sondern `DataRow`-Objekte. Die PowerShell enthält aber den Automatismus, beim Pipelining eines `DataTable`-Objekts dieses automatisch in Zeilen und Spalten zu zerlegen. Bei den einzelnen `DataRow`-Objekten funktioniert dann durch die automatische Abbildung der Zugriff auf die Spalten über Ihren Namen, z.B.:

```
$Tabelle | % { $_.Spaltenname }
```

Ebenfalls funktioniert

```
$Tabelle | % { $_["UserSurname"] }
$Tabelle | % { $_."UserSurname" }
```

Diese beiden Syntaxformen braucht man aber nur, falls der Spaltenname ein Leerzeichen enthalten sollte.

```
# Eingabedaten
$Conn = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=h:\demo\
dokumente\users.mdb;"
$SQL = "Select * from users order by UserSurname"

# Datenbank öffnen
"Oeffne die Datenbank..."
$conn = New-Object System.Data.OleDb.OleDbConnection($Conn)
$conn.open()
"Zustand der Datenbank: " + $conn.State

# SQL-Befehl ausführen
"Befehl ausfuehren: " + $SQL
$cmd = New-Object System.Data.OleDb.OleDbCommand($sql,$conn)
$ada = New-Object System.Data.OleDb.OleDbDataAdapter($cmd)
$ds = New-Object System.Data.DataSet
$ada.Fill($ds, "Benutzer") | out-null

# Verbindung schließen
$conn.close

# Ausgabe
"Anzahl der Tabellen im Dataset: " + $ds.Tables.Count
"Anzahl der Datensätze in Tabelle 1: " + $ds.Tables[0].Rows.Count

"Ausgabe der Daten:"
$ds.Tables[0] | select-Object -first 15 | ft UserFirstName,
UserSurname, userid
```

Listing 30.7
Datenbankzugriff mit einem Dataset über einen providerspezifischen Datenadapter auf eine Access-Datenbank [DataSet Access.ps1]

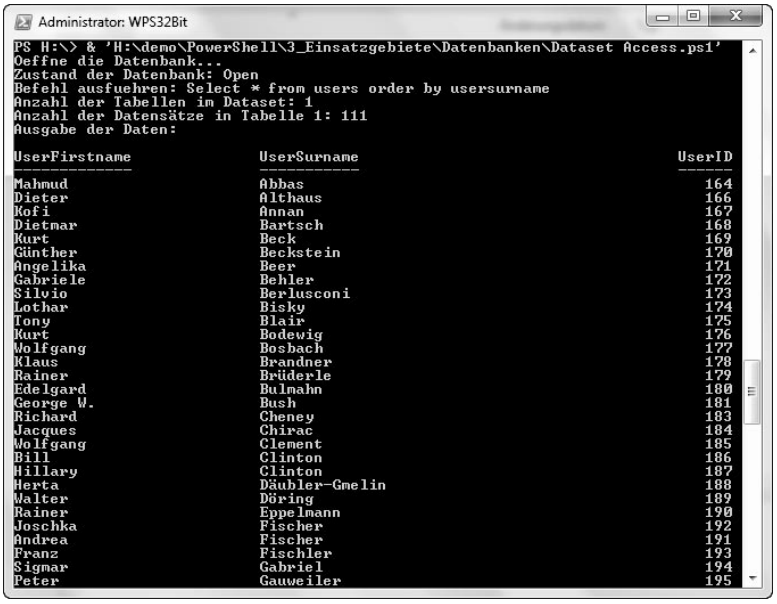


Abbildung 30.9
Ausgabe des Skripts

Providerneutrales Beispiel

Im zweiten Beispiel wird der Datenadapter von der Providerfabrik erzeugt.

Listing 30.8 Datenbankzugriff mit einem Dataset über einen providerneutralen Datenadapter auf eine Microsoft-SQL-Server-Datenbank [DataSet SQL Server1.ps1]

```
# Parameters
$PROVIDER = "System.Data.SqlClient"
$CONNSTRING = "Data Source=.\SQLEXPRESS;AttachDbFileName=H:\demo\
PowerShell\Datenbanken\users.mdf;Integrated Security=True;"
$SQL = "Select * from users"

# Create Factory
$provider =
[System.Data.Common.DbProviderFactories]::GetFactory($PROVIDER)

# Create Connection
$conn = $provider.CreateConnection()
$conn.ConnectionString = $CONNSTRING

# Open Connection
$conn.Open();
"Zustand der Datenbank: " + $conn.State

# Create Command
$cmd = $provider.CreateCommand()
$cmd.CommandText = $SQL
$cmd.Connection = $conn

# Create Adapter
[System.Data.Common.DbDataAdapter] $ada =
$provider.CreateDataAdapter()
$ada.SelectCommand = $cmd

# Create Dataset
$ds = New-Object System.Data.DataSet

# Retrieve data
$e = $ada.Fill($ds, "Benutzer")

# Datenbank schließen
$Conn.Close()
"Zustand der Datenbank: " + $conn.State
```

```
# Output
"Tabellenanzahl: " + $ds.Tables.Count
"Zeilenanzahl in Tabelle 1: " + $ds.Tables[0].Rows.Count

# Access table
$Tabelle = $ds.Tables[0]

# Print all rows
"Zeilen:"
$Tabelle | Select UserFirstName, UserSurname, userid
```

30.4 Datenzugriff mit den PowerShell-Erweiterungen

Commandlets

Die *www.IT-Visions.de-PowerShell-Erweiterungen* stellen folgende Commandlets zur Verfügung:

- `Test-DBConnection`: zeigt an (True/False), ob ein Verbindungsaufbau möglich ist.
- `Invoke-DBCommand`: führt eine SQL-Anweisung auf der Datenquelle aus. Rückgabewert ist eine Zahl, die angibt, wie viele Zeilen betroffen waren.
- `Get-DBTable`: liefert eine Datenmenge gemäß SQL-Anweisung aus einer Datenquelle in Form einer Menge von `DataRow`-Objekten.
- `Get-DBRow`: liefert eine Zeile aus einer Datenquelle in Form eines ADO.NET-`DataRow`-Objekts. Sofern die angegebene SQL-Anweisung mehr als eine Zeile zurückgibt, wird nur die erste Zeile geliefert.
- `Set-DBTable`: speichert Änderungen in einem `DataTable`-Objekt in der Datenquelle.
- `Set-DBRow`: speichert Änderungen in einem `DataRow`-Objekt in der Datenquelle.

Alle Commandlets basieren auf providerneutraler Programmierung. Sofern Commandlets eine Verbindungszeichenfolge erwarten, erlauben Sie auch die Angabe eines Providers (Parameter `-Provider`). Die Angabe des Providers ist optional, die Standardeinstellung ist „MSSQL“. Andere mögliche Werte sind „OLEDB“, „ODBC“, „ORACLE“ und „ACCESS“. Bitte beachten Sie, dass diese Kürzel erwartet werden, nicht der vollständige „Provider Invariant Name“.

Abbildung 30.10
Einsatz von Get-DataTable zum Zugriff auf eine Microsoft-SQL-Server-Tabelle, die Flugdaten enthält

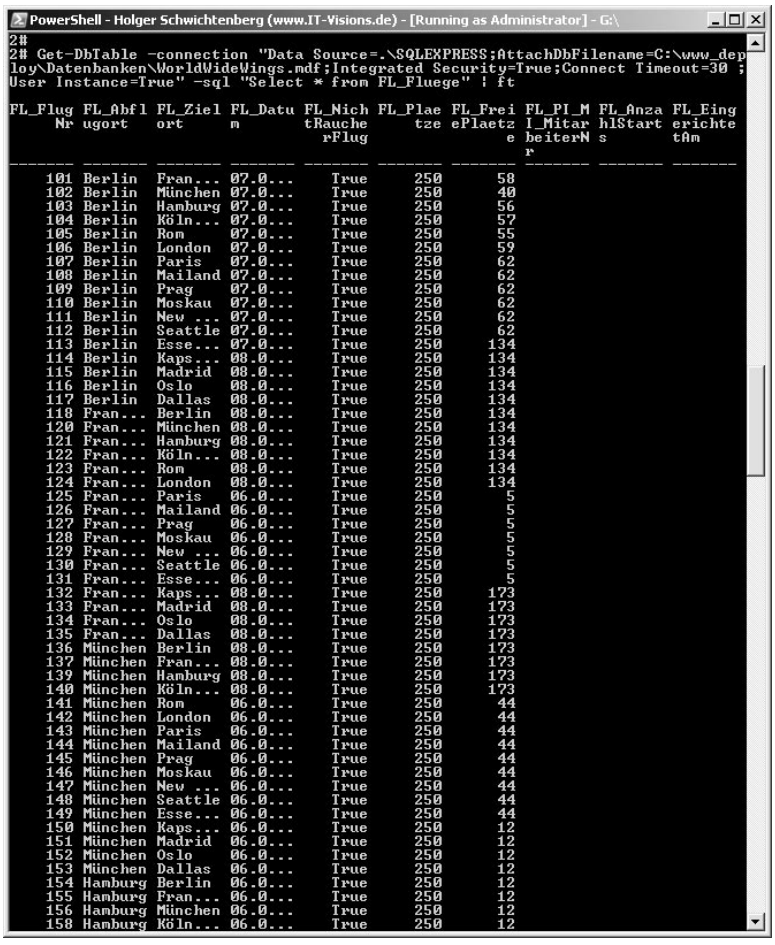
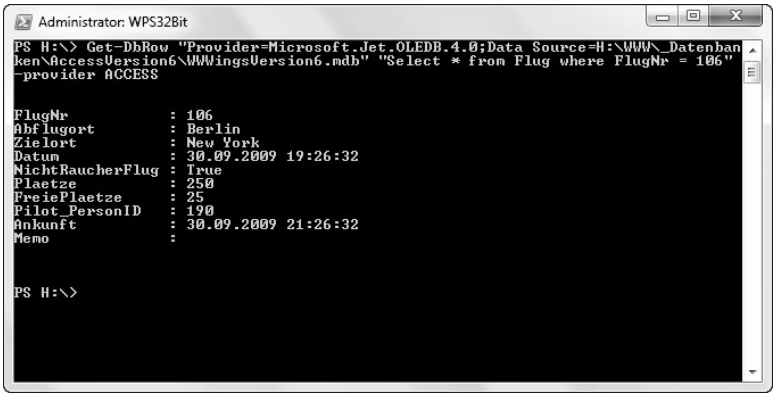


Abbildung 30.11
Einsatz von Get-DataRow zum Zugriff auf den ersten Datensatz in einer Access-Tabelle



Das folgende Skript zeigt die oben genannten Commandlets in Aktion. Das Skript erledigt alle Aufgaben der vorherigen Skripte – und das wesentlich prägnanter!

Beispiel

Listing 30.9 *Datenbankzugriff mit den PowerShell Extensions von www.IT-Visions.de [/Datenbank/ITV Database Commandlet Demo.ps1]*

```
# Requirements: www.IT-Visions.de Commandlet Extension Library
# http://www.powershell-doktor.de

# Parameters
$SQL = "Select * from users order by UserSurname"
$Conn = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=H:\demo\
PowerShell\Datenbanken\users.mdb;"
$Provider = "ACCESS"

"----------Test database connections:"
test-dbconnection -connection $Conn   -sql $SQL -provider $Provider

"---------- Execute Commands:"

$SQL1 = "Select count(*) from users"
$SQL2 = "insert into users ( UserFirstName, UserSurname) values
('Hans', 'Meier')"
$SQL3 = "delete from users where UserSurname='Meier'"

invoke-ScalarDbCommand -connection $Conn   -sql $SQL1 -provider
$Provider
invoke-DbCommand -connection $Conn   -sql $SQL2 -provider $Provider
invoke-ScalarDbCommand -connection $Conn   -sql $SQL1 -provider
$Provider
invoke-DbCommand -connection $Conn   -sql $SQL3 -provider $Provider
invoke-ScalarDbCommand -connection $Conn   -sql $SQL1 -provider
$Provider

"---------- Get Data "

$table = Get-DbTable -connection $Conn   -sql $SQL -provider
$Provider
$table | ft

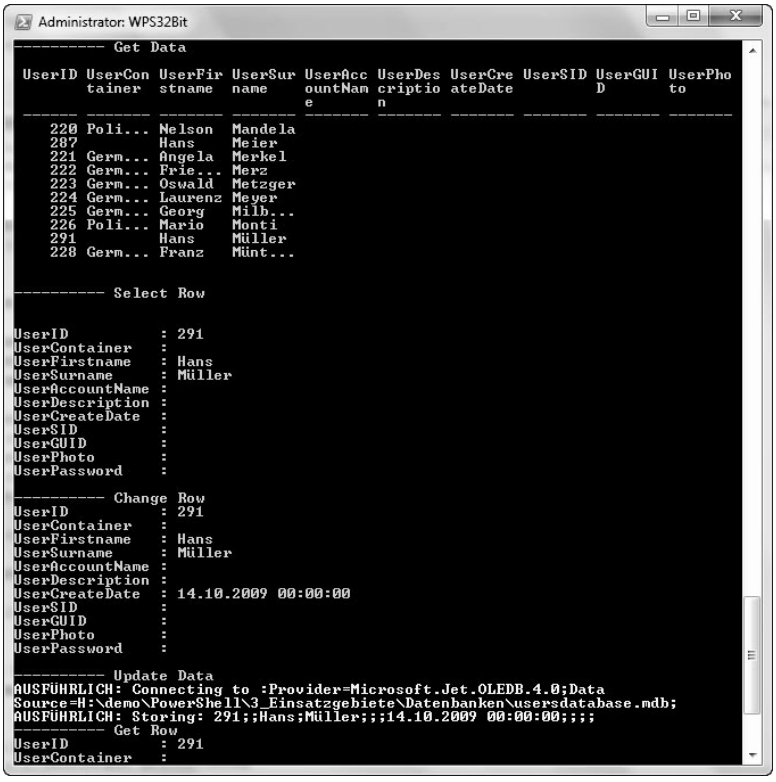
"---------- Select Row "
$row = $table | where  { $_.usersurname -eq "Müller" }
$Row

"---------- Change Row "
$row.UsercreateDate = [DateTime] "14/10/2009"
$Row

"---------- Update Data "
$table | Set-DbTable -connection $Conn -sql $sql -provider
$Provider  -verbose
```

```
"---------- Get Row"
$SQL = "Select * from users where usersurname = 'Müller'"
$row = Get-DbRow $Conn $SQL $Provider
$row
```

Abbildung 30.12
Ausgabe des obigen Skripts



31 Registrierungsdatenbank (Registry)

Für die Windows-Registrierungsdatenbank (Registry) steht in der PowerShell ein PowerShell Provider zur Verfügung. Dies bedeutet, dass die Navigations-Commandlets (`Set-Location`, `Get-ChildItem`, `New-Item`, `Get-ItemProperty` etc.) in der Registrierungsdatenbank zur Verfügung stehen.

Registrierungsdatenbank-Navigation

31.1 Schlüssel auslesen

Die Unterschlüssel eines Registrierungsdatenbankschlüssels listet man auf mit:

```
Get-ChildItem hklm:\software (Alias: dir hklm:\software)
```

Man kann auch mit CD den aktuellen Pfad in die Registrierungsdatenbank verlegen:

`Set-Location hklm:\software` (Alias: `cd hklm:\software`)

und dann einfach mit `Get-ChildItem` auflisten.

Zugriff auf einen einzelnen Schlüssel der Registrierungsdatenbank erhält man mit:

```
Get-Item www.it-visions.de
```

bzw. mit absolutem Pfad:

```
Get-Item hklm:\software\www.it-visions.de
```

Das Ergebnis sind .NET-Objekte des Typs `Microsoft.Win32.RegistryKey`. `Get-Item` liefert immer eine einzelne Instanz dieser Klasse. `Get-ChildItem` liefert keine, eine oder mehrere Instanzen.

In die Registrierungsdatenbank wechselt man mit dem Befehl:

```
Cd hklm:\software\www.it-visions.de
```

31.2 Schlüssel anlegen und löschen

Einen Schlüssel in der Registrierungsdatenbank erzeugt man mit:

```
New-Item -path hklm:\software -name "www.IT-visions.de"
```

oder

```
md -path hklm:\software\www.IT-visions.de
```



`New-Item` steht auch als `md` zur Verfügung. `md` ist jedoch kein Alias, sondern eine eingebaute Funktion.

Man kann ganze Schlüssel kopieren mit `Copy-Item`:

```
Copy-Item hklm:\software\www.it-visions.de hklm:\software\www.IT-
Visions.de_Backup
```

Einen Schlüssel aus der Registrierungsdatenbank zusammen mit allen enthaltenen Werten löscht man mit:

```
Remove-Item "hklm:\software\www.it-visions.de" -Recurse
```

31.3 Laufwerke definieren

New-PSDrive

Durch das Definieren eines neuen PowerShell-Laufwerks kann man eine Abkürzung zum schnellen Zugang zu Schlüsseln definieren:

```
New-PSDrive -Name ITV -PSProvider Registry -Root
 hklm:\software\www.it-visions.de
```

Danach kann man anstelle von

```
Get-Item hklm:\software\www.it-visions.de
```

auch schreiben:

```
Get-Item itv:
```

Zwei solcher Abkürzungen sind bereits vordefiniert (siehe Tabelle 31.1).

Tabelle 31.1 *Definierte Abkürzungen für Registrierungsdatenbank-Hauptschlüssel*

HKLM	HKEY_LOCAL_MACHINE
HKCU	HKEY_CURRENT_USER

31.4 Werte anlegen und löschen

New-Itemproperty

Einen Zeichenkettenwert zu einem Schlüssel legt man an mit:

```
New-Itemproperty -path "hklm:\software\www.it-visions.de" -name
"Inhaber" -value  "Dr. Holger Schwichtenberg" -type string
```

Einen Zahlenwert zu einem Schlüssel legt man an mit:

```
New-Itemproperty -path "hklm:\software\www.it-visions.de" -name
"Gruendungsjahr" -value  1996 -type DWord
```

Eine Mehrfachzeichenkette zu dem Schlüssel legt man an mit:

```
$Websites = "www.IT-Visions.de", "www.IT-Visionen.de", "hs.IT-
Visions.de"
New-Itemproperty -path "www.IT-visions.de" -name "Websites" -value
$Websites -type multistring
```

Einen Binärwert zu dem Schlüssel legt man an mit:

```
$Werte = Get-Content
  H:\demo\PowerShell\Registry\www.IT-Visions.de_Logo.jpg -encoding
byte
New-Itemproperty -path "www.IT-visions.de" -name "Logo" -value
$Werte
   -type binary
```

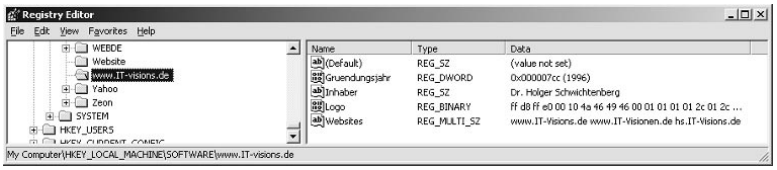


Abbildung 31.1 *Ergebnis der Registrierungsdatenbankoperationen*

Die folgende Tabelle zeigt alle möglichen Datentypen und deren Verwendung in der PowerShell.

Registry-Datentyp	Bedeutung	Typbezeichner	Verarbeitung in der PowerShell
REG_BINARY	Array von Byte	Binary	Byte[]
REG_DWORD	Zahl	DWord	Int
REG_EXPAND_SZ	Zeichenkette mit Platzhaltern	Multistring	String[]
REG_MULTI_SZ	Mehrere Zeichenketten	ExpandString	String
REG_SZ	Einfache Zeichenkette	String	String

Tabelle 31.2*: Datentypen in der Registrierungsdatenbank*

Einen bestehenden Wert ändert man mit `Set-ItemProperty`:

```
# Wert verändern
$Websites = "www.IT-Visions.de", "www.IT-Visionen.de", "hs.IT-
Visions.de",
   "IT-Visions.de"
Set-Itemproperty -path "www.IT-visions.de" -name "Websites" -value
$Websites
   -type multistring
```

Einen Wert aus einem Registrierungsdatenbankschlüssel löscht man mit `Remove-ItemProperty`:

```
Remove-ItemProperty -path "hklm:\software\www.it-visions.de" -name
"Inhaber"
```

31.5 Werte auslesen

Get-ItemProperty

Die vorhandenen Werte in einem Registrierungsdatenbankschlüssel listet man auf mit:

```
Get-ItemProperty -Path "hklm:\software\www.it-visions.de"
```

Den Inhalt eines einzelnen Eintrags bekommt man durch:

```
(Get-Item "hklm:\software\www.it-visions.de").GetValue("Inhaber")
```

oder

```
(Get-ItemProperty  "hklm:/software/firmenname").Inhaber
```

31.6 Praxisbeispiel

Das folgende Skript speichert Daten über Website-Konfiguration in der Registrierungsdatenbank.

Abbildung 31.2
Eingabedaten

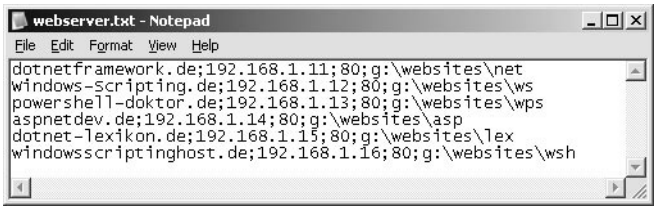
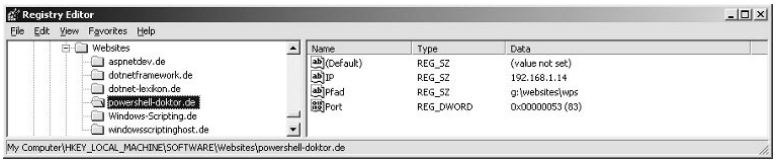


Abbildung 31.3
Ergebnis



Listing 31.1
Werte aus einer CSV-Datei in der Registrierungsdatenbank speichern [Einsatzgebiete/Registry/Registry_CreateWebsitesKeys.ps1]

```
# Registry-Schlüssel anlegen aus CSV-Daten

$Pfad = "hklm:/software/Websites"

if (Test-Path $Pfad) { del $Pfad -recurse -force }
if (!(Test-Path $Pfad )) { md $Pfad  }

$Websiteliste = Get-Content "H:\demo\PowerShell\Registry\
webserver.txt"

foreach($Website in $WebsiteListe)
{
$WebsiteDaten = $Website.Split(";")
md ($Pfad + "\" + $WebsiteDaten[0])
New-Itemproperty -path ($Pfad + "\" + $WebsiteDaten[0]) -name "IP"
   -value $WebsiteDaten[1] -type String
```

```
New-Itemproperty -path ($Pfad + "\" + $WebsiteDaten[0]) -name
"Port"
   -value $WebsiteDaten[2] -type dword
New-Itemproperty -path ($Pfad + "\" + $WebsiteDaten[0]) -name
"Pfad"
   -value $WebsiteDaten[3] -type String
$WebsiteDaten[0] + " angelegt!"
}
```

32 Computer-verwaltung

In PowerShell 1.0 wurde die Computerverwaltung nicht gut durch PowerShell Commandlets unterstützt. Neu ab PowerShell 2.0 ist die Unterstützung für Verwaltungsaufgaben rund um den Computer, die bisher nur aufwändiger über WMI möglich waren.

32.1 Computerinformationen

Win32_ Computer-system, Win32_ OperatingSystem

Für Informationen über den Computer gibt es kein eigenes Commandlet. Wichtige Informationen über den Computer und das installierte Betriebssystem erhält man mit den WMI-Klassen `Win32_Computersystem` und `Win32_OperatingSystem`.

```
Get-Wmiobject win32_computersystem
Get-Wmiobject win32_operatingSystem
```

Die Seriennummer des Betriebssystems erhält man mit:

```
Get-Wmiobject Win32_OperatingSystem | select serialnumber
```

Die Versionsnummer des Betriebssystems erhält man über das Attribut `Version` in der WMI-Klasse `Win32_OperatingSystem` oder über die .NET-Klasse `System.Environment`:

```
Get-Wmiobject Win32_OperatingSystem | select Version
[System.Environment]::OSVersion
```

Möchte man keine Versionsnummer, sondern den tatsächlichen Produktnamen, hilft das Commandlet `Get-OSVersion` aus dem Modul „PSSystemTools“ aus dem PowerShellPack. `Get-OSVersion` übersetzt wie nachfolgend dargestellt.

Listing 32.1
Ausschnitt aus der Implementierung von Get-OSVersion

```
Switch -regex ($os.Version)
    {
        "5.1.2600" { "Windows XP" }
        "5.1.3790" { "Windows Server 2003" }
        "6.0.6001" {
            if ($os.ProductType -eq 1) {
                "Windows Vista"
            } else {
                "Windows Server 2008"
            }
        }
        "6.1."    { "Windows 7" }
        DEFAULT { Throw "Version not listed" }
    }
```

Noch etwas genauer ist `Get-WindowsEdition` (ebenfalls aus „PSSystem-Tools“). Hier erhalten Sie auch eine Information über die eingesetzte Variante (Home, Professional, Enterprise, Ultimate etc.).

Informationen über das BIOS gewinnt man über die WMI-Klasse `Win32_Bios`:

```
Get-Wmiobject win32_bios
```

Die Startkonfiguration steht in `Win32_BootConfiguration`:

```
Get-Wmiobject Win32_BootConfiguration
```

Das Windows-Systemverzeichnis steht wieder in `System.Environment`:

```
"Systemverzeichnis: "+ [System.Environment]::SystemDirectory
```

Den Status der Windows-Produktaktivierung findet man hier:

```
Get-Wmiobject Win32_WindowsProductActivation
```

Es gibt auch Daten über die gewählten Wiederherstellungsoptionen des Windows-Betriebssystems:

```
Get-Wmiobject Win32_OSRecoveryConfiguration
```

Die Umgebungsvariablen erhält man über das PowerShell-Laufwerk „env“:

```
dir env:
```

Die Information über eine einzelne Umgebungsvariable erhält man, indem man den Namen der Umgebungsvariablen an das Laufwerk env: anfügt, z.B.:

```
dir env:/Path
```

Möchte man nur den Inhalt einer Umgebungsvariablen wissen, verwendet man am besten `Get-Content`:

```
Get-Content env:/Path
```

Den Wert, den `Get-Content` zurückliefert, kann man in eine Variable speichern, um diese weiterzuverwenden, z.B. zum Aufspalten der Path-Zeichenkette mit Hilfe der `Split()`-Methode aus der .NET-Klasse `System.String`:

```
$Pfade = Get-Content env:/Path

$Pfade.Split(";")
```

Wenn Sie wissen wollen, wie viele Dateien es in den Suchpfaden von Windows gibt, wäre folgender Befehl einzugeben:

```
(Get-Content env:/Path).Split(";")  | Get-ChildItem | measure-Object
```

```
PS C:\Skripte> Dir env:

Name                           Value
----                           -----
ALLUSERSPROFILE                C:\ProgramData
APPDATA                        C:\Users\hs.ITV\AppData\Roaming
CommonProgramFiles             C:\Program Files\Common Files
CommonProgramFiles(x86)        C:\Program Files (x86)\Common Files
CommonProgramW6432             C:\Program Files\Common Files
COMPUTERNAME                   ES0
ComSpec                        C:\Windows\system32\cmd.exe
FP_NO_HOST_CHECK               NO
HOMEDRIVE                      C:
HOMEPATH                       \Users\hs.ITV
LOCALAPPDATA                   C:\Users\hs.ITV\AppData\Local
LOGONSERVER                    \\E02
NUMBER_OF_PROCESSORS           8
OS                             Windows_NT
Path                           %SystemRoot%\system32\WindowsPowerShell\v1.0\;C:\Windows\system32;C:\Windows;C:\Windo...
PATHEXT                        .COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PROCESSOR_ARCHITECTURE         AMD64
PROCESSOR_IDENTIFIER           Intel64 Family 6 Model 26 Stepping 5, GenuineIntel
PROCESSOR_LEVEL                6
PROCESSOR_REVISION             1a05
ProgramData                    C:\ProgramData
ProgramFiles                   C:\Program Files
ProgramFiles(x86)              C:\Program Files (x86)
ProgramW6432                   C:\Program Files
PSModulePath                   C:\Users\hs.ITV\Documents\WindowsPowerShell\Modules;C:\Windows\system32\WindowsPowerS...
PUBLIC                         C:\Users\Public
SESSIONNAME                    Console
SystemDrive                    C:
SystemRoot                     C:\Windows
TEMP                           C:\Users\hs.ITV\AppData\Local\Temp
TMP                            C:\Users\hs.ITV\AppData\Local\Temp
USERDNSDOMAIN                  II-VISIONS.LOCAL
USERDOMAIN                     ITV
USERNAME                       HS
USERPROFILE                    C:\Users\hs.ITV
VS90COMNTOOLS                  c:\Program Files (x86)\VS9\Common7\Tools\
windir                         C:\Windows


PS C:\Skripte>
```

Abbildung 32.1 *Auflisten der Umgebungsvariablen*

Die installierten Schriftarten listet `Get-Font` aus dem Modul „PSSystemTools“ (PowerShellPack) auf.

32.2 Computername und Domäne

Umbenennen

Zum Umbenennen eines Computers bietet PowerShell ab 2.0 den Befehl:

```
Rename-Computer -NewComputerName NeuerName
```

Unter Angabe von `-ComputerName` kann man die Aktion auch für ein entferntes System ausführen:

```
Rename-Computer -ComputerName AlterName -NewComputerName NeuerName
```

Zur Domäne hinzufügen

Zum Hinzufügen des Computers zu einer Domäne oder Arbeitsgruppe steht zur Verfügung:

```
Add-Computer -DomainName fbi.org
```

Zusätzlich kann man mit `-Server` einen Domänencontroller und mit `-OU` einen Zielcontainer angeben. Alternativ kann man mit `-Workgroup` den Computer zu einer Arbeitsgruppe hinzufügen.

Auch das Entfernen aus einer Domäne ist möglich:

```
Remove-Computer
```

32.3 Herunterfahren und Neustarten

Das Herunterfahren und Neustarten eines Computers konnte in der PowerShell 1.0 nur über WMI erfolgen:

Restart-ComputerBei beiden Commandlets kann man mit `-ComputerName` einen entfernten Computer festlegen und mit `-Credentials` ein Rechte-Objekt für den Wechsel der Identität angeben. Mit `-force` erzwingt man die Aktion, selbst wenn Benutzer angemeldet sind (siehe Bildschirmabbildung).

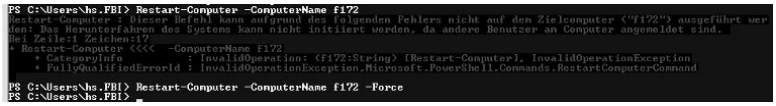


Abbildung 32.2 Herunterfahren eines entfernten Systems

32.4 Wiederherstellungspunkte verwalten

Systemwiederherstellung

Für den Umgang mit Wiederherstellungspunkten bietet PowerShell 2.0:

`Checkpoint-Computer`: Wiederherstellungspunkt erstellen

`Disable-ComputerRestore`: deaktiviert die Systemwiederherstellung auf einem Laufwerk (dieses Laufwerk wird bei einer Wiederherstellung nicht zurückgesetzt)

`Enable-ComputerRestore`: aktiviert die Systemwiederherstellung für ein Laufwerk

`Get-ComputerRestorePoint`: Liste der verfügbaren Wiederherstellungspunkte

`Restore-Computer`: Zurücksetzen des Computers auf einen der Wiederherstellungspunkte

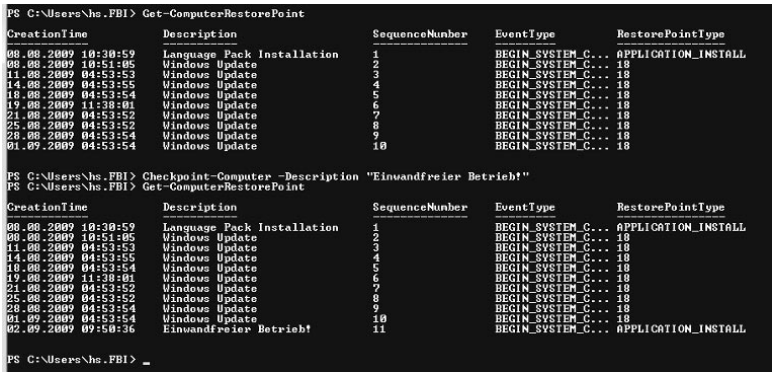


Abbildung 32.3 Erstellen eines Wiederherstellungspunkts und Auflisten der Wiederherstellungspunkte

33 Hardware-verwaltung

Für den Zugriff auf Hardwareinformationen bietet die PowerShell 1.0 keine Commandlets. Hier bleibt aber der Zugriff auf WMI. Alternativ kann man einige Funktionen über die PowerShell-Erweiterungen von *www.IT-Visions.de* erreichen.

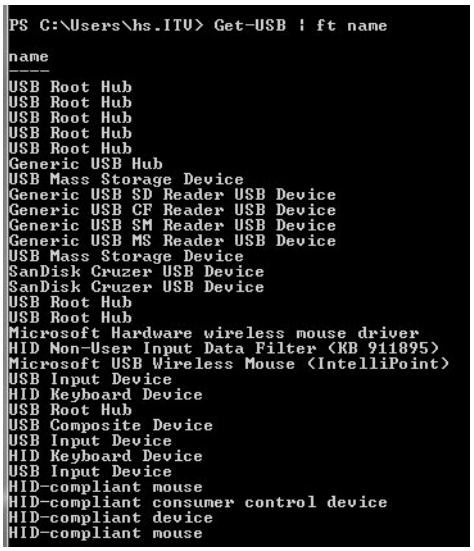
33.1 Hardwarebausteine

Informationen über die installierte Hardware erhält man innerhalb der Windows PowerShell über WMI, also über die Verwendung des Commandlets `Get-WmiObject` zusammen mit der entsprechenden WMI-Klasse (siehe Tabelle).

Hardware-baustein	PowerShell-Befehl (Standard)	www.IT-Visions.de PowerShell Extensions	PSSystemTools im Power-Shell-Pack
Prozessoren	`Get-Wmiobject Win32_Processor`	`Get-Processor`	`Get-Processor`
Haupt-speicher	`Get-Wmiobject Win32_MemoryDevice`	`Get-MemoryDevice`	
Grafikkarte	`Get-Wmiobject Win32_VideoController`	`Get-Videocontroller`	
Soundkarte	`Get-Wmiobject Win32_SoundDevice`	`Get-SoundDevice`	
Festplatten	`Get-Wmiobject Win32_Diskdrive`	`Get-Disk`	
Bandlauf-werke	`Get-Wmiobject Win32_Tapedrive`	`Get-Tapedrive`	
Optische Laufwerke (CD/DVD)	`Get-Wmiobject Win32_CDRomdrive`	`Get-CDRomdrive`	
Netzwerk-karten	`Get-Wmiobject Win32_NetworkAdapter`	`Get-Networkadapter`	
USB-Controller	`Get-Wmiobject Win32_USBController`	`Get-USBController`	`Get-USB`
Tastatur	`Get-Wmiobject Win32_Keyboard`	`Get-Keyboard`	
Maus	`Get-Wmiobject Win32_PointingDevice`	`Get-PointingDevice`	

Tabelle 33.1 *Abruf von Hardwareinformationen in der PowerShell*

Abbildung 33.1
Anwendung von Get-USB zur Anzeige der USB-Geräte



Prozessor

Die Anzahl der Prozessoren auf einem System erhält man auch über die .NET-Klasse `System.Environment`:

```
"Anzahl der Prozessoren: " + [System.Environment]::ProcessorCount
```

32- versus 64-Bit

Die Frage, ob man ein 32- oder ein 64-Bit-System vor sich hat, kann man einfach über die Umgebungsvariable `$env:PROCESSOR_ARCHITECTURE` oder die Commandlets `Test-32Bit` und `Test-64Bit` aus dem Modul „PSSystemTools“ aus dem PowerShellPack klären. Diese beiden Commandlets liefern `true` oder `false` als Ergebnis.

Details über die Prozessoren liefern `Get-Wmiobject Win32_Processor` bzw. `Get-Processor` (aus den *www.IT-Visions.de* PowerShell Extensions oder im Modul PSSystemTools im PowerShellPack).

Anzeige

Win32_VideoController

Informationen über die Grafikkarte liefert `Get-Wmiobject Win32_VideoController` bzw. `Get-Videocontroller` (aus den *www.IT-Visions.de* PowerShell Extensions).

Die aktuelle Bildschirmauflösung liefert `Get-DisplaySetting` (aus dem Modul „PSSystemTools“ im PowerShellPack. `Get-MultiTouchMaximum` (gleiches Modul) gibt die Anzahl der Finger wieder, die man zur Bedienung einsetzen kann. „0“ bedeutet, dass keine Multi-Touch-Funktion verfügbar ist.

Batterien und USV

Win32_Battery

Batterien in Notebooks und Unterbrechungsfreie Stromversorgungen (USVs) an Servern kann man mit der WMI-Klasse `Win32_Battery` abfragen.

```
Get-WmiObject Win32_Battery -ComputerName E10 | fl name,
EstimatedChargeRemaining, EstimatedRunTime
```

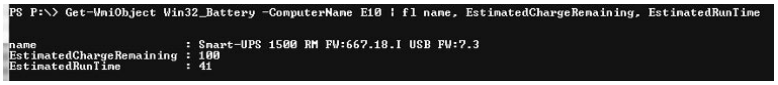


Abbildung 33.2 *Einsatz der WMI-Klasse Win32_Battery zur Abfrage der USV*

33.2 Druckerverwaltung

Der Befehl

Win32_Printer

```
Get-Wmiobject Win32_Printer
```

liefert eine Liste der verfügbaren Drucker.

Mit

Win32_Printjob

```
Get-Wmiobject Win32_Printjob
```

zeigt man alle aktuellen Druckaufträge.

Mit dem folgenden Befehl hält man alle Druckaufträge für einen bestimmten Drucker an:

```
Get-Wmiobject Win32_Printjob -Filter "Drivername='HP LaserJet 2100
PCL6'" | foreach-Object { $_.pause() }
```



Zum Ausgeben von Informationen zum Drucker verwendet man in der PowerShell `Out-Printer`. Dieses Commandlet wurde bereits vorher in diesem Buch besprochen (siehe *Kapitel 8.12, Weitere Ausgabefunktionen*).

Zum Löschen von Druckaufträgen ruft man die Delete()-Methode auf:

```
"Vorher:"
Get-Wmiobject Win32_Printjob -Filter "Drivername='Dell MFP Laser
3115cn PCL6'"
"Löschen aller Aufträge":
Get-Wmiobject Win32_Printjob -Filter "Drivername='Dell MFP Laser
3115cn PCL6'" | Foreach-Object { $_.Delete() }
"Nachher:"
Get-Wmiobject Win32_Printjob -Filter "Drivername='Dell MFP Laser
3115cn PCL6'"
```

Listing 33.1 *Löschen von Druckaufträgen [Drucker/CancelAll.ps1]*

Mit Hilfe von WMI kann man auch Druckerports verwalten:

Listing 33.2 *Anlegen eines TCP-Druckerports [Drucker/Create-Port.ps1]*

```
$p = [WmiClass] 'Win32_TCPIPPrinterPort'
#oder: $p = New-Object system.management.managementclass Win32_
TCPIPPrinterPort
$Port = $p.CreateInstance()
$Port.Name = "IP_192.168.1.224"
$Port.Protocol = 1
$Port.HostAddress = "192.168.1.224"
$Port.PortNumber = 1234
$Port.SNMPEnabled = $FALSE
$Port.Put()
```

34 Softwareverwaltung

Zum Bereich Softwareverwaltung gehören:

- Inventarisierung der installierten Anwendungen
- Installieren von Anwendungen
- Deinstallieren von Anwendungen

Für die Softwareverwaltung bietet die PowerShell keine eigenen Commandlets. Die WMI-Klasse `Win32_Product` mit Informationen über die installierten MSI-Pakete steht zur Verfügung, sofern der „WMI-Provider für Windows Installer" installiert ist. Unter Windows Server 2003 ist dieser Provider eine Installationsoption von Windows und nicht mehr Standardinstallationsumfang.

`Win32_Product` gilt aber nur für Anwendungen, die mit Windows Installer installiert wurden. Alle Anwendungen, die man in der Systemsteuerung sieht, bekommt man nur über den Schlüssel *HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall* der Registrierungsdatenbank.

34.1 Softwareinventarisierung

Win32_ Products

Die Klasse `Win32_Product` liefert die installierten MSI-Pakete:

```
Get-Wmiobject Win32_Product
```

Natürlich kann man filtern. Der folgende Befehl listet nur die MSI-Pakete, deren Namen mit a beginnen:

```
Get-Wmiobject Win32_Product | Where-Object { $_.name -like "a*" }
```

Der zweite Filter sortiert alle MSI-Pakete heraus, deren Hersteller Microsoft ist:

```
Get-Wmiobject Win32_Product | Where-Object { $_.vendor -like
"microsoft*" }
```

Sie können auch gezielt feststellen, ob eine bestimmte Anwendung installiert ist:

Listing 34.1 *Prüfen, ob eine Software installiert ist [Einsatzgebiete/Software/InstallCheck.ps1]*

```
function Get-IsInstall($Application, $Computer, $Version)
{
$a = (Get-WmiObject -Class Win32_Product -Filter
"Name='$Application' and Version='$Version'" -computername
$Computer)
return ($a -ne $null)
}
$e = Get-IsInstall "QuickTime" "E01" "7.2.0.240"
if ($e) { "Software is installed!" }
else { "Software is not installed!" }
```

In einem Pipeline-Befehl kann man auch eine komplette Inventarisierungslösung schreiben, die nacheinander mehrere Computer gemäß einer Liste in einer Textdatei abgefragt hat und die gefundenen Anwendungen in eine CSV-Datei exportiert.

```
Get-Content "computernamen.txt" |
foreach { Get-wmiobject win32_product -computername $_ } |
where { $_.vendor -like "*Microsoft*" } |
export-csv "Softwareinventar.csv" -notypeinformation
```

Noch etwas verfeinern kann man die Inventarisierungslösung, indem man vor dem Zugriff auf den Computer mit einem Ping prüft, ob der Computer überhaupt erreichbar ist, um die lange Timeout-Zeit von WMI zu vermeiden. Da hierzu dann ein Pipelining-Befehl nicht mehr ausreicht und man ein Skript braucht, kann man auch direkt die Lösung besser parametrisieren.

Listing 34.2 *Softwareinventarisierung per PowerShell-Skript [SoftwareInventory_WMI_WithPing.ps1]*

```
$Hersteller = "*Microsoft*"
$Eingabedateiname = "computernamen.txt"
$Ausgabedateiname = "Softwareinventar.csv"

# Import der Computernamen
$Computernamen = Get-Content "computernamen.txt"
$Computernamen | foreach {

if (Ping($_))
{
Write-Host "Inventarisiere Software für Computer $_ ..."
# Auslesen der installierten MSI-Pakete auf allen Computern
$Software = foreach { Get-wmiobject win32_product -computername $_
} |
  where { $_.vendor -like $Hersteller }

# Export in CSV
$Software | export-csv "Softwareinventar.csv" -notypeinformation
}
else
```

```
{
Write-Error "Computer nicht erreichbar!"
}
}
# Ping ausführen
function Ping
{
$status = Get-WmiObject Win32_PingStatus -filter
"Address='$args[0]'" | select StatusCode
return $status.Statuscode -eq 0
}
```

Die Liste der installierten Softwareaktualisierungen (Patches, Hotfixes) erhält man mit:

```
Get-Wmiobject Win32_Quickfixengineering
```

Die installierten Audio-/Video-Codecs kann man sich so anzeigen lassen:

```
Get-Wmiobject Win32_CodecFile | select group,name
```

`Win32_Product` gilt aber nur für Anwendungen, die mit Windows Installer installiert wurden. Alle Anwendungen, die man in der Systemsteuerung sieht, bekommt man nur über den Schlüssel *HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall* der Registrierungsdatenbank heraus.

```
Get-ChildItem HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\
Uninstall
```

Vereinfachen kann man den Zugang dorthin, indem man ein neues PowerShell-Laufwerk definiert:

```
New-PSDrive -Name Software -PSProvider Registrierungsdatenbank -
Root HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall
```

Danach kann man einfach schreiben:

```
Get-ChildItem Software:
```

Beim Filtern muss man in jedem Fall beachten, dass die Eigenschaften (z.B. `DisplayName`, `Comments` und `UninstallString`) keine Attribute des Objekts vom Typ `Microsoft.Win32.RegistryKey` sind, sondern Unterelemente dieses Objekts. Daher muss `GetValue()` verwendet werden für den Zugriff auf die Daten.

```
Get-ChildItem Software: | Where-Object -FilterScript { $_
.GetValue("DisplayName") -like "a*"} | ForEach-Object -Process {$_
.GetValue("DisplayName") , $_.GetValue("Comments"), $_
.GetValue("UninstallString") }
```

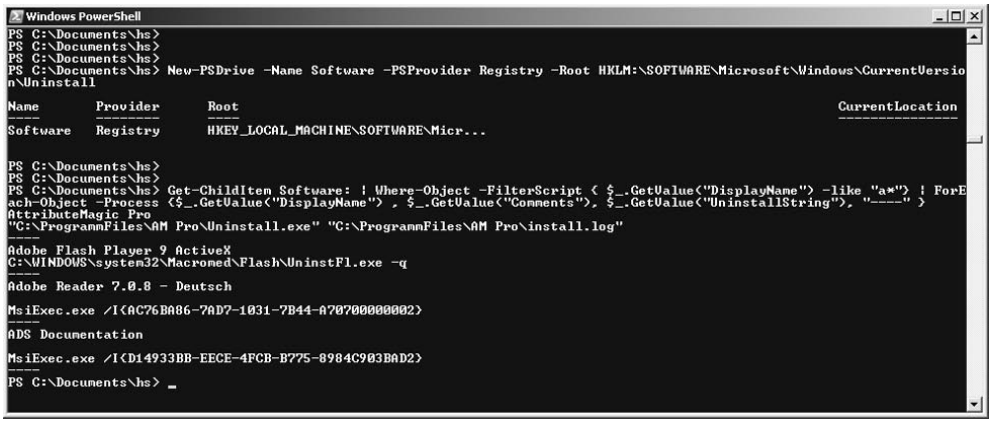


***Abbildung 34.1**: Auflisten der installierten Software, die mit „a" beginnt*

Programme, die beim Systemstart automatisch gestartet werden, findet man in `Win32_StartupCommand`:

```
Get-Wmiobject Win32_StartupCommand
```

34.2 Installation von Anwendungen

Install()

Eine skriptbasierte Installation ist bei vielen Anwendungen möglich, allerdings ist die Vorgehensweise abhängig von der verwendeten Installationstechnologie. Microsoft liefert in WMI eine Installationsunterstützung für Installationspakete, die auf Windows Installer (alias Microsoft Installer), abgekürzt MSI, basieren.

WMI erlaubt den Aufruf des Microsoft Installers, um ein beliebiges MSI-Paket zu installieren. Die Klasse `Win32_Product` bietet dazu die Methode `Install()` an. Die Methode erwartet einen oder drei Parameter:

- den Pfad zu dem MSI-Paket,
- an das Paket zu übergebende Kommandozeilenparameter,
- die Entscheidung, ob die Anwendung für alle Benutzer (`True`) oder nur den angemeldeten Benutzer (`False`) installiert werden soll.

Zu beachten ist, dass die `Install()`-Methode eine statische Methode der WMI-Klasse `Win32_Product` ist. Eine Ferninstallation ist möglich unter Bezugnahme auf diese Klasse auf einem entfernten System.

***Listing 34.3** Installation eines MSI-Pakets [Software_Installation.ps1]*

```
$Anwendung = "H:\demo\PS\Setup_for_HelloWorld_VBNET.msi"
"Installiere Anwendung..." + $Anwendung
(Get-WmiObject -ComputerName E01 -List | Where-Object -FilterScript
{$_.Name -eq "Win32_Product"}).Install($Anwendung)
"Fertig!"
```

34.3 Deinstallation von Anwendungen

Uninstall()

Die WMI-Klasse `Win32_Product` bietet auch eine `Uninstall()`-Methode ohne Parameter zur Deinstallation von MSI-Paketen. Zu beachten ist, dass zur Identifizierung der zu deinstallierenden Anwendung nicht der Name des Installationspakets, sondern der Anwendungsname (`Name` oder `Caption`) oder der GUID (`IdentifyingNumber`) anzugeben ist. Im Fall von *Setup_for_HelloWorld_VBNET.msi* ist der Name „Hello World VB.NET“.

```
$Name = "Hello World VB.NET"
"Starte Deinstallation..."
$Ergebnis = (Get-WmiObject -Class Win32_Product -Filter
"Name='$Name'" -ComputerName E01).Uninstall().Returnvalue
if ($Ergebnis -ne 0) { Write-Error "Deinstallationsfehler:
$Ergebnis"; Exit }
"Deinstallation beendet!"
```

Listing 34.4 *Deinstallation eines MSI-Pakets [Software_Installation.ps1]*

Zu jeder Anwendung ist in der Registrierungsdatenbank ein sogenannter Uninstall-String angegeben, der sagt, was man ausführen muss, um die Anwendung zu deinstallieren. Dies funktioniert auch für nicht MSI-basierte Anwendungen.

Der folgende Befehl listet die Deinstallationsanweisungen für alle Anwendungen auf, deren Name mit „a“ beginnt.

```
Get-ChildItem -Path HKLM:\SOFTWARE\Microsoft\Windows\
CurrentVersion\Uninstall
| Where-Object -FilterScript { $_.GetValue("DisplayName") -like
"a*"} | ForEach-Object -Process {$_.GetValue("DisplayName"), $_
.GetValue("UninstallString") }
```

Listing 34.5 *Installationsdaten aus der Registrierungsdatenbank [SoftwareInventory_Registry.ps1]*

34.4 Praxisbeispiel: Installationstest

Das folgende Skript installiert zum Test eine Anwendung und deinstalliert sie dann direkt wieder. Zu Beginn, nach der Installation und am Ende wird jeweils geprüft, ob die Anwendung installiert ist.

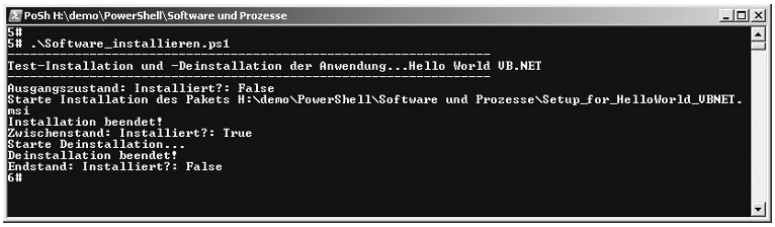


Abbildung 34.2 *Ausgabe des Skripts*

Listing 34.6 *Testen einer Softwareinstallation [Software_Testinstallation.ps1]*

```
function Get-IsInstall($Application, $Computer)
{
$a = (Get-WmiObject -Class Win32_Product -Filter
"Name='$Application'" -Computer $Computer)
return ($a -ne $null)
}

$Name = "Hello World VB.NET"
$Computer = "E01"
$Paket = "H:\demo\PowerShell\Software und Prozesse\Setup_for_
HelloWorld_VBNET.msi"

"-----------------------------------------------------------------"
"Testinstallation und -deinstallation der Anwendung..." + $Name
"-----------------------------------------------------------------"

"Ausgangszustand: Installiert?: " + (Get-IsInstall $Name $Computer)

"Starte Installation des Pakets " + $Paket
$Ergebnis = ([WMIClass] "Win32_
Product").Install($Paket).Returnvalue
if ($Ergebnis -ne 0) { Write-Error "Installationsfehler:
$Ergebnis"; Exit }
"Installation beendet!"

"Zwischenstand: Installiert?: " + (Get-IsInstall $Name $Computer)

"Starte Deinstallation..."
$Ergebnis = (Get-WmiObject -Class Win32_Product -Filter
"Name='$Name'" -ComputerName E01).Uninstall().Returnvalue
if ($Ergebnis -ne 0) { Write-Error "Deinstallationsfehler:
$Ergebnis"; Exit }
"Deinstallation beendet!"

"Endstand: Installiert?: " + (Get-IsInstall $Name $Computer)
```

34.5 Versionsnummer ermitteln

Get-FileVersionInfo

Die Versionsnummer einer ausführbaren Datei ermittelt man über Get-FileVersionInfo aus den PSCX (siehe *Abbildung 34.3*). Das Commandlet liefert eine Instanz der .NET-Klasse `System.Diagnostics.FileVersionInfo`.

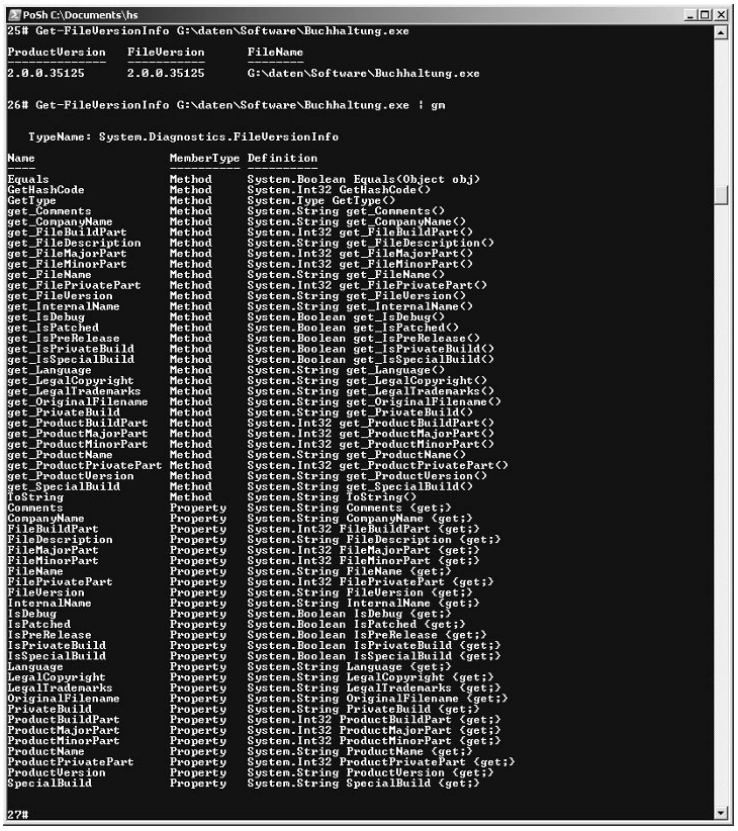


Abbildung 34.3
Get-FileVersionInfo

34.6 Servermanager

Das PowerShell-Modul „Servermanager“, das in Windows Server 2008 R2 mitgeliefert wird, ergänzt die PowerShell nur um bescheidende drei Commandlets, die aber nützlich zur Einrichtung des Betriebssystems sind:

- `Get-WindowsFeature`
- `Add-WindowsFeature`
- `Remove-WindowsFeature`

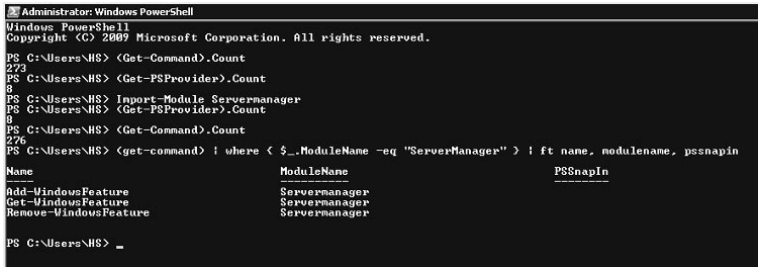


Abbildung 34.4
Importieren des Moduls „Server-Manager“

Get-WindowsFeature

Features auflisten

Get-WindowsFeature liefert eine sehr ausführliche Liste aller installierbaren Betriebssystemoptionen in Windows Server 2008 R2. Hierbei ist der Begriff „WindowsFeature“ in der PowerShell leider nicht gleichzusetzen mit „Feature“ aus der Sicht der Server-Manager-Benutzerschnittstelle. „WindowsFeature“ umfasst drei Konzepte der Benutzerschnittstelle:

- Rollen (Roles) sind zentrale Bereiche, z.B. Active Directory Rights Management Services.
- Rollendienste (Role Services) sind Teilbereiche von Rollen, z.B. „Identity Federation Support“, innerhalb der Active Directory Rights Management Services.
- Features sind kleinere Funktionen bzw. Hilfsbausteine wie z.B. .NET Framework 3.5.1. Features sind zum Teil hierarchisch aufgebaut. Features können außerdem bei der Installation von Rollen erforderlich sein, z.B. .NET Framework 3.5.1 für die Rolle „Application Server“.

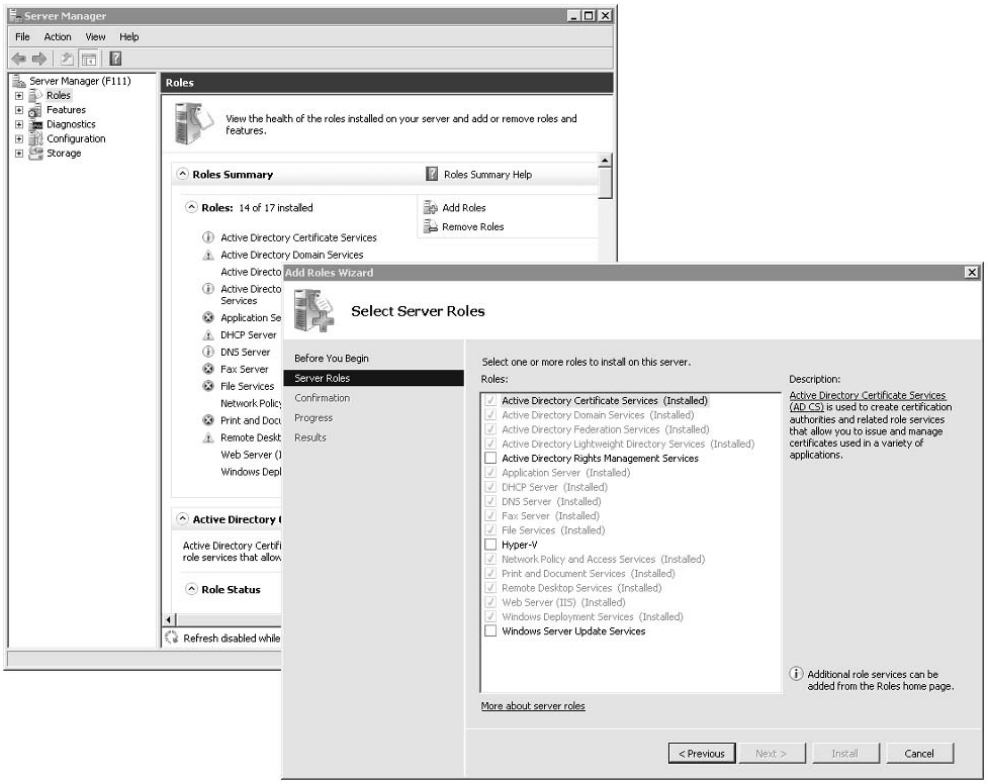


***Abbildung 34.5**: Rollen in Windows Server 2008 R2*

`Get-WindowsFeature` zeigt alle installierbaren Rollen/Rollendienste/Features an – in einer hierarchischen Darstellung mit Markierung der installierten Teile.

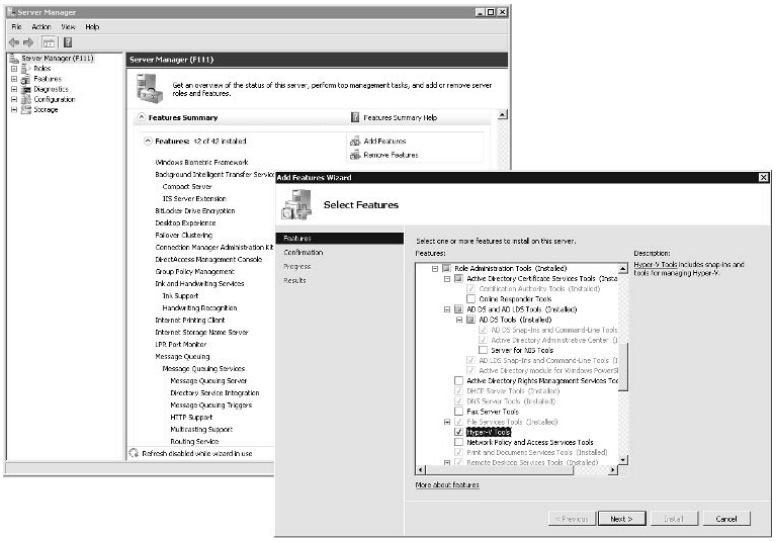


Abbildung 34.6 *Features in Windows Server 2008 R2*

```
Administrator: Windows PowerShell
PS C:\Users\NS> Get-WindowsFeature

Display Name                                            Name
------------                                            ----
[X] Active Directory Certificate Services               AD-Certificate
    [X] Certification Authority                         ADCS-Cert-Authority
    [ ] Certification Authority Web Enrollment          ADCS-Web-Enrollment
    [ ] Online Responder                                ADCS-Online-Cert
    [ ] Network Device Enrollment Service               ADCS-Device-Enrollment
    [ ] Certificate Enrollment Web Service              ADCS-Enroll-Web-Svc
    [ ] Certificate Enrollment Policy Web Service       ADCS-Enroll-Web-Pol
[X] Active Directory Domain Services                    AD-Domain-Services
    [X] Active Directory Domain Controller              ADDS-Domain-Controller
    [ ] Identity Management for UNIX                    ADDS-Identity-Mgmt
        [ ] Server for Network Information Services     ADDS-NIS
        [ ] Password Synchronization                    ADDS-Password-Sync
        [ ] Administration Tools                        ADDS-IDMU-Tools
[X] Active Directory Federation Services                AD-Federation-Services
    [X] Federation Service                              ADFS-Federation
    [ ] Federation Service Proxy                        ADFS-Proxy
    [ ] AD FS Web Agents                                ADFS-Web-Agents
        [ ] Claims-aware Agent                          ADFS-Claims
        [ ] Windows Token-based Agent                   ADFS-Windows-Token
[X] Active Directory Lightweight Directory Services     ADLDS
[ ] Active Directory Rights Management Services         ADRMS
    [ ] Active Directory Rights Management Server       ADRMS-Server
    [ ] Identity Federation Support                     ADRMS-Identity
[X] Application Server                                  Application-Server
    [X] .NET Framework 3.5.1                            AS-NET-Framework
    [X] Web Server (IIS) Support                        AS-Web-Support
    [X] COM+ Network Access                             AS-Ent-Services
    [ ] TCP Port Sharing                                AS-TCP-Port-Sharing
    [X] Windows Process Activation Service Support      AS-WAS-Support
        [X] HTTP Activation                             AS-HTTP-Activation
        [X] Message Queuing Activation                  AS-MSMQ-Activation
        [X] TCP Activation                              AS-TCP-Activation
        [X] Named Pipes Activation                      AS-Named-Pipes
    [X] Distributed Transactions                        AS-Dist-Transaction
        [X] Incoming Remote Transactions                AS-Incoming-Trans
        [X] Outgoing Remote Transactions                AS-Outgoing-Trans
        [X] WS-Atomic Transactions                      AS-WS-Atomic
[X] DHCP Server                                         DHCP
[X] DNS Server                                          DNS
[X] Fax Server                                          Fax
[X] File Services                                       File-Services
    [X] File Server                                     FS-FileServer
    [X] Distributed File System                         FS-DFS
        [X] DFS Namespaces                              FS-DFS-Namespace
        [X] DFS Replication                             FS-DFS-Replication
    [X] File Server Resource Manager                    FS-Resource-Manager
    [X] Services for Network File System                FS-NFS-Services
    [X] Windows Search Service                          FS-Search-Service
    [ ] Windows Server 2003 File Services               FS-Win2003-Services
        [ ] Indexing Service                            FS-Indexing-Service
    [X] BranchCache for network files                   FS-BranchCache
[ ] Hyper-V                                             Hyper-V
[X] Network Policy and Access Services                  NPAS
    [X] Network Policy Server                           NPAS-Policy-Server
    [ ] Routing and Remote Access Services              NPAS-RRAS-Services
        [ ] Remote Access Service                       NPAS-RRAS
        [ ] Routing                                     NPAS-Routing
    [ ] Health Registration Authority                   NPAS-Health
    [ ] Host Credential Authorization Protocol          NPAS-Host-Cred
[X] Print and Document Services                         Print-Services
    [X] Print Server                                    Print-Server
    [ ] LPD Service                                     Print-LPD-Service
    [ ] Internet Printing                               Print-Internet
    [ ] Distributed Scan Server                         Print-Scan-Server
[X] Remote Desktop Services                             Remote-Desktop-Services
    [X] Remote Desktop Session Host                     RDS-RD-Server
    [ ] Remote Desktop Virtualization Host              RDS-Virtualization
    [X] Remote Desktop Licensing                        RDS-Licensing
    [X] Remote Desktop Connection Broker                RDS-Connection-Broker
    [X] Remote Desktop Gateway                          RDS-Gateway
```

Abbildung 34.7*: Darstellung von Get-WindowsFeature*

Man könnte glauben, `Get-WindowsFeature` würde diese formatierte Darstellung direkt erzeugen (was den PowerShell-Prinzipien widersprechen würde). Dem ist aber nicht so: `Get-WindowsFeature` liefert (wie es sich für ein „ordentliches" Commandlet gehört) nur Objekte des Typs `Microsoft.Windows.ServerManager.Commands.Feature`.

Abbildung 34.8
Inhalt der Klasse Microsoft.Windows.ServerManager.Commands.Feature

```
PS C:\Users\HS> Get-WindowsFeature | gm


   TypeName: Microsoft.Windows.ServerManager.Commands.Feature

Name                MemberType Definition
----                ---------- ----------
Equals              Method     bool Equals(System.Object obj)
GetHashCode         Method     int GetHashCode()
GetType             Method     type GetType()
ToString            Method     string ToString()
AdditionalInfo      Property   System.Collections.Hashtable AdditionalInfo {get;}
BestPracticesModelId Property   System.String BestPracticesModelId {get;}
DependsOn           Property   System.String[] DependsOn {get;}
Depth               Property   System.Int32 Depth {get;}
DisplayName         Property   System.String DisplayName {get;}
FeatureType         Property   System.String FeatureType {get;}
Installed           Property   System.Boolean Installed {get;}
Name                Property   System.String Name {get;}
Notification        Property   Microsoft.Windows.ServerManager.Commands.Notification[] Notification {get;}
Parent              Property   System.String Parent {get;}
Path                Property   System.String Path {get;}
SubFeatures         Property   System.String[] SubFeatures {get;}
SystemService       Property   System.String[] SystemService {get;}
```

Die Darstellung wird durch die Standardvorlage `Feature.format.ps1xml` erzeugt, die man unter `\Windows\System32\WindowsPowerShell\v1.0\Modules\ServerManager` findet. Weil das Einrücken und die „Häkchen" für andere Fälle interessant sein könnten, sei diese Vorlage hier abgedruckt.

```
<View>
    <Name>Feature</Name>
    <ViewSelectedBy>
<TypeName>Microsoft.Windows.ServerManager.Commands.Feature</
TypeName>
    </ViewSelectedBy>
    <TableControl>
<TableHeaders>
    <TableColumnHeader>
<Label>Display Name</Label>
<Width>55</Width>
<Alignment>left</Alignment>
    </TableColumnHeader>
    <TableColumnHeader>
<Label>Name</Label>
<Width>23</Width>
<Alignment>left</Alignment>
    </TableColumnHeader>
</TableHeaders>
<TableRowEntries>
    <TableRowEntry>
<TableColumnItems>
    <TableColumnItem>
                <ScriptBlock>
                $indent=""
for ($i=$_.Depth; $i -gt 1; $i--)
{
    $indent += "    "
}
```

```
                if ($_.Installed -eq $TRUE)
                {
                    $indent += "[X] "
                }
                else
                {
                    $indent += "[ ] "
                }

                $indent + $_.DisplayName
            </ScriptBlock>
        </TableColumnItem>
        <TableColumnItem>
        <PropertyName>Name</PropertyName>
        </TableColumnItem>
</TableColumnItems>
        </TableRowEntry>
</TableRowEntries>
        </TableControl>
</View>
```

Ausschnitt aus Feature.format.ps1xml

Wie immer steht es dem PowerShell-Nutzer frei, eine von der Standarddarstellung abweichende Darstellung zu erzwingen, z.B. eine einfache Tabelle.

```
Get-WindowsFeature | ft name, installed
```

Wenn durch eine Konfiguration ein Neustart des Systems aussteht, können Sie Get-WindowsFeature nicht aufrufen.

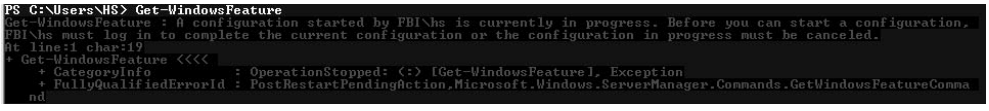


***Abbildung 34.9**: Get-WindowsFeature ist wegen eines ausstehenden Neustarts blockiert.*

Add-WindowsFeature

Features hinzufügen

Mit `Add-WindowsFeature` kann man Rollen, Rollendienste und Features hinzufügen. Abhängigkeiten werden dabei automatisch berücksichtigt. Anzugeben ist der „Name“ (z.B. AS-NET-Framework), nicht der „DisplayName“ (z.B. „.NET Framework 3.5.1“). Es können mehrere Features auf einmal installiert werden.

Der folgende Befehl installiert für die Internet Information Services (IIS) die Features Server Side Includes („Web-Includes“) und Web-DAV-Publishing („Web-DAV-Publishing“).

```
Add-WindowsFeature Web-Includes, Web-DAV-Publishing
```

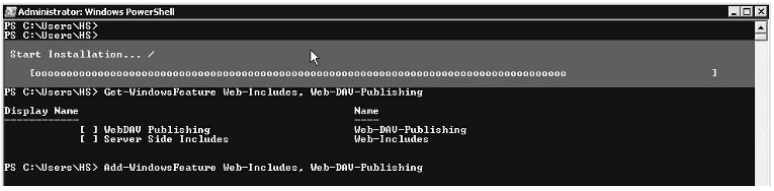


Abbildung 34.10 *Während der Ausführung zeigt die PowerShell-Konsole eine Fortschrittsanzeige.*

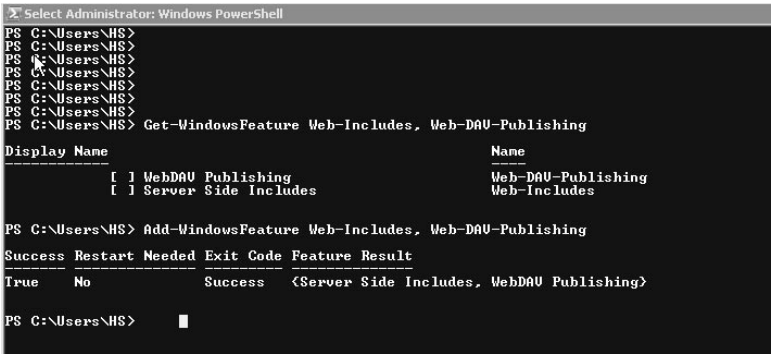


Abbildung 34.11 *Ergebnis nach Abschluss von Add-WindowsFeature*



Mit dem Parameter `–IncludeAllSubFeature` kann man alle Unterfeatures zu einem Feature auch installieren lassen.



Man kann `Add-WindowsFeature` nicht aufrufen, wenn währenddessen im Server Manager die Assistenten „Add/Remove Role“ oder „Add/Remove Feature“ gestartet sind. Gleiches gilt, wenn Add-WindowsFeature oder Remove-WindowsFeature schon in einer anderen Instanz der PowerShell laufen.

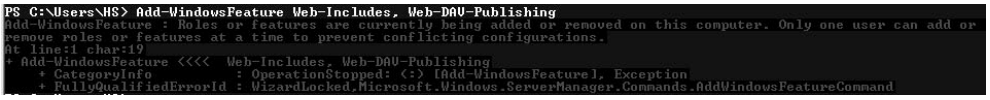


Abbildung 34.12*: Add-WindowsFeature kann nicht arbeiten, weil parallel bereits einer der Server Manager-Assistenten gestartet ist.*



Man kann durch den Zusatzparameter `-concurrent` erzwingen, dass das Commandlet auch arbeitet, wenn einer der Assistenten gestartet ist. Dies sollte man aber vermeiden, um Inkonsistenzen zu verhindern!

Remove-WindowsFeature

Features entfernen

Mit `Remove-WindowsFeature` kann man Features deinstallieren, z.B.:

```
Remove-WindowsFeature Web-Includes, Web-DAV-Publishing
```

Auch dabei sieht der PowerShell-Nutzer eine Fortschrittsanzeige.

Umgang mit Neustarts

`Add-WindowsFeature` und `Remove-WindowsFeature` können nach Abschluss der Arbeiten einen Neustart erfordern. Das Beispiel zeigt, dass „Add-WindowsFeature Fax" keinen Neustart erfordert, aber „Remove-WindowsFeature Fax"! Dies wird durch eine Warnmeldung sowie die zurückgegebene Instanz der Klasse `Microsoft.Windows.ServerManager.Commands.FeatureOperationResult` angezeigt.



Abbildung 34.13 *Das Entfernen eines Features erfordert einen Neustart.*

Man kann einen Computer mit `Restart-Computer` dann neu starten. Man kann aber auch bei `Add-WindowsFeature` und `Remove-WindowsFeature` den Parameter `-Restart` angeben. Dann findet ein Neustart automatisch statt, sofern er erforderlich ist.

Ausgabe des Befehls Get-WindowsFeature auf einem Beispielsystem

```
Display Name                                            Name
------------                                            ----
[ ] Active Directory Certificate Services               AD-Certificate
    [ ] Certification Authority                         ADCS-Cert-Authority
    [ ] Certification Authority Web Enrollment          ADCS-Web-Enrollment
    [ ] Online Responder                                ADCS-Online-Cert
    [ ] Network Device Enrollment Service               ADCS-Device-Enrollment
    [ ] Certificate Enrollment Web Service              ADCS-Enroll-Web-Svc
    [ ] Certificate Enrollment Policy Web Service       ADCS-Enroll-Web-Pol
[ ] Active Directory Domain Services                    AD-Domain-Services
    [ ] Active Directory Domain Controller              ADDS-Domain-Controller
    [ ] Identity Management for UNIX                    ADDS-Identity-Mgmt
        [ ] Server for Network Information Services     ADDS-NIS
        [ ] Password Synchronization                    ADDS-Password-Sync
        [ ] Administration Tools                        ADDS-IDMU-Tools
[ ] Active Directory Federation Services                AD-Federation-Services
    [ ] Federation Service                              ADFS-Federation
    [ ] Federation Service Proxy                        ADFS-Proxy
    [ ] AD FS Web Agents                                ADFS-Web-Agents
        [ ] Claims-aware Agent                          ADFS-Claims
        [ ] Windows Token-based Agent                   ADFS-Windows-Token
```

```
[ ] Active Directory Lightweight Directory Services     ADLDS
[ ] Active Directory Rights Management Services         ADRMS
    [ ] Active Directory Rights Management Server       ADRMS-Server
    [ ] Identity Federation Support                     ADRMS-Identity
[X] Application Server                                  Application-Server
    [X] .NET Framework 3.5.1                            AS-NET-Framework
    [X] Web Server (IIS) Support                        AS-Web-Support
    [ ] COM+ Network Access                             AS-Ent-Services
    [X] TCP Port Sharing                                AS-TCP-Port-Sharing
    [X] Windows Process Activation Service Support      AS-WAS-Support
        [X] HTTP Activation                             AS-HTTP-Activation
        [X] Message Queuing Activation                  AS-MSMQ-Activation
        [X] TCP Activation                              AS-TCP-Activation
        [X] Named Pipes Activation                      AS-Named-Pipes
    [ ] Distributed Transactions                        AS-Dist-Transaction
        [ ] Incoming Remote Transactions                AS-Incoming-Trans
        [ ] Outgoing Remote Transactions                AS-Outgoing-Trans
        [ ] WS-Atomic Transactions                      AS-WS-Atomic
[ ] DHCP Server                                         DHCP
[ ] DNS Server                                          DNS
[ ] Fax Server                                          Fax
[X] File Services                                       File-Services
    [X] File Server                                     FS-FileServer
    [X] Distributed File System                         FS-DFS
        [X] DFS Namespaces                              FS-DFS-Namespace
        [X] DFS Replication                             FS-DFS-Replication
    [X] File Server Resource Manager                    FS-Resource-Manager
    [ ] Services for Network File System                FS-NFS-Services
    [ ] Windows Search Service                          FS-Search-Service
    [ ] Windows Server 2003 File Services               FS-Win2003-Services
        [ ] Indexing Service                            FS-Indexing-Service
    [ ] BranchCache for network files                   FS-BranchCache
[ ] Hyper-V                                             Hyper-V
[ ] Network Policy and Access Services                  NPAS
    [ ] Network Policy Server                           NPAS-Policy-Server
    [ ] Routing and Remote Access Services              NPAS-RRAS-Services
        [ ] Remote Access Service                       NPAS-RRAS
        [ ] Routing                                     NPAS-Routing
    [ ] Health Registration Authority                   NPAS-Health
    [ ] Host Credential Authorization Protocol          NPAS-Host-Cred
[ ] Print and Document Services                         Print-Services
    [ ] Print Server                                    Print-Server
    [ ] LPD Service                                     Print-LPD-Service
    [ ] Internet Printing                               Print-Internet
    [ ] Distributed Scan Server                         Print-Scan-Server
[ ] Remote Desktop Services                             Remote-Desktop-Services
    [ ] Remote Desktop Session Host                     RDS-RD-Server
    [ ] Remote Desktop Virtualization Host              RDS-Virtualization
    [ ] Remote Desktop Licensing                        RDS-Licensing
    [ ] Remote Desktop Connection Broker                RDS-Connection-Broker
    [ ] Remote Desktop Gateway                          RDS-Gateway
    [ ] Remote Desktop Web Access                       RDS-Web-Access
```

```
[X] Web Server (IIS)                                    Web-Server
    [X] Web Server                                      Web-WebServer
        [X] Common HTTP Features                        Web-Common-Http
            [X] Static Content                          Web-Static-Content
            [X] Default Document                        Web-Default-Doc
            [X] Directory Browsing                      Web-Dir-Browsing
            [X] HTTP Errors                             Web-Http-Errors
            [X] HTTP Redirection                        Web-Http-Redirect
            [ ] WebDAV Publishing                       Web-DAV-Publishing
        [X] Application Development                     Web-App-Dev
            [X] ASP.NET                                 Web-Asp-Net
            [X] .NET Extensibility                      Web-Net-Ext
            [X] ASP                                     Web-ASP
            [X] CGI                                     Web-CGI
            [X] ISAPI Extensions                        Web-ISAPI-Ext
            [X] ISAPI Filters                           Web-ISAPI-Filter
            [X] Server Side Includes                    Web-Includes
        [X] Health and Diagnostics                      Web-Health
            [X] HTTP Logging                            Web-Http-Logging
            [X] Logging Tools                           Web-Log-Libraries
            [X] Request Monitor                         Web-Request-Monitor
            [X] Tracing                                 Web-Http-Tracing
            [ ] Custom Logging                          Web-Custom-Logging
            [ ] ODBC Logging                            Web-ODBC-Logging
        [X] Security                                    Web-Security
            [X] Basic Authentication                    Web-Basic-Auth
            [X] Windows Authentication                  Web-Windows-Auth
            [X] Digest Authentication                   Web-Digest-Auth
            [X] Client Certificate Mapping Authentic... Web-Client-Auth
            [X] IIS Client Certificate Mapping Authe... Web-Cert-Auth
            [X] URL Authorization                       Web-Url-Auth
            [X] Request Filtering                       Web-Filtering
            [X] IP and Domain Restrictions              Web-IP-Security
        [X] Performance                                 Web-Performance
            [X] Static Content Compression              Web-Stat-Compression
            [X] Dynamic Content Compression             Web-Dyn-Compression
    [X] Management Tools                                Web-Mgmt-Tools
        [X] IIS Management Console                      Web-Mgmt-Console
        [X] IIS Management Scripts and Tools            Web-Scripting-Tools
        [X] Management Service                          Web-Mgmt-Service
        [X] IIS 6 Management Compatibility              Web-Mgmt-Compat
            [X] IIS 6 Metabase Compatibility            Web-Metabase
            [X] IIS 6 WMI Compatibility                 Web-WMI
            [X] IIS 6 Scripting Tools                   Web-Lgcy-Scripting
            [X] IIS 6 Management Console                Web-Lgcy-Mgmt-Console
    [ ] FTP Server                                      Web-Ftp-Server
        [ ] FTP Service                                 Web-Ftp-Service
        [ ] FTP Extensibility                           Web-Ftp-Ext
    [ ] IIS Hostable Web Core                           Web-WHC
[ ] Windows Deployment Services                         WDS
    [ ] Deployment Server                               WDS-Deployment
    [ ] Transport Server                                WDS-Transport
```

```
[ ] Windows Server Update Services                      OOB-WSUS
[X] .NET Framework 3.5.1 Features                       NET-Framework
    [X] .NET Framework 3.5.1                            NET-Framework-Core
    [X] WCF Activation                                  NET-Win-CFAC
        [X] HTTP Activation                             NET-HTTP-Activation
        [X] Non-HTTP Activation                         NET-Non-HTTP-Activ
[ ] Background Intelligent Transfer Service (BITS)      BITS
    [ ] Compact Server                                  BITS-Compact-Server
    [ ] IIS Server Extension                            BITS-IIS-Ext
[ ] BitLocker Drive Encryption                          BitLocker
[ ] BranchCache                                         BranchCache
[ ] Connection Manager Administration Kit               CMAK
[ ] Desktop Experience                                  Desktop-Experience
[ ] DirectAccess Management Console                     DAMC
[X] Failover Clustering                                 Failover-Clustering
[ ] Group Policy Management                             GPMC
[ ] Ink and Handwriting Services                        Ink-Handwriting
    [ ] Ink Support                                     IH-Ink-Support
    [ ] Handwriting Recognition                         IH-Handwriting
[ ] Internet Printing Client                            Internet-Print-Client
[ ] Internet Storage Name Server                        ISNS
[ ] LPR Port Monitor                                    LPR-Port-Monitor
[X] Message Queuing                                     MSMQ
    [X] Message Queuing Services                        MSMQ-Services
        [X] Message Queuing Server                      MSMQ-Server
        [ ] Directory Service Integration               MSMQ-Directory
        [ ] Message Queuing Triggers                    MSMQ-Triggers
        [ ] HTTP Support                                MSMQ-HTTP-Support
        [ ] Multicasting Support                        MSMQ-Multicasting
        [ ] Routing Service                             MSMQ-Routing
    [ ] Message Queuing DCOM Proxy                      MSMQ-DCOM
[X] Multipath I/O                                       Multipath-IO
[ ] Network Load Balancing                              NLB
[ ] Peer Name Resolution Protocol                       PNRP
[ ] Quality Windows Audio Video Experience              qWave
[ ] Remote Assistance                                   Remote-Assistance
[ ] Remote Differential Compression                     RDC
[X] Remote Server Administration Tools                  RSAT
    [X] Role Administration Tools                       RSAT-Role-Tools
        [ ] Active Directory Certificate Services Tools RSAT-ADCS
            [ ] Certification Authority Tools           RSAT-ADCS-Mgmt
            [ ] Online Responder Tools                  RSAT-Online-Responder
        [ ] AD DS and AD LDS Tools                      RSAT-AD-Tools
            [ ] AD DS Tools                             RSAT-ADDS
                [ ] AD DS Snap-Ins and Command-Line ... RSAT-ADDS-Tools
                [ ] Active Directory Administrative ... RSAT-AD-AdminCenter
                [ ] Server for NIS Tools                RSAT-SNIS
            [ ] AD LDS Snap-Ins and Command-Line Tools  RSAT-ADLDS
            [ ] Active Directory module for Windows ... RSAT-AD-PowerShell
        [ ] Active Directory Rights Management Servi... RSAT-RMS
        [ ] DHCP Server Tools                           RSAT-DHCP
        [ ] DNS Server Tools                            RSAT-DNS-Server
```

```
        [ ] Fax Server Tools                              RSAT-Fax
        [X] File Services Tools                           RSAT-File-Services
            [X] Distributed File System Tools             RSAT-DFS-Mgmt-Con
            [X] File Server Resource Manager Tools        RSAT-FSRM-Mgmt
            [ ] Services for Network File System Tools    RSAT-NFS-Admin
        [ ] Hyper-V Tools                                 RSAT-Hyper-V
        [ ] Network Policy and Access Services Tools      RSAT-NPAS
        [ ] Print and Document Services Tools             RSAT-Print-Services
        [ ] Remote Desktop Services Tools                 RSAT-RDS
            [ ] Remote Desktop Session Host Tools         RSAT-RDS-RemoteApp
            [ ] Remote Desktop Gateway Tools              RSAT-RDS-Gateway
            [ ] Remote Desktop Licensing Tools            RSAT-RDS-Licensing
            [ ] Remote Desktop Connection Broker Tools    RSAT-RDS-Conn-Broker
        [X] Web Server (IIS) Tools                        RSAT-Web-Server
        [ ] Windows Deployment Services Tools             RSAT-WDS
    [X] Feature Administration Tools                      RSAT-Feature-Tools
        [ ] BitLocker Drive Encryption Administratio...   RSAT-BitLocker
            [ ] BitLocker Drive Encryption Tools          RSAT-Bitlocker-DriveEnc
            [ ] BitLocker Recovery Password Viewer        RSAT-Bitlocker-RecPwd
        [ ] BITS Server Extensions Tools                  RSAT-Bits-Server
        [X] Failover Clustering Tools                     RSAT-Clustering
        [ ] Network Load Balancing Tools                  RSAT-NLB
        [ ] SMTP Server Tools                             RSAT-SMTP
        [ ] WINS Server Tools                             RSAT-WINS
[ ] RPC over HTTP Proxy                                   RPC-over-HTTP-Proxy
[ ] Simple TCP/IP Services                                Simple-TCPIP
[ ] SMTP Server                                           SMTP-Server
[ ] SNMP Services                                         SNMP-Services
    [ ] SNMP Service                                      SNMP-Service
    [ ] SNMP WMI Provider                                 SNMP-WMI-Provider
[ ] Storage Manager for SANs                              Storage-Mgr-SANS
[ ] Subsystem for UNIX-based Applications                 Subsystem-UNIX-Apps
[ ] Telnet Client                                         Telnet-Client
[ ] Telnet Server                                         Telnet-Server
[ ] TFTP Client                                           TFTP-Client
[ ] Windows Biometric Framework                           Biometric-Framework
[ ] Windows Internal Database                             Windows-Internal-DB
[X] Windows PowerShell Integrated Scripting Environm...   PowerShell-ISE
[X] Windows Process Activation Service                    WAS
    [X] Process Model                                     WAS-Process-Model
    [X] .NET Environment                                  WAS-NET-Environment
    [X] Configuration APIs                                WAS-Config-APIs
[ ] Windows Server Backup Features                        Backup-Features
    [ ] Windows Server Backup                             Backup
    [ ] Command-line Tools                                Backup-Tools
[ ] Windows Server Migration Tools                        Migration
[ ] Windows System Resource Manager                       WSRM
[ ] Windows TIFF IFilter                                  TIFF-IFilter
[ ] WinRM IIS Extension                                   WinRM-IIS-Ext
[ ] WINS Server                                           WINS-Server
[ ] Wireless LAN Service                                  Wireless-Networking
[ ] XPS Viewer                                            XPS-Viewer
```

34.7 Softwareeinschränkungen mit dem PowerShell-Modul „AppLocker“

Application Control Policies

Das AppLocker-Modul dient der Steuerung der Funktion Application Locker (AppLocker) (alias: „Application Control Policies“) in Windows 7 (wirksam nur in Ultimate und Enterprise) und Windows Server 2008 R2, mit der man den Start von Anwendungen einschränken kann. AppLocker ist der Nachfolger der Software Restriction Policies (SRP).

AppLocker basiert wie SRP auf Regeln, die man in Gruppenrichtlinien hinterlegt (siehe Bildschirmabbildung). Es gibt dort Regeln für ausführbare Dateien (EXE), Bibliotheken (DLL), Windows Installer (MSI) und Skripte (Script). Regeln basieren wie bei SRP auf Kriterien (z.B. Standort, Datei-Hashwert, Hersteller). Anders als bei SRP kann es aber Ausnahmen von Regeln geben. Im Gruppenrichtlinieneditor kann man Standardregeln aktivieren, die verhindern, dass man sich selbst aussperrt. Außerdem kann man automatisch Regeln für Inhalte von Ordnern erstellen lassen („Automatically Generated Rules“). Regeln kann man als XML-Dateien importieren oder exportieren.

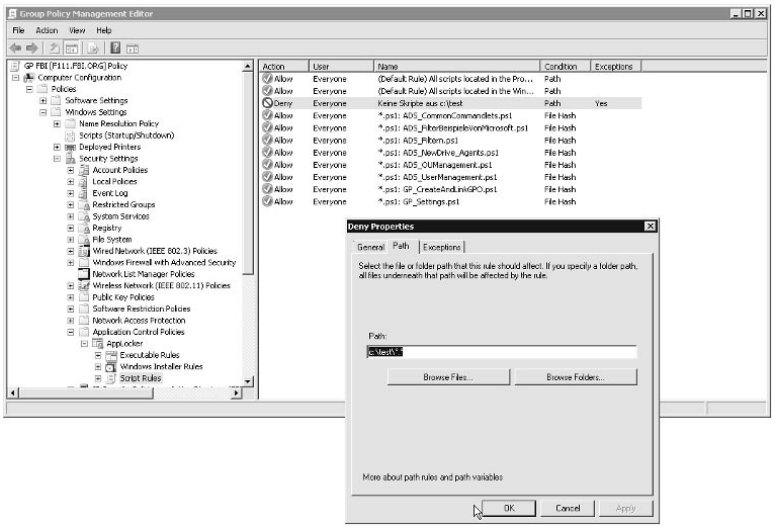


Abbildung 34.14 *Definition einer Einschränkungsregel für Skripte*



Regeln gelten erst, wenn die Regelart unter „Configure Rule Enforcement“ auch aktiviert wurde.

DLL-Regeln sind im Standard nicht sichtbar. Diese muss man erst unter „Configure Rule Enforcement/Advanced“ aktivieren.

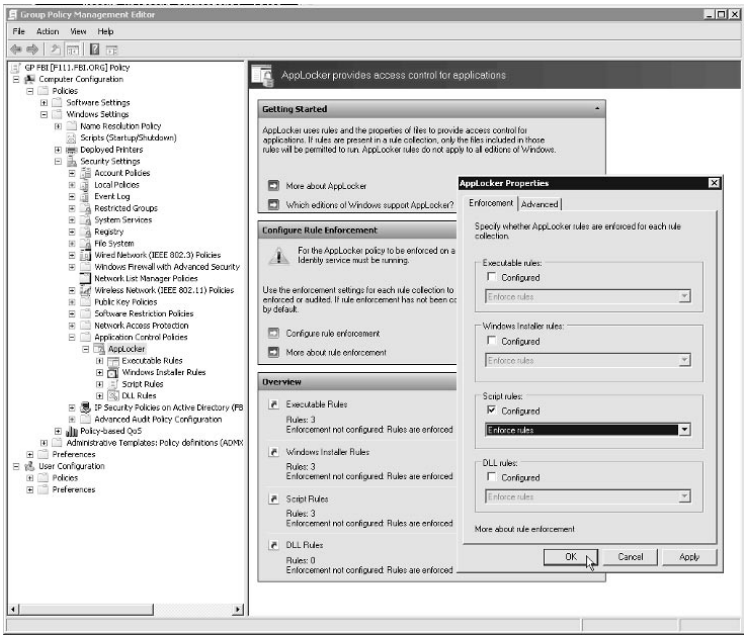


Abbildung 34.15
Aktivieren der AppLocker-Regeln

Das AppLocker-Modul enthält folgende Commandlets:

Commmandlets

- `Get-AppLockerPolicy`: gibt eine Liste der AppLocker-Regeln aus, wahlweise den lokalen PC (-local) oder eine Gruppenrichtlinie (-domain zusammen mit -ldap). Man kann auch die aktuell wirkenden Regeln ausgeben (-effective).
- `New-AppLockerPolicy`: erzeugt eine neue AppLocker-Regel.
- `Get-AppLockerFileInformation`: liefert zu einer Datei ein Objekt des Typs `Microsoft. Security.ApplicationId.PolicyManagement.PolicyModel.FileInformation`, das dazu dienen kann, mit `New-AppLockerPolicy` eine neue Regel zu erstellen.
- `Set-AppLockerPolicy`: legt eine AppLocker-Regel in einer Gruppenrichtlinie fest.
- `Test-AppLockerPolicy`: prüft, wie sich Regeln auf bestimmte Dateien auswirken

```
PS IIS:\Sites> import-module applocker -verbose
VERBOSE: Loading module from path 'C:\Windows\system32\WindowsPowerShell\v1.0\Modules\applocker\applocker.psd1'.
VERBOSE: Importing cmdlet 'Set-AppLockerPolicy'.
VERBOSE: Importing cmdlet 'Get-AppLockerPolicy'.
VERBOSE: Importing cmdlet 'Test-AppLockerPolicy'.
VERBOSE: Importing cmdlet 'Get-AppLockerFileInformation'.
VERBOSE: Importing cmdlet 'New-AppLockerPolicy'.
VERBOSE: Exporting cmdlet 'Set-AppLockerPolicy'.
VERBOSE: Exporting cmdlet 'Get-AppLockerPolicy'.
VERBOSE: Exporting cmdlet 'Test-AppLockerPolicy'.
VERBOSE: Exporting cmdlet 'Get-AppLockerFileInformation'.
VERBOSE: Exporting cmdlet 'New-AppLockerPolicy'.
VERBOSE: Importing cmdlet 'Get-AppLockerFileInformation'.
VERBOSE: Importing cmdlet 'Get-AppLockerPolicy'.
VERBOSE: Importing cmdlet 'New-AppLockerPolicy'.
VERBOSE: Importing cmdlet 'Set-AppLockerPolicy'.
VERBOSE: Importing cmdlet 'Test-AppLockerPolicy'.
PS IIS:\Sites> Get-Command -module applocker | ft name, modulename, pssnapin

Name                                ModuleName                          PSSnapIn
----                                ----------                          --------
Get-AppLockerFileInformation        applocker
Get-AppLockerPolicy                 applocker
New-AppLockerPolicy                 applocker
Set-AppLockerPolicy                 applocker
Test-AppLockerPolicy                applocker


PS IIS:\Sites> _
```

Abbildung 34.16
Inhalt des AppLocker-Moduls

Auflisten von Regeln

Get-AppLocker-Policy

`Get-AppLockerPolicy` liefert ein einzelnes Objekt vom Typ `AppLocker-Policy` aus dem Namensraum `Microsoft.Security.ApplicationId.Policy-Management.PolicyModel`. Hier ist das Attribut `RuleCollections` vom Typ `RuleCollection` interessant. Es gibt jeweils ein `RuleCollection`-Objekt für jede der vier Regelarten (EXE, DLL, MSI, Script). In dieser RuleCollection sind dann die Regeln enthalten (`FilePathRule`, `FilePublisherRule`, `FileHashRule`).

Mit `Get-AppLockerPolicy -local` fragt man die lokalen AppLocker-Einstellungen ab.

Mit dem Schalter `-Domain` kann man die AppLocker-Einstellungen aus einem einzelnen Gruppenrichtlinienobjekt abfragen, z.B.:

```
Get-AppLockerPolicy -Domain -LDAP "LDAP:// CN={685A9EAA-2CA5-4552-
B553-92A027F9E2B6},CN=Policies,CN=System,DC=FBI,DC=ORG"
```

Die wirkenden AppLocker-Regeln lassen sich wie folgt ermitteln:

```
Get-AppLockerPolicy -Effective
```

Um die einzelnen Regeln auszugeben, muss man zumindest die RuleCollections-Menge mit `Foreach-Object` aufspalten:

```
foreach { $_.RuleCollections }
```

Oder man schreibt noch ausführlicher:

```
foreach { $_.RuleCollections } | foreach { $_ }
```

Dann hat man Zugriff auf die einzelnen Regel-Objekte.

Beispiel

Das folgende Beispiel listet die einzelnen Regeln in Form einer Tabelle auf, in der man die Regelart und die Regelbedingung (nicht aber die Ausnahmen) sehen kann.

```
Get-AppLockerPolicy -Effective | foreach { $_
.RuleCollections.GetEnumerator() } | foreach { $_ } | sort name |
ft Name, { $_.GetType().Name }, PathConditions, HashConditions,
PublisherConditions
```

```
PS C:\Users\FoxMulder> Get-AppLockerPolicy -Effective | foreach { $_.RuleCollections.GetEnumerator() } | foreach { $_ }
| sort name | ft Name, { $_.GetType().Name }, PathConditions, HashConditions, PublisherConditions

Name                    $_.GetType().Name      PathConditions          HashConditions          PublisherConditions
----                    -----------------      --------------          --------------          -------------------
(Default Rule) All d... FilePublisherRule                                                      {*\*\*,0.0.0.0-*}
(Default Rule) All f... FilePathRule           {*}
(Default Rule) All f... FilePathRule           {%PROGRAMFILES%\*}
(Default Rule) All f... FilePathRule           {%WINDIR%\*}
(Default Rule) All s... FilePathRule           {*}
(Default Rule) All s... FilePathRule           {%PROGRAMFILES%\*}
(Default Rule) All s... FilePathRule           {%WINDIR%\*}
(Default Rule) All W... FilePathRule           {*.*}
(Default Rule) All W... FilePathRule           {%WINDIR%\Installer\*}
Keine Skripte aus c:... FilePathRule           {c:\test}
wps: ADS_CommonComma... FileHashRule                                   {SHA256 0xD159E38939...
wps: ADS_FilterBeisp... FileHashRule                                   {SHA256 0xF76CD9E2B9...
wps: ADS_Filtern.ps1    FileHashRule                                   {SHA256 0x89E63C6A8A...
wps: ADS_NewDrive_Ag... FileHashRule                                   {SHA256 0x6320783AC7...
wps: ADS_OUManagemen... FileHashRule                                   {SHA256 0x9784D35999...
wps: ADS_UserManagen... FileHashRule                                   {SHA256 0x6362571020...
wps: GP_CreateAndLin... FileHashRule                                   {SHA256 0xBB4A1D3B14...
wps: GP_Settings.ps1    FileHashRule                                   {SHA256 0x76682B9FA0...


PS C:\Users\FoxMulder> _
```

***Abbildung 34.17**: Ausgabe der wirksamen Regeln*

Die folgende Bildschirmabbildung zeigt, wie sich durch Aktualisierung der Gruppenrichtlinien mit `gpupdate /force` die Anzahl der wirkenden Regeln verändern kann.

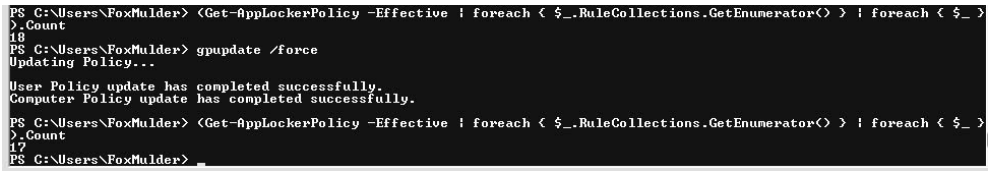


***Abbildung 34.18**: Nach dem Löschen einer Regel in einer der wirkenden Gruppenrichtlinien reduziert sich die Anzahl der Regeln von 18 auf 17.*

Prüfen der Regelwirkung

Test-AppLocker-Policy

Mit `Test-AppLockerPolicy` kann man prüfen, wie sich Regeln auf bestimmte Dateien auswirken.

Beispiel

Das Beispiel prüft, wie sich die wirksamen Regeln auf einen bestimmten Benutzer für die Skripte in c:\test auswirken.

```
Get-AppLockerPolicy -Effective | Test-AppLockerPolicy -path C:\
test\*.ps1 -user FBI\DanaScully
```

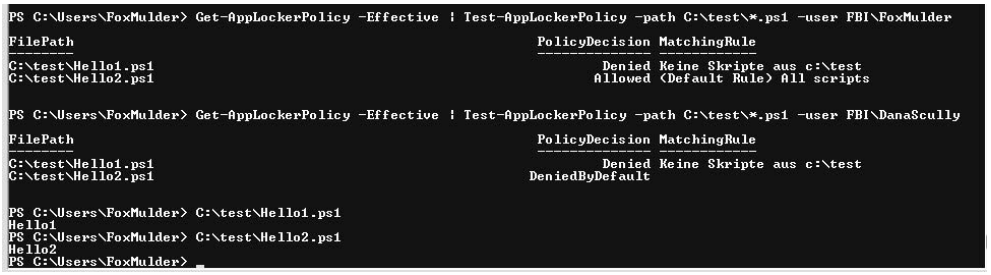


***Abbildung 34.19**: Testergebnisse für zwei verschiedene Benutzer*



Wie die vorherige Bildschirmabbildung beweist, gibt es in Windows 7 RC noch einen Fehler: Obwohl das Skript „Hello1.ps1“ für den Benutzer „FoxMulder“ gesperrt ist, kann dieser es starten.

Erstellen neuer Regeln

Das Erstellen von Regeln kann aufwändig sein, wenn viele Anwendungen erlaubt sein sollen. Beim Erstellen von Regeln unterstützt die PowerShell. Mit `Get-AppLockerFileInformation` erhält man Informationen über Dateien in Form von Instanzen der Klasse `Microsoft.Security.ApplicationId.PolicyManagement.PolicyModel.FileInformation`. Diese Objekte enthalten zu der Datei den Path, Hash und Publisher.

Aus den Instanzen der Klasse `FileInformation` kann man mit `New-AppLockerPolicy`-Regeln generieren lassen. Diese Regeln kann man anschließend mit `Set-AppLockerPolicy` in lokalen Richtlinien oder Gruppenrichtlinien speichern.

Beispiel Der folgende Befehl erstellt genau eine Hashregel für alle Skripte in dem Ordner *c:\wps* und seinen Unterordnern. Die Regel wird in einer Gruppenrichtlinie gespeichert. Die bestehenden Regeln der Gruppenrichtlinie bleiben erhalten, was `-merge` bewirkt.

```
Get-AppLockerFileInformation -Directory c:\Wps\ -recurse -filetype
script |
New-AppLockerPolicy -RuleType Hash -User Everyone -optimize |
Set-AppLockerPolicy -merge
-LDAP:"LDAP://CN={685A9EAA-2CA5-4552-B553-
92A027F9E2B6},CN=Policies,CN=System,DC=FBI,DC=ORG"
```



Dass hier nur eine Regel insgesamt und nicht eine Regel pro Skript entsteht, liegt an dem Parameter `-optimize`.

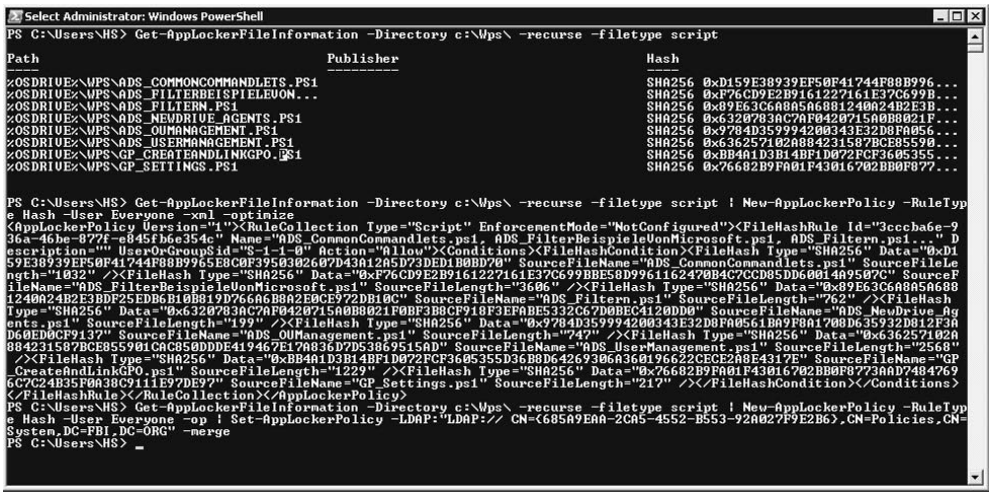


***Abbildung 34.20**: Der obige Befehl mit Zwischenschritten: Erst werden nur die Dateiinformationen ausgegeben, dann die erzeugte Regel in XML-Form (Parameter –xml). Zum Schluss wird die Regel in die Gruppenrichtlinie gesetzt (hier ohne –xml!).*

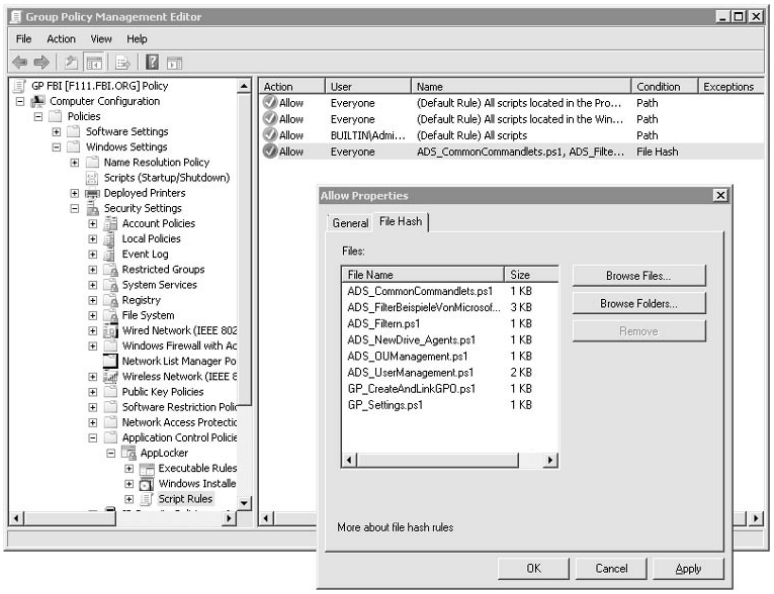


Abbildung 34.21 *Ergebnis der Ausführung des Befehls zur Erzeugung von Regeln für die Skripte in c:\wps*

35 Prozess-verwaltung

Die Verwaltung der laufenden Prozesse ist im Kern der Windows PowerShell mit den Commandlets `Get-Process` und `Start-Process` sowie `Stop-Process` gut unterstützt.

35.1 Prozesse auflisten

Get-Process

Das Commandlet `Get-Process` (Aliase `ps` oder `gps`) wurde schon im Kapitel mit der PowerShell-Einführung sehr häufig verwendet. Es liefert Instanzen der .NET-Klassen `System.Diagnostics.Process`.

Eine Liste aller Prozesse erhält man mit:

```
Get-Process
```

Informationen zu einem Prozess liefert der folgende Befehl: `Get-Process iexplore`

Eine Liste aller Prozesse, die mit einem „i“ anfangen, erhält man so: `Get-Process i*`

Ein interessanter Pipeline-Befehl ist:

```
Get-Process | Get-Item -ErrorAction SilentlyContinue | Group-Object Directory | Sort-Object Count -Descending
```

Mit ihm erhält man eine Liste aller Verzeichnisse, aus denen aktuelle Prozesse gestartet wurden. Das kann hilfreich sein, wenn man sich wundert, dass man ein Verzeichnis nicht löschen, verschieben oder umbenennen kann.

```
PS P:\> . 'H:\Demo\PowerShell\3_Einsatzgebiete\Prozessverwaltung\DirectoriesWithRunningProcesses.ps1'

Count Name                      Group
----- ----                      -----
   43 C:\Windows\system32       {C:\Windows\system32\AEADISRV.EXE, C:\Windows\system32\conhost.exe, C:\Win
    6 C:\Program Files (x86)... {C:\Program Files (x86)\Internet Explorer\iexplore.exe, C:\Program Files (
    3 C:\WINDOWS\system32\Wi... {C:\WINDOWS\system32\WindowsPowerShell\v1.0\powershell.exe, C:\WINDOWS\sys
    3 C:\Program Files (x86)... {C:\Program Files (x86)\TechSmith\SnagIt 8\SnagIt32.exe, C:\Program Files
    2 C:\Program Files (x86)... {C:\Program Files (x86)\Office2007\Office12\OUTLOOK.EXE, C:\PROGRA~2\OFFIC
    2 C:\Program Files (x86)... {C:\Program Files (x86)\Common Files\LightScribe\LightScribeControlPanel.e
    2 C:\Program Files (x86)... {C:\Program Files (x86)\Analog Devices\SoundMAX\SoundMAX.exe, C:\Program F
    2 C:\Program Files (x86)... {C:\Program Files (x86)\Microsoft Office\OFFICE11\POWERPNT.EXE, C:\Program
    2 C:\Program Files (x86)... {C:\Program Files (x86)\CyberLink\Shared files\brs.exe, C:\Program Files (
    2 C:\Program Files\Micro... {C:\Program Files\Microsoft IntelliPoint\dpupdchk.exe, C:\Program Files\Mi
    2 C:\Windows                {C:\Windows\Explorer.EXE, C:\Windows\splwow64.exe}
    1 C:\Program Files\Micro... {C:\Program Files\Microsoft SQL Server\MSSQL10.SQLEXPRESS\MSSQL\Binn\sqlse
    1 C:\Program Files (x86)... {C:\Program Files (x86)\Analog Devices\Core\smax4pnp.exe}
    1 C:\Program Files (x86)... {C:\Program Files (x86)\Microsoft SQL Server\90\Shared\sqlbrowser.exe}
    1 C:\Program Files\Micro... {C:\Program Files\Microsoft SQL Server\90\Shared\sqlwriter.exe}
    1 C:\Program Files\WinRAR   {C:\Program Files\WinRAR\WinRAR.exe}
    1 C:\Program Files (x86)... {C:\Program Files (x86)\Windows Live\Sync\WindowsLiveSync.exe}
    1 C:\Windows\system32\wbem  {C:\Windows\system32\wbem\wmiprvse.exe}
    1 C:\Windows\WindowsMobile  {C:\Windows\WindowsMobile\wmdc.exe}
    1 C:\Program Files (x86)... {C:\Program Files (x86)\TechSmith\Camtasia Studio 4\TSCHelp.exe}
    1 C:\Program Files (x86)... {C:\Program Files (x86)\Microsoft Team Foundation Server 2008 Power Tools\
    1 C:\Program Files (x86)... {C:\Program Files (x86)\VisagesoftwareXPert PDF\wspdfprsrv.exe}
    1 C:\Program Files (x86)... {C:\Program Files (x86)\Elaborate Bytes\VirtualCloneDrive\VCDDaemon.exe}
    1 C:\Windows\system32\in... {C:\Windows\system32\inetsrv\inetinfo.exe}
    1 C:\Program Files\Java\... {C:\Program Files\Java\jre6\bin\jusched.exe}
    1 C:\Program Files (x86)... {C:\Program Files (x86)\Java\jre6\bin\jusched.exe}
    1 C:\Program Files (x86)... {C:\Program Files (x86)\lg_fwupdate\fwupdate.exe}
    1 C:\Users\hs.ITU\Bluebirds {C:\Users\hs.ITU\Bluebirds\BlueBirds.exe}
    1 C:\Program Files (x86)... {C:\Program Files (x86)\CyberLink\Power2Go\CLMLSvc.exe}
    1 C:\Program Files (x86)... {C:\Program Files (x86)\Ilium Software\eWallet\eWallet.exe}
    1 C:\Program Files (x86)... {C:\Program Files (x86)\CyberLink\PowerDVD\PDVDServ.exe}
```

Abbildung 35.1
Ausgabe des obigen Befehls

35.2 Prozesse starten

Prozesse starten

Wenn man ein Commandlet oder eine Kommandozeilenanwendung in der PowerShell aufruft, dann starten diese im Prozess der PowerShell. Wenn man eine Windows-Anwendung (z.B. *Notepad.exe*) aufruft, dann startet diese in einem eigenen Prozess. In jedem Fall läuft der externe Prozess unter dem gleichen Benutzerkonto wie der aufrufende Prozess.

Mit `Start-Process` (Alias: saps) kann man ab PowerShell 2.0 Prozesse unter einer anderen Identität starten (vgl. runas.exe), d.h. mit anderen Rechten als denen des angemeldeten Benutzers. Man kann durch den Parameter `-Credential` ein Objekt vom Typ `PSCredential` mit anderen Anmeldedaten übergeben. Ein Objekt vom Typ `PSCredential` erhält man von `Get-Credential`.

Beispiel

Zum Starten eines zweiten PowerShell-Fensters unter einem anderen Benutzerkonto (als Administrator) gibt man also ein:

```
Start-Process PowerShell.exe -credential Get-credential
```

Dies dokumentieren die nachstehenden zwei Abbildungen.

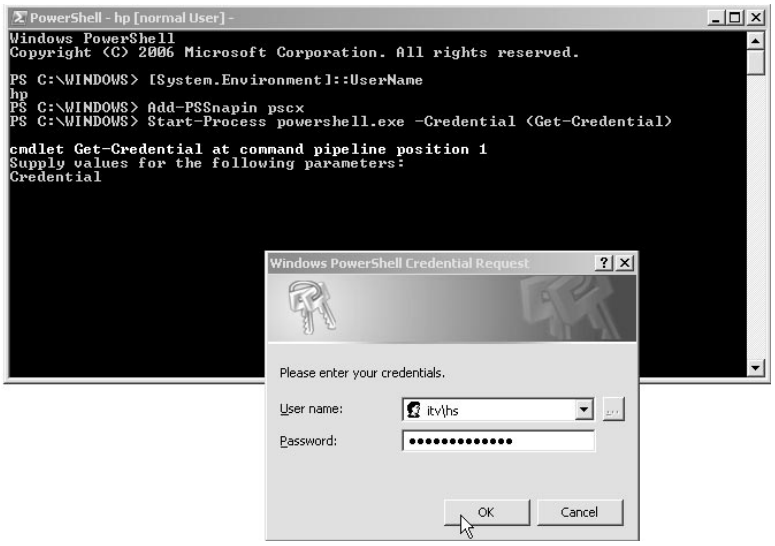


Abbildung 35.2 *Aufruf von Start-Process durch einen normalen Benutzer*



In PowerShell 1.0 konnte man `Start-Process` über die PowerShell Community Extensions nutzen.

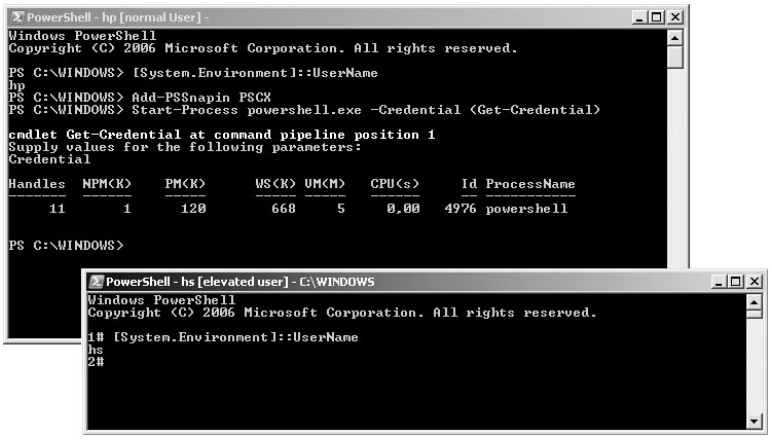


Abbildung 35.3 *Nach Eingabe der Anmeldedaten erhält man ein zweites PowerShell-Fenster für einen Benutzer, der zur Administratoren-Gruppe gehört.*

35.3 Prozesse beenden

Prozesse beenden

Einen Prozess beenden kann man wahlweise durch Aufruf der `Kill()`-Methode:

```
Get-Process | Where-Object { $_.name -eq "iexplore" } | Foreach-
Object { $_.Kill() }
```

oder prägnanter durch das Commandlet `Stop-Process`:

```
Stop-Process -name iexplore
```

Stop-Process

`Stop-Process` erwartet als Parameter normalerweise die Prozessnummer. Wenn man den Prozessnamen angeben will, muss man den Parameter `-name` verwenden.

Weitere Beispiele:

- Beenden aller Prozesse, deren Name mit „P“ anfängt:

  ```
  Get-Process p* | Stop-Process
  ```

- Beenden aller Prozesse, die mehr als 10 Megabyte RAM verbrauchen:

  ```
  Get-Process | where { $_.WS -gt 10M } | stop-process
  ```

35.4 Warten auf das Beenden einer Anwendung

WaitForExit()

Mit den folgenden Befehlen wartet die PowerShell darauf, dass Microsoft Outlook beendet wird:

Listing 35.1 Warten auf das Ende eines Prozesses [Einsatzgebiete/Prozessverwaltung/WaitingForOutlook.ps1]

```
$p = Get-Process  outlook

if ($p)
{
$p.WaitForExit()
"Outlook wurde beendet!"
}
else
{
"Outlook war nicht gestartet!"
}
```

Dies kann man ab PowerShell 2.0 auch mit `Wait-Process` erreichen. Der Prozess wird durch seinen Namen, seine Prozess-ID oder ein Process-Objekt festgelegt.

Beispiele:

```
Wait-Process -id 4040
Wait-Process -name notepad
Wait-Process -name $p
```

Hintergrundprozesse

Nicht vorhanden in der PowerShell 1.0 ist eine Prozessverwaltung für Hintergrundprozesse, wie man sie von Unix kennt. Dies kann der Administrator aber durch eigene Skripte erreichen, siehe [Truher01].

36 Systemdienste

Die Arbeit mit den Windows-Systemdiensten (oft auch „NT-Dienste“ genannt) unterstützt die PowerShell mit den Commandlets: **Get-Service**

- `Get-Service`
- `New-Service`
- `Restart-Service`
- `Resume-Service`
- `Set-Service`
- `Start-Service`
- `Stop-Service`
- `Suspend-Service`

Darüber hinaus steht die WMI-Klasse `Win32_Service` zur Verfügung.

36.1 Dienste auflisten

Eine Liste der Systemdienste in Form von Instanzen der .NET-Klasse `System.ServiceProcess.ServiceController` liefert das Commandlet `Get-Service`.

Die Liste der laufenden Systemdienste erhält man mit:

```
Get-Service | Where-Object {$_.status -eq "running"}
```

Die Liste der gestoppten Dienste liefert dementsprechend:

```
Get-Service | Where-Object {$_.status -eq "stopped"}
```

Man kann per Skript prüfen, ob ein Dienst installiert ist:

```
$service = Get-Service -name iisadmin
if ( ! $service ) { "IIS ist nicht installiert auf diesem
Computer." }
else
{ "IIS Server hat Status: " + $service.Status }
```

Listing 36.1 *Prüfung, ob der IIS installiert ist [Einsatzgebiete/Systemdienste/IsService installed.ps1]*

In PowerShell 1.0 war die Fernabfrage mit `Get-Service` nicht möglich. Seit PowerShell 2.0 gibt es den Parameter `-ComputerName`. **Fernabfrage**

```
Get-Service -ComputerName ServerEssen04| Where-Object {$_.status -
eq "running"}
```

In PowerShell 1.0 ist lediglich über den Umweg über die Windows Management Instrumentation (WMI) ein Zugang zu anderen Systemen möglich. Dafür steht das Commandlet `Get-WmiObject` zur Verfügung. Der folgende Befehl liest die laufenden Systemdienste von dem Computer mit Namen „ServerEssen04“ aus:

```
Get-WmiObject Win32_Service -computer ServerEssen04 -filter
"State='running'"
```

Zu beachten ist dabei, dass das Ergebnis der Operation nunmehr keine Instanzen von der .NET-Klasse `System.ServiceProcess.ServiceController`, sondern Instanzen der WMI-Klasse `root\cimv2\Win32_Service` sind, die in die .NET-Klasse `System.Management. ManagementObject` verpackt wurden. Das Commandlet `Get-Member` stellt diesen komplexen Typ wie folgt dar: "`System.Management.ManagementObject verpackt # root\cimv2\Win32_Service`". `Get-WmiObject` hat eine andere Filter-Syntax (hier ist das Gleichheitszeichen anstelle von `-eq` zu verwenden) und außerdem steht der Zustand eines Dienstes in der WMI-Klasse im Attribut `State` und nicht wie mit der .NET-Klasse in `Status`. Einsteiger kommen hier schnell durcheinander.

Wenn man die abhängigen Dienste eines Dienstes ermitteln will, muss man auf die Eigenschaft `DependentServices` des .NET-Objekts `System.ServiceProcess.ServiceController` zugreifen:

```
Get-service iisadmin | % { $_.DependentServices }
```

Das Ergebnis für Windows Server 2003 Release 2 zeigt *Abbildung 36.1*.

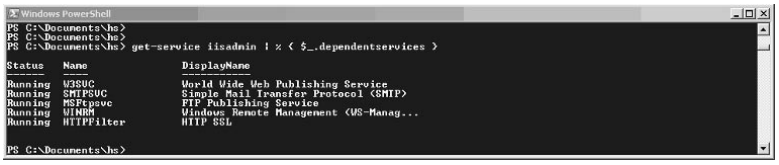


Abbildung 36.1
Die von IISAdmin abhängigen Dienste

Abhängige Systemdienste ermitteln

Die abhängigen Dienste eines Systemdienstes kann man alternativ auch in WMI durch die Methode `GetRelated()` in der Klasse `ManagementObject` in der .NET-Klassenbibliothek ermitteln. Der folgende Befehl ermittelt die Dienste, die von dem Dienst "`IISAdmin`" abhängig sind.

```
(Get-wmiObject win32_service -filter "Name =
'iisadmin'").GetRelated($null,"Win32_
DependentService",$null,$null,$null,"Antecedent",$false,$null) |
select name
```

Die gleiche Objektmenge erhält man auch über eine WQL-Abfrage mit Bezug auf den feststehenden Ausdruck `AssocClass`:

```
([wmiSearcher]"Associators of {Win32_Service.Name='iisadmin'} Where
AssocClass=Win32_DependentService Role=Antecedent").get()
```



Abbildung 36.2
Ermitteln der abhängigen Dienste

36.2 Dienstzustand ändern

Weitere Dienst-Commandlets

Zur Beeinflussung des Dienststatus stehen folgende Commandlets zur Verfügung: `Suspend-Service`, `Resume-Service`, `Stop-Service`, `Start-Service` und `Restart-Service`. Dabei ist jeweils der Dienstname als Parameter anzugeben.

Der folgende Befehl startet also den Dienst "`IISAdmin`":

```
Start-Service IISADMIN
```

Bei Systemdiensten, die abhängige Dienste besitzen, ist außerdem `-force` hinzuzufügen:

```
Start-Service IISADMIN -force
```

Da das Commandlet `Start-Service` (ebenso wie alle anderen Aktions-Commandlets rund um Dienste) nur für den lokalen Computer wirkt, muss man auf die WMI-Klasse `Win32_Service` zurückgreifen, um einen Dienst auf einem entfernten System zu starten. Leider wird bei diesen Commandlets der Schalter `-ComputerName` auch in der Version 2 der PowerShell noch nicht unterstützt.

Der folgende Befehl startet einen Systemdienst auf einem anderen Rechner:

```
Get-wmiobject -computer XY win32_service -filter "name='tlntsvr'" |
   foreach-Object {$_startservice()}
```



Das Commandlet `Restart-Service` führt einen Neustart eines Dienstes (erst stoppen, dann starten) durch. Wenn der Dienst nicht gestartet war, wird er gestartet.

36.3 Diensteigenschaften ändern

Die Eigenschaften von Diensten, z.B. die Startart, beeinflusst man über `Set-Service`:

```
Set-Service IISADMIN -startuptype "manual"
```

Für weitergehende Änderungen, z.B. Setzen des Dienstkontos, braucht man dann wieder WMI.

Listing 36.2 *ChangeService-Configuration.ps1*

```
# Change Service Configuration

"Before:"
Get-wmiobject Win32_Service -filter "name='WWWAppServer'" | select
startname, startmode

$service = Get-wmiobject Win32_Service -filter
"name='WWWAppServer'"
$service.change($null,$null,$null,$null,"manual",$null,"itv\hs",
"secret+123")

"After:"
Get-wmiobject Win32_Service -filter "name='WWWAppServer'" | select
startname, startmode
```

37 Netzwerk

Dieses Kapitel thematisiert Automatisierungsaufgaben rund um Netzwerkfunktionen. Dazu gehören die Konfiguration der Netzwerkkarten, die Namensauflösung, die Prüfung der Erreichbarkeit, das Abrufen von Informationen per HTTP und das Senden von E-Mails.

37.1 Netzwerkkonfiguration

IP-Adressen auflisten

In `Win32_NetworkAdapterConfiguration` sind die IP-Adressen als ein Array in `IpAddress` abgespeichert.

```
Get-Wmiobject Win32_NetworkAdapterConfiguration -Filter
"IPEnabled=true" | select Description,IPAddress
```

IP-Adresse vergeben

Die WMI-Klasse `Win32_NetworkAdapterConfiguration` ermöglicht auch zahlreiche Einstellungen für die Netzwerkkarten.

Das folgende PowerShell-Skript wechselt für eine Netzwerkkarte zwischen DHCP und statischer IP-Adresse.

Listing 37.1
Änderung der Netzwerkkonfiguration [Einsatzgebiete/Netzwerk/Switch_DHCP_StaticIP.ps1]

```
# Wechsel zwischen DHCP und statischer IP-Adresse
$config = Get-WmiObject Win32_NetworkadapterConfiguration -Filter
"IPEnabled=true" | where { $_.Description -like "*Controller #2*" }
"DHCP-Status Vorher: " + $Config.dhcpenabled
Get-WmiObject Win32_Networkadapterconfiguration -Filter
"IPEnabled=true" | select Description,IPAddress
if (!$Config.dhcpenabled)
{
"Aktiviere DHCP..."
$Config.EnableDHCP()
}
else
{
"Aktiviere statische IP-Addresse..."
[array] $ip =   "192.168.1.15"
[array] $subnet = "255.255.255.0"
$Config.EnableStatic($ip, $subnet)
}
$config = Get-WmiObject Win32_NetworkadapterConfiguration -Filter
"IPEnabled=true" | where { $_.Description -like "*Controller #2*" }
"DHCP-Status nachher: " + $Config.dhcpenabled
Get-WmiObject Win32_Networkadapterconfiguration -Filter
"IPEnabled=true" | select Description,IPAddress
```



Die WMI-Methode `EnableStatic()` funktioniert nur, wenn die Netzwerkkarte aktiviert ist.



Den aktuellen DHCP-Server ermitteln Sie durch das Commandlet `Get-DHCPServer` aus den PSCX.

Abbildung 37.1
Ausgabe des Beispiels bei zweimaligem Aufruf

```
PS J:\Demo\Skripte> .\netzwerk.ps1
DHCP-Status Vorher: True

Description                               IPAddress
-----------                               ---------
NVIDIA nForce Networking Controller ... {192.168.1.10, 192.168.1.11}
NVIDIA nForce Networking Controller ... {0.0.0.0}
Aktiviere Statische IP-Addresse...
0
DHCP-Status nachher: False
NVIDIA nForce Networking Controller ... {192.168.1.10, 192.168.1.11}
NVIDIA nForce Networking Controller ... {192.168.1.15}


PS J:\Demo\Skripte> .\netzwerk.ps1
DHCP-Status Vorher: False

Description                               IPAddress
-----------                               ---------
NVIDIA nForce Networking Controller ... {192.168.1.10, 192.168.1.11}
NVIDIA nForce Networking Controller ... {192.168.1.15}
Aktiviere DHCP...
0
DHCP-Status nachher: True
NVIDIA nForce Networking Controller ... {192.168.1.10, 192.168.1.11}
NVIDIA nForce Networking Controller ... {0.0.0.0}


PS J:\Demo\Skripte>
```

37.2 Erreichbarkeit prüfen (Ping)

Einen „Ping“ (Internet Control Message Protocol-/ICMP-Anfrage) zur Prüfung der Erreichbarkeit eines anderen Rechners (vgl. ping.exe) erzeugt man mit `Win32_PingStatus` oder `Test-Connection`.

Ping über WMI

Win32_PingStatus

Zur Prüfung der Erreichbarkeit eines Computers konnte man die WMI-Klasse `Win32_PingStatus` einsetzen:

```
Get-WmiObject Win32_PingStatus -filter "Address='www.Windows-
Scripting.de'" | select protocoladdress, statuscode, responsetime
```

Die PowerShell Community Extensions (PSCX) bieten auch ein Commandlet `Ping-Host`, das eine Datenstruktur des Typs `Pscx.Commands.Net.PingHostStatistics` liefert:

```
Ping-Host 'www.Windows-Scripting.de'
```

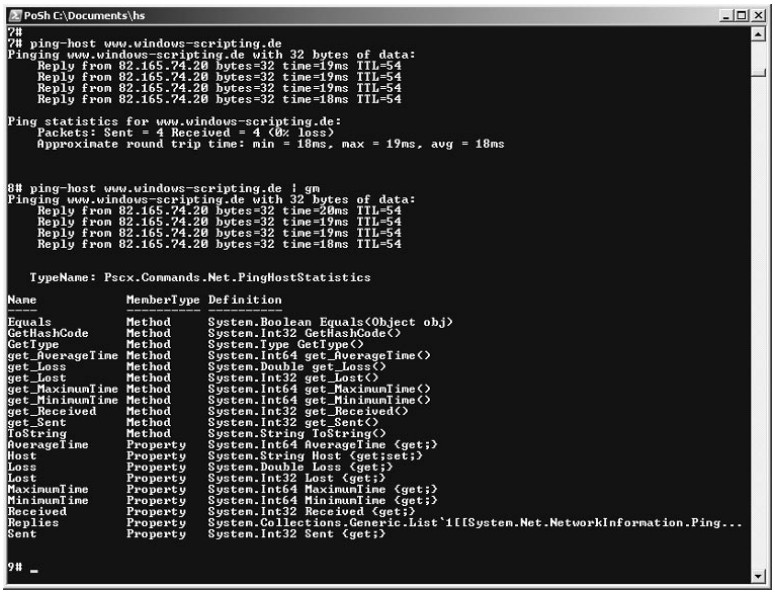


Abbildung 37.2
Anwendung von Ping-Host

Ping über Test-Connection

Ping

Ab PowerShell 2.0 gibt es ein eigenes Commandlet für „Ping". `Test-Connection` erwartet als Parameter mindestens einen Rechnernamen oder eine IP-Adresse. Beim Rückgabewert gibt es zwei Optionen: Entweder werden Instanzen der WMI-Klasse `Win32_PingStatus` geliefert oder aber nur ein Boolean-Wert (`True` oder `False`), wenn man dies mit `-Quiet` erzwingt.



Abbildung 37.3
Einsatz von Test-Connection

Mit `Test-Connection` kann man auch von einem oder mehreren dritten Computern einen Ping ausführen lassen. Der folgende Befehl führt jeweils zwei Pings von den drei Rechnern F171, F172 und F173 zu F111 aus.

```
Test-Connection -Source F171,F172,F173 -ComputerName F111 -count 2
```

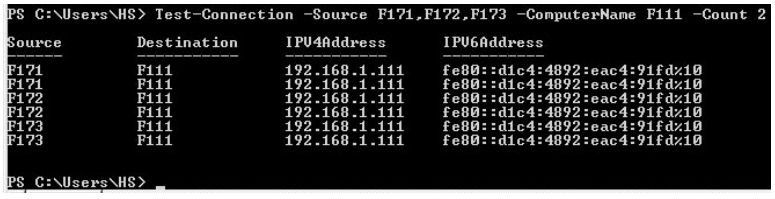


Abbildung 37.4
Beispiel für einen entfernten „Multi-Ping"



Für die Fernausführung eines Pings muss PowerShell-Remoting nicht aktiviert sein.

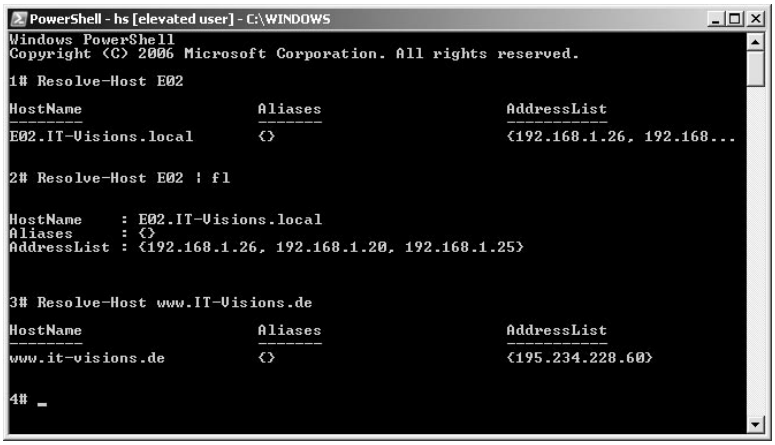
37.3 Namensauflösung

Resolve-Host

Zur Unterstützung der Namensauflösung gibt es das Commandlet `Resolve-Host` in den PSCX. Das Ergebnis ist eine Instanz der .NET-Klasse `System.Net.IPHostEntry`. Das Ergebnis der folgenden drei Beispiele sehen Sie in der Bildschirmabbildung:

- `Resolve-Host E02`
- `Resolve-Host E02 | fl`
- `Resolve-Host www.IT-Visions.de`

Abbildung 37.5
Einsatz von Resolve-Host



37.4 E-Mails senden

Zum Versenden einer E-Mail über SMTP gibt es folgende Möglichkeiten:

- .NET-Klassen im Namensraum `System.Net.Mail`
- `Send-SmtpMail` aus den PSCX
- `Send-MailMessage` (ab PowerShell 2.0)

E-Mail senden mit System.Net.Mail

SendSmtpMail

Zum Versenden einer E-Mail-Nachricht per SMTP können Sie die .NET-Klassen `System.Net.Mail.MailMessage` und `System.Net.Mail.SmtpClient` verwenden.

```
# Mail ohne Attachment
$client = New-Object System.Net.Mail.SmtpClient

$client.Host = "E01.IT-Visions.local"

$client.Send("hs@E01.IT-Visions.local","hp@E01.IT-
Visions.local","Test2", "Test TEst")

# Mail mit Anhang
[System.Net.Mail.MailMessage]  $message = New-Object
System.Net.Mail.MailMessage(
               "hs@E01.IT-Visions.local",
               "hp@E01.IT-Visions.local",
               "Aktueller Auftragsbericht.",
               "Siehe anliegender Bericht.")

$data = New-Object System.Net.Mail.Attachment("C:\TEMP\
Auftragsbericht.txt")

$client.Send($message)
```

Listing 37.2 *Beispiel zum Einsatz der .NET-Klassenbibliothek zum Senden einer E-Mail [Einsatzgebiete/Netzwerk/Email_FCL.ps1]*

E-Mail senden mit Send-SmtpMail

`Send-SmtpMail` ist ein Commandlet aus den PowerShell Community Extensions.

```
# Parameters
$Subject = "PowerShell Skript"
$Body = "Your daily script executed succefully!"
$From = "script@E01.IT-Visions.local"
$To = "hs@E01.IT-Visions.local"
$MailHost = "E01.IT-Visions.local"

# Send Mail
Send-SmtpMail -SmtpHost $MailHost -To $To -From $from -Subject
$subject  -Body $body
```

Listing 37.3 *Beispiel zum Einsatz von Send-SmtpMail [Einsatzgebiete/Netzwerk/SendMail_PSCX.ps1]*



Wenn eine Authentifizierung notwendig ist, können Sie diese mit dem Parameter `-Credential` und dem Commandlet `Get-Credential` anfordern. Dann aber fragt Windows immer durch einen Anmeldedialog nach einem Benutzerkonto und eine interaktionslose Ausführung ist nicht mehr möglich.

E-Mail senden mit Send-MailMessage

E-Mail

Mit `Send-MailMessage` (ab PowerShell 2.0) kann man eine E-Mail über SMTP senden. Die Parameter sind selbst erklärend. Auch hier gibt es den Parameter `-Credential`.

```
Send-Mailmessage -from "Alien Detection Skript <ADS@FBI.org>" -to
"Fox Mulder <fm@fbi.org>", "Dana Scully <ds@fbi.org" -subject "New
Alien detected" -body "Please see attachment" -Attachment "c:\data\
log.txt" -priority High -dno onSuccess, onFailure -smtpServer
smtp.FBI.org
```

37.5 Abruf von Daten von einem HTTP-Server

System.Net. WebClient

Das folgende Skript zeigt, wie man eine HTML-Seite von einem Webserver abruft. Zum Einsatz kommt hier die Klasse `System.Net.WebClient` aus der .NET-Klassenbibliothek. Diese Klasse bietet eine Methode `DownloadString()` an, die den Inhalt der angegebenen URL in einer Zeichenkette liefert. Mit Hilfe des Commandlets `Set-Content` wird die Zeichenkette dann im lokalen Dateisystem abgelegt. Die letzten vier Zeilen sind die Fehlerbehandlung, die dafür sorgt, dass bei Fehlern in dem Skript eine Meldung ausgegeben wird.

Listing 37.4 *Herunterladen einer Datei per HTTP [Einsatzgebiete/Netzwerk/HTTP-Download.ps1]*

```
# Eingabeparameter
$Url = "http://www.powershell-doktor.de"
$Ziel = "c:\temp\hauptseite.htm"

# Skript
Write-Host "Lade Webseite " $url "..."
$html = (New-Object System.Net.WebClient).DownloadString($Url)
$html | Set-Content -Path $ziel
Write-host "Heruntergeladene Seite wurde gespeichert unter " $Ziel

trap [System.Exception]
 {
    Write-host "Fehler beim Laden der URL: `"$url`"" `n
    exit
 }
```

Das nächste Beispiel zeigt, wie man die Titel der letzten acht Nachrichten aus einem RSS-Feed (Weblog) abruft. Auch hier kommt wieder `DownloadString()` aus der Klasse `System.Net.WebClient` zum Einsatz. Da der Inhalt in XML-Form vorliegt, ist eine Auswertung über die direkte Notation `$blog.RDF.item` möglich.

Listing 37.5 *Laden und Filtern eines RSS-Feeds [Einsatzgebiete/Netzwerk/RSS_Download.ps1]*

```
Write-Host "Aktuelle Nachrichten im Weblog von Dr. Holger
Schwichtenberg:"
$Url = " http://www.heise.de/developer/rss/dotnet-doktor/blog-
atom.xml"
$blog = [xml](New-Object System.Net.WebClient).DownloadString($Url)
$blog.RDF.item | select title -first 8
```



```
<?xml version="1.0" encoding="utf-8" ?>
- <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns="http://my.netscape.com/rdf/simple/0.9/">
  - <channel>
      <title>iX Blog - Der Dotnet-Doktor</title>
      <link>http://www.heise.de/ix/blog/1/</link>
      <description>Aktuelle Artikel im iX-Blog</description>
    </channel>
  - <item>
      <title>Fachbücher zu ASP.NET 2.0 erschienen</title>
      <link>http://www.heise.de/ix/blog/artikel/77803/from/rss09</link>
      <description>Mein Buch zu ASP.NET 2.0 gibt es jetzt sowohl in einer Variante
        mit Visual Basic 2005 als auch C# 2005.</description>
    </item>
  - <item>
      <title>Release Candidate 1 für Windows Vista und das .NET Framework
        3.0</title>
      <link>http://www.heise.de/ix/blog/artikel/77660/from/rss09</link>
      <description>Microsoft hat einen "Release Candidate" für das neue
        Betriebssystem Vista und für das Microsoft .NET Framework 3.0
        veröffentlicht.</description>
    </item>
  - <item>
      <title>Visual Studio 2005 für .NET 1.1 nutzen mit MSBee</title>
      <link>http://www.heise.de/ix/blog/artikel/77534/from/rss09</link>
      <description>Mit dem kostenlosen Add-On MSBuild Extras – Toolkit for .NET
        1.1 (MSBee) kann man mit Visual Studio 2005 Projekte auch in .NET-1.1-
        Code übersetzen lassen.</description>
    </item>
```

Abbildung 37.6
Beispiel für ein RSS-Dokument

In den PowerShell Community Extensions (PSCX) gibt es ein Commandlet `Get-HttpResource`, das die Verwendung von `DownloadString()` kapselt.

```
Get-HttpResource http://www.it-visions.de
```

Das Ergebnis ist eine Zeichenkette. Als Parameter kann man z.B. eine Browseridentifikationszeichenkette (UserAgent), für Websites, die verschiedene Inhalte für verschiedene Browser bieten, oder eine Zeichenkodierung (Encoding) angeben.

```
Get-HttpResource http://www.it-visions.de -Encoding Unicode
```

37.6 Aufrufe von Webdiensten

Webdienste

`New-WebServiceProxy` (ab PowerShell 2.0) ist ein sehr mächtiges Commandlet zum Aufruf von XML-Webservices. `New-WebServiceProxy` ist in der Lage, einen Webservice-Proxy dynamisch zur Laufzeit zu erzeugen und der PowerShell als Objekt zur Verfügung zu stellen.

Als Beispiel soll hier der öffentliche Webservice *http://www.it-visions.de/Webservices/leser.asmx* verwendet werden, über den sich Leser dieses Buchs für das Leser-Portal und den Newsletter registrieren können. Der Webservice bietet zwei Methoden:

- `RegisterReader(Buch, Name, Firma, E-Mail)` erlaubt die Registrierung eines Lesers. Rückgabetyp ist eine Zeichenkette.
- `GetBooks()` liefert eine Liste aller Bücher, zu denen es im Leser-Portal Unterstützung gibt. Rückgabetyp ist eine Instanz der .NET-Klasse `System.Data.DataTable`.

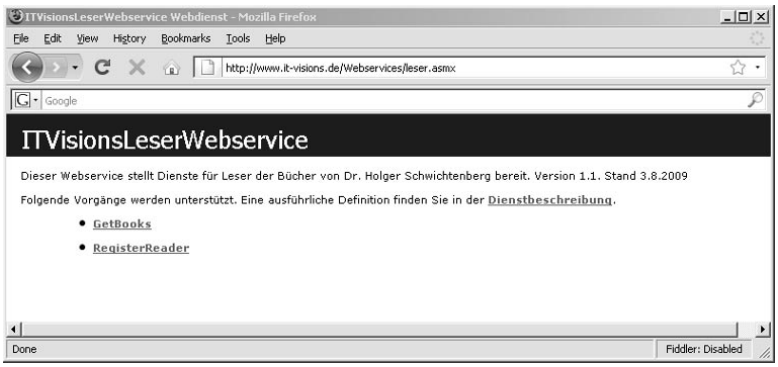


Abbildung 37.7 *HTML-Informationsseite zum Leser-Webservice*

Um einen Proxy zu erzeugen, muss man lediglich die URL angeben, unter der man die Dienstbeschreibung in der Web Service Description Language (WSDL) findet.

```
$ws = New-WebServiceProxy -URI http://www.IT-Visions.de/
Webservices/leser.asmx?WSDL
```

Hinweis: Bei einem .NET-basierten Webservice wie diesem dürfte man die Zeichenkette `?WSDL` am Ende auch weglassen.

Danach kann man sofort auf die von dem Webservice bereitgestellten Methoden zugreifen, z.B.:

```
$ws.RegisterReader("WS6","Holger Schwichtenberg","www.IT-
Visions.de","hs@IT-Visions.de")
```

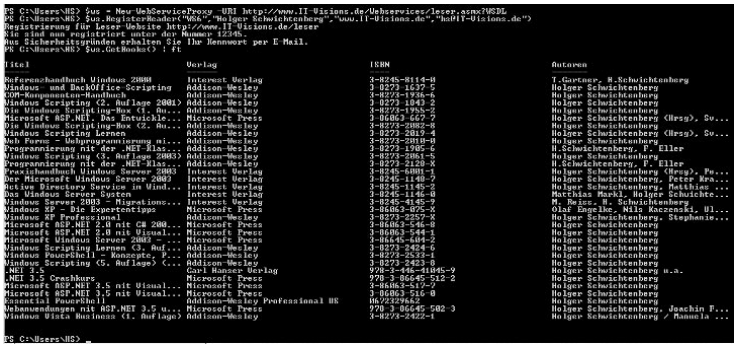


Abbildung 37.8 *Ergebnis des Aufrufs der beiden Webservice-Methoden*



Sie können sich gerne tatsächlich auf diesem Weg für das Leser-Portal registrieren. Ihr Kennwort erhalten Sie per E-Mail. Mit Ihrer E-Mail-Adresse und dem Kennwort können Sie sich dann unter *http://www.IT-Visions.de/leser* anmelden. „WS6“ beschreibt in dem obigen Aufruf das Buch „Windows Scripting, 6. Auflage“.

Weitere Möglichkeiten des Commandlets `New-WebserviceProxy`: Über den Schalter `–UseDefaultCredential` können Sie Ihre aktuellen Windows-Anmeldedaten an den Webservice weitergeben, wenn dieser eine Authentifizierung verlangt. Über den Schalter `-Credential Get-Credential` können Sie auch eine andere Identität übergeben.

38 Ereignisprotokolle

Für die Windows-Ereignisprotokolle gibt es ab PowerShell 2.0 zusätzlich zu dem `Get-EventLog` aus PowerShell 1.0 auch noch die Commandlets `New-EventLog`, `Remove-EventLog` und `Write-EventLog` sowie `Limit-EventLog`.

38.1 Protokolleinträge auslesen

Get-EventLog

Informationen über und aus Ereignisprotokollen stellt das Commandlet `Get-EventLog` bereit.

Eine Liste der auf dem System verfügbaren Ereignisprotokolle liefert:

```
Get-EventLog -list
```

System.Diagnostics.EventLog

Das Ergebnis sind Instanzen der Klasse `System.Diagnostics.EventLog`.

Ruft man hingegen das Commandlet `Get-EventLog` ohne den Parameter `-list` und stattdessen mit dem Namen eines Ereignisprotokolls auf, liefert das Commandlet alle Einträge in dem Ereignisprotokoll in Form von Objekten des Typs `System.Diagnostics.EventLogEntry`.

```
Get-EventLog Application
```

Ereignisprotokolleinträge des heutigen Tages

Hier ist eine Einschränkung sinnvoll, weil die Operation sonst sehr lange dauert. Das Commandlet besitzt eine eingebaute Filterfunktion:

```
Get-EventLog Application -newest 30
```

Mit einer kleinen Hilfsroutine ist es möglich, die Protokolleinträge auf die Einträge des heutigen Tages zu begrenzen:

Listing 38.1 Protokolleinträge von heute [Einsatzgebiete/Ereignisprotokolle/ EventLog_Misc.ps1]

```
function isToday ([datetime]$date)
{[datetime]::Now.Date  -eq  $date.Date}

Get-EventLog Application -newest 2048 |where {isToday $_
.TimeWritten}
```

Oder alle Einträge der letzten drei Tage abzurufen:

Listing 38.2 Protokolleinträge der letzten drei Tage [Einsatzgebiete/Ereignisprotokolle/EventLog_Misc.ps1]

```
function isWithin([int]$days, [datetime]$Date)
{
    [DateTime]::Now.AddDays($days).Date -le $Date.Date
}

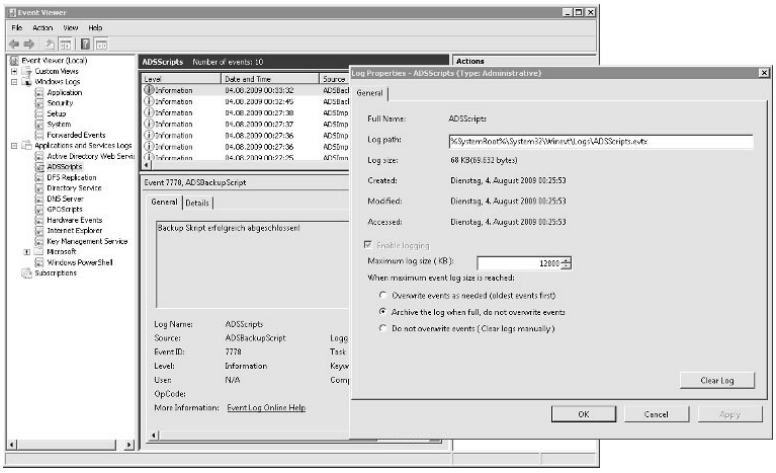
Get-EventLog Application |where {isWithin -3 $_.TimeWritten}
```

Interessant ist es, die Einträge nach Ereignisnummer zu gruppieren, um wiederkehrende Probleme zu identifizieren:

```
Get-EventLog Application | Group-Object eventid | Sort-Object Count
```

`Show-EventLog` (ab PowerShell 2.0) ist ein außergewöhnliches Commandlet für eine konsolenbasierte Shell, weil es die grafische Benutzerschnittstelle der Windows-Ereignisanzeige öffnet.

Abbildung 38.1 Ein neues Ereignisprotokoll mit Einträgen



38.2 Ereignisprotokolle erzeugen

New-EventLog

Mit `New-EventLog` kann man ein neues Ereignisprotokoll mit einer Ereignisquelle in Windows anlegen:

```
New-EventLog -LogName "ADSScripts" -Source "ADSImportScript"
```

Um für ein bestehendes Ereignisprotokoll eine neue Quelle zu registrieren, wiederholt man den Befehl mit einer anderen Zeichenkette bei „-source“:

```
New-EventLog -LogName "GPOScripts" -Source "GPOBackupScript"
```

38.3 Protokolleinträge erzeugen

Write-EventLog

Mit `Write-EventLog` kann man in das Ereignisprotokoll schreiben:

```
Write-EventLog -LogName ADSScripts -Source ADSBackupScript -EventID
7778 -Message "Backup Skript erfolgreich abgeschlossen!" -EntryType
Information
```

38.4 Protokollgröße festlegen

Limit-EventLog

Die Größe des Ereignisprotokolls kann man einschränken:

```
Limit-EventLog -LogName ADSScripts -MaximumSize 256KB -
OverFlowAction OverWriteAsNeeded
```

38.5 Protokolleinträge löschen

ClearEventlog

`Clear-Eventlog` löscht alle Einträge aus einem Protokoll. Mit `Remove-EventLog` kann man das Ereignisprotokoll zusammen mit allen vorhandenen Einträgen löschen. Beide Commandlets haben im Standard keine Sicherheitsabfrage.

39 Leistungsdaten

Zum Zugriff auf Leistungsindikatoren gibt es zwei Möglichkeiten:

- Zugriff über WMI (ab PowerShell 1.0)
- Commandlets `Get-Counter` (ab PowerShell 2.0)

39.1 Zugriff auf Leistungsindikatoren über WMI

Win32_PerfRawData

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`.

Informationen über die Speichernutzung der laufenden Prozesse liefert:

```
Get-Wmiobject Win32_PerfRawData_PerfProc_Process | select
Name,Workingset
```

Daten über den verfügbaren Hauptspeicher gibt es hier:

```
Get-Wmiobject Win32_PerfRawData_PerfOS_Memory
```

Die Auslastung des Prozessors kann man so auslesen:

```
Get-Wmiobject Win32_PerfRawData_PerfOS_Processor
```



`Win32_PerfRawData` ist die abstrakte Basisklasse über alle Leistungsdatenklassen. Den Befehl `Get-WmiObject Win32_PerfRawData` sollten Sie aber nicht ausführen, da Sie sonst sehr, sehr viele Objekte erhalten.

39.2 Get-Counter

Leistungsindikatoren

`Get-Counter` dient ab PowerShell 2.0 der Abfrage von Leistungsindikatoren (Performance Countern).

`Get-Counter` ohne Parameter liefert einige ausgewählte der vielen Leistungsindikatoren.

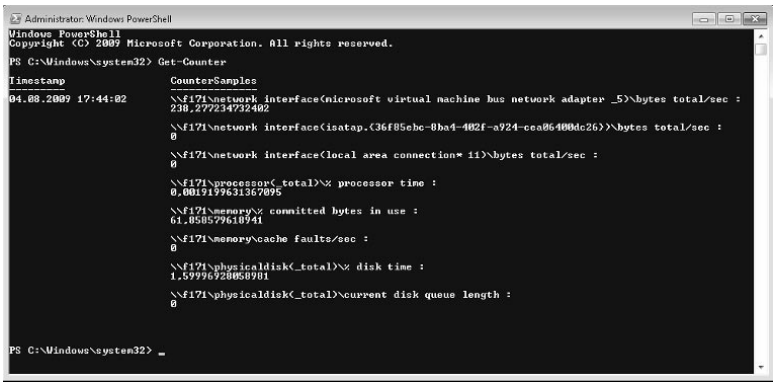


Abbildung 39.1
Standardausgabe von Get-Counter

Die Leistungsindikatoren sind in Mengen organisiert. Der Aufruf

```
Get-counter -listset *
```

liefert eine Liste der Mengen.

Eine Liste der einzelnen Leistungsindikatoren, die jeweils über einen Pfad adressiert werden, bekommt man über:

```
Get-counter -listset * | foreach { $_.Paths }
```

Unter Angabe des Pfads kann man gezielt einzelne Leistungsindikatoren abfragen:

```
Get-Counter "\processor(_total)\% processor time"
```

Auch Fernzugriffe sind möglich unter Angabe des Rechnernamens im Pfad oder im Parameter `-computername`:

```
Get-counter "\\F111\processor(_total)\% processor time"
Get-counter "\processor(_total)\% processor time" -computername f111
```



Wenn man sowohl im Pfad als auch als Parameter einen Computernamen angibt, wird der Parameter ignoriert!

Durch die Angabe von `-Continuous` erhält man fortwährende Ergebnisse bis zum Abbrechen mit STRG + C. Das Intervall ist eine Sekunde. Dies kann man mit `-sampleinterval` ändern.

```
Get-Counter "\\F111\processor(_total)\% processor time"  -Continuous
-sampleinterval 2
```

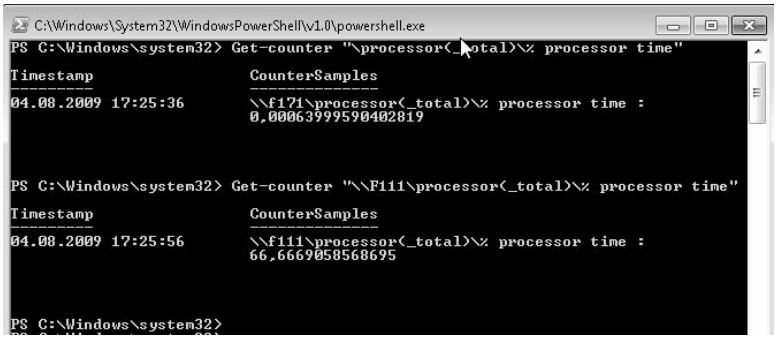


Abbildung 39.2
Get-Counter mit gezielten Pfaden



Mit `Export-Counter` kann man Daten von Leistungsindikatoren exportieren als .blg-Datei (binäre Protokolldatei), .csv (Comma Separated Value) oder .tsv (Tabulator Separated Value). Diese Daten kann man später mit `Import-Counter` wieder importieren.

Sicherheitseinstellungen

Get-Acl
Set-Acl

Ressourcen wie Dateisystemobjekte und Registrierungsdatenbankeinträge werden durch Zugriffsrechtelisten (Access Control Lists – ACLs) geschützt. Die PowerShell bietet zwei eingebaute Commandlets für die Arbeit mit ACLs:

- `Get-Acl`
- `Set-Acl`

Diese erledigen die Grundfunktionen des Ladens und Speicherns einer Zugriffsrechteliste abhängig von dem abgegebenen Ressourcenpfad. Derzeit unterstützt werden aber nur das Dateisystem und die Registrierungsdatenbank.



Neben den oben genannten Commandlets ist auch Wissen aus dem .NET-Namensraum `System.Security.AccessControl` erforderlich.

39.3 Grundlagen

Zum besseren Verständnis der Nutzung und der Veränderungen von Sicherheitseinstellungen seien an dieser Stelle kurz die Grundlagen der Windows-Sicherheit dargestellt.

Security Identifier (SID)

SID

Jeder Benutzer und jede Benutzergruppe besitzen einen sogenannten *Security Identifier* (kurz: *SID*), der den Benutzer bzw. die Gruppe eindeutig identifiziert. Ein SID ist ein Zahlenarray variabler Länge.

Security Descriptor (SD)

SD Jedes Objekt (z.B. eine Datei, ein Dateiordner, ein Eintrag im Active Directory, ein Registrierungsschlüssel) besitzt zur Speicherung der Zugriffsrechte einen sogenannten *Security Descriptor* (kurz: *SD*; dt. Sicherheitsbeschreiber). Ein SD besteht aus drei Teilen:

- Aus dem *Security Identifier (SID)* des Besitzers. Ein SID ist ein Zahlenarray variabler Länge.
- Aus einer *Discretionary ACL (DACL)*, welche die Zugriffsrechte beschreibt.
- Aus einer *System ACL (SACL)*, welche die Überwachungseinstellungen enthält.

Access Control List (ACL)

ACE, ACL Eine *Access Control List (ACL)* (sowohl DACL als auch SACL) besteht aus *Access Control Entries (ACE)*. Eine ACE wiederum enthält folgende Informationen:

- **Identity** (alias **Trustee**): der SID des Benutzers bzw. der Benutzergruppe.
- **Access Mask**: Die Zugriffsmaske definiert die Rechte. Für jeden Objekttyp (z.B. Dateisystemeintrag, Registrierungsdatenbankeintrag, Active-Directory-Eintrag) gibt es unterschiedliche Rechte. Jedes Recht ist dabei ein Bit bzw. eine Kombination von Bits in diesem Long-Wert. Eine Zugriffsmaske besteht in der Regel aus der Addition mehrerer einzelner Zugriffsrechte.
- **Access Control Type**: Der Typ ist entweder Zulassen (ALLOW) oder Verbieten (DENY).
- **Inheritance Flags**: Über die Inheritance Flags wird die Vererbung der Rechte nach unten im Baum gesteuert. `ObjectInherit` bedeutet, dass die ACE ihre Einstellung an untergeordnete Blatt-Objekte (z.B. Dateien im Dateisystem) vererbt. `ContainerInherit` bedeutet, dass die ACE ihre Einstellung an untergeordnete Container-Objekte (z.B. Ordner im Dateisystem) vererbt. `ObjectInherit` und `ContainerInherit` können miteinander kombiniert werden. Alternativ kann keine Vererbung (NONE) definiert werden.
- **Propagation Flags**: Über die Propagation Flags erfolgt eine weitere Steuerung der Vererbung. `InheritOnly` bedeutet, dass die ACE nur vererbt wird, aber nicht bei dem aktuellen Objekt selbst wirkt. `NoPropagateInherit` bedeutet, dass die ACE vererbt wird, aber nicht von den erbenden Objekten nochmals weitervererbt werden darf.

Zugriffsmasken Die folgende Liste enthält beispielhaft die möglichen Rechte für Einträge im Dateisystem.



Die folgende Tabelle wird aus der MSDN-Dokumentation unverändert zitiert [MSDN01]. Autor der Tabelle ist Microsoft.

Recht	Beschreibung
AppendData	Gibt die Berechtigung an, Daten an das Ende einer Datei anzufügen.
ChangePermissions	Gibt die Berechtigung an, die einer Datei zugeordneten Sicherheits- und Überwachungsregeln zu ändern.
CreateDirectories	Gibt die Berechtigung an, einen Ordner zu erstellen. Für diese Berechtigung ist der Synchronize-Wert erforderlich. Beachten Sie, dass der Synchronize-Wert beim Erstellen einer Datei oder eines Ordners automatisch festgelegt wird, wenn Sie ihn nicht explizit festlegen.
CreateFiles	Gibt die Berechtigung an, eine Datei zu erstellen. Für diese Berechtigung ist der Synchronize-Wert erforderlich. Beachten Sie, dass der Synchronize-Wert beim Erstellen einer Datei oder eines Ordners automatisch festgelegt wird, wenn Sie ihn nicht explizit festlegen.
Delete	Gibt die Berechtigung an, einen Ordner oder eine Datei zu löschen.
DeleteSubdirectoriesAndFiles	Gibt die Berechtigung an, einen Ordner und sämtliche in diesem Ordner enthaltenen Dateien zu löschen.
ExecuteFile	Gibt die Berechtigung an, eine Anwendungsdatei auszuführen.
FullControl	Gibt die Berechtigung für einen Vollzugriff auf eine Datei oder einen Ordner an sowie die Berechtigung, die Zugriffs- und Überwachungsregeln zu ändern. Dieser Wert stellt die Berechtigung dar, jede mögliche Aktion für diese Datei durchzuführen. Er ist eine Kombination aller Werte dieser Enumeration.
ListDirectory	Gibt die Berechtigung an, den Inhalt eines Verzeichnisses zu lesen.
Modify	Gibt die Berechtigung an, den Inhalt eines Ordners zu lesen, zu schreiben und aufzulisten, Dateien und Ordner zu löschen und Anwendungsdateien auszuführen. Diese Berechtigung schließt die Berechtigungen ReadAndExecute, Write und Delete ein.
Read	Gibt die Berechtigung an, Ordner oder Dateien schreibgeschützt zu öffnen und zu kopieren. Diese Berechtigung schließt die Berechtigungen ReadData, ReadExtendedAttributes, ReadAttributes und ReadPermissions ein.
ReadAndExecute	Gibt die Berechtigung an, Ordner oder Dateien schreibgeschützt zu öffnen und zu kopieren und Anwendungsdateien auszuführen. Diese Berechtigung schließt die Read-Berechtigung und die ExecuteFile-Berechtigung ein.
ReadAttributes	Gibt die Berechtigung an, Dateisystemattribute einer Datei oder eines Ordners zu öffnen und zu kopieren. Dieser Wert gibt z.B. die Berechtigung an, das Erstellungsdatum oder das Änderungsdatum einer Datei zu lesen. Dies schließt nicht die Berechtigung ein, Daten, erweiterte Dateisystemattribute oder Zugriffs- und Überwachungsregeln zu lesen.

***Tabelle 39.1**: Zugriffsrechte auf dem Windows-Dateisystem (Quelle: [MSDN01]*

Recht	Beschreibung
ReadData	Gibt die Berechtigung an, eine Datei oder einen Ordner zu öffnen und zu kopieren. Dies schließt nicht die Berechtigung ein, Dateisystemattribute, erweiterte Dateisystemattribute oder Zugriffs- und Überwachungsregeln zu lesen.
ReadExtendedAttributes	Gibt die Berechtigung an, erweiterte Dateisystemattribute einer Datei oder eines Ordners zu öffnen und zu kopieren. Dieser Wert gibt zum Beispiel die Berechtigung an, den Autor oder Inhaltsinformationen anzuzeigen. Dies schließt nicht die Berechtigung ein, Daten, Dateisystemattribute oder Zugriffs- und Überwachungsregeln zu lesen.
ReadPermissions	Gibt die Berechtigung an, Zugriffs- und Überwachungsregeln für eine Datei oder einen Ordner zu öffnen und zu kopieren. Dies schließt nicht die Berechtigung ein, Daten, Dateisystemattribute oder erweiterte Dateisystemattribute zu lesen.
Synchronize	Gibt an, ob die Anwendung warten kann, bis ein Dateihandle mit dem Abschluss eines E/A-Vorgangs synchronisiert ist. Der Synchronize-Wert wird automatisch festgelegt, wenn der Zugriff gewährt wird, und automatisch ausgeschlossen, wenn der Zugriff verweigert wird. Für die Berechtigung, eine Datei oder einen Ordner zu erstellen, ist dieser Wert erforderlich. Beachten Sie, dass dieser Wert beim Erstellen einer Datei automatisch festgelegt wird, wenn Sie ihn nicht explizit festlegen.
TakeOwnership	Gibt die Berechtigung an, den Besitzer eines Ordners oder einer Datei zu ändern. Beachten Sie, dass Besitzer einer Ressource über einen Vollzugriff auf diese Ressource verfügen.
Traverse	Gibt die Berechtigung an, den Inhalt eines Ordners aufzulisten und in diesem Ordner enthaltene Anwendungen auszuführen.
Write	Gibt die Berechtigung an, Ordner und Dateien zu erstellen, Dateien und Daten hinzuzufügen und Daten aus Dateien zu entfernen. Diese Berechtigung schließt die Berechtigungen WriteData, AppendData, WriteExtendedAttributes und WriteAttributes ein.
WriteAttributes	Gibt die Berechtigung an, Dateisystemattribute einer Datei oder eines Ordners zu öffnen und zu schreiben. Dies schließt nicht die Berechtigung ein, Daten, erweiterte Attribute oder Zugriffs- und Überwachungsregeln zu schreiben.
WriteData	Gibt die Berechtigung an, eine Datei oder einen Ordner zu öffnen und in die Datei bzw. den Ordner zu schreiben. Dies schließt nicht die Berechtigung ein, Dateisystemattribute, erweiterte Dateisystemattribute oder Zugriffs- und Überwachungsregeln zu öffnen und zu schreiben.
WriteExtendedAttributes	Gibt die Berechtigung an, erweiterte Dateisystemattribute einer Datei oder eines Ordners zu öffnen und zu schreiben. Dies schließt nicht die Berechtigung ein, Daten, Attribute oder Zugriffs- und Überwachungsregeln zu schreiben.

***Tabelle 39.1**: Zugriffsrechte auf dem Windows-Dateisystem (Quelle: [MSDN01] (Forts.)*

Programmierschnittstellen

System. Security. AccessControl

Der Namensraum `System.Security.AccessControl` enthält zahlreiche Klassen zur Verwaltung von Berechtigungen (Access Control Lists, ACLs). Dieser Namensraum wird insbesondere von den Klassen `System.IO.File`, `System.IO.Directory`, `Microsoft.Win32.RegistryKey` und `System.Threading.Semaphore` verwendet. Für jede Art von Ressource, deren ACLs verwaltet werden können, bietet der Namensraum `AccessControl` eine Klasse an, die von `System.Security.AccessControl.ObjectSecurity` abgeleitet ist. Beispielsweise dient `System.Security.AccessControl.FileSecurity` dazu, die ACLs einer Datei im Dateisystem zu lesen und zu verarbeiten.

Die folgende Abbildung zeigt diese Klassen im Vererbungsbaum der .NET-Klassenbibliothek. Die anderen dort genannten Ressourcen (z.B. Active Directory) können derzeit noch nicht über `Get-Acl` abgefragt werden. Hier ist aber eine direkte Ansprache über die .NET-Klassenbibliothek möglich.

Über die gesamte .NET-Klassenbibliothek verteilt findet man Klassen, die eine Methode `GetAccessControl()` besitzen, die ein von der Klasse `ObjectSecurity` abgeleitetes Objekt liefert. Beispiele für solche Klassen sind:

- `System.IO.File`
- `System.IO.Directory`
- `System.IO.FileInfo`
- `System.IO.DirectoryInfo`
- `Microsoft.Win32.RegistryKey`
- `System.Threading.Semaphore`

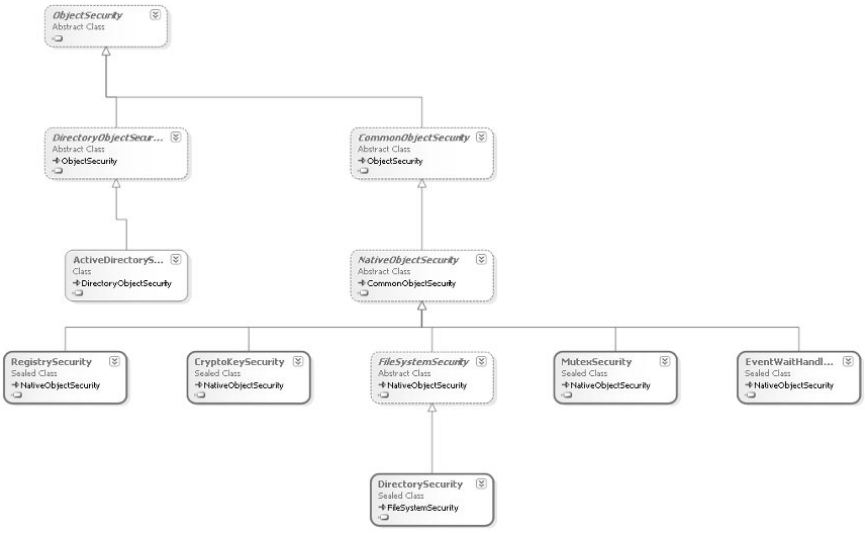


***Abbildung 39.3**: Vererbungshierarchie der Klassen zur ACL-Speicherung*

Die Basisklasse `ObjectSecurity` vererbt u.a. folgende Mitglieder, so dass diese in allen untergeordneten Klassen zur Verfügung stehen:

- `GetOwner()`: ermittelt den Besitzer des Objekts
- `SetOwner()`: setzt den Besitzer
- `GetAccessRules()`: liefert eine Liste der Rechteeinträge (Access Control Entry – ACE). Der Rückgabewert hat den Typ `AuthorizationRuleCollection`. Die enthaltenen Elemente sind vom Ressourcentyp abhängig (z.B. `FileSystemAccessRule` oder `RegistryAccessRule`).
- `GetAuditRules()`: liefert die Einträge der System-ACL (SACL).
- `IsSddlConversionSupported`: zeigt an, ob die Zugriffsrechteliste in SDDL ausgedrückt werden kann.
- `GetSecurityDescriptorSddlForm()`: liefert die Zugriffsrechteliste als SDDL-Zeichenkette.

Kontenname und SID

Principal

Der Namensraum `System.Security.AccessControl` verwendet Klassen aus `System.Security.Principal` zur Darstellung der Berechtigungsträger (Benutzer und Gruppen). `System.Security.Principal` unterstützt die beiden in Windows bekannten Bezeichner für Berechtigungsträger:

- Prinzipalname (z.B. „ITVisions\hs“) durch die Klasse `System.Security.Principal.NTAccount`
- Security Identifier (z.B. `S-1-5-21-565061207-3232948068-1095265983-500`) durch die Klasse `System.Security.Principal.SecurityIdentifier`

Jeder Benutzer und jede Benutzergruppe besitzen einen sogenannten Security Identifier (kurz: SID), der den Benutzer bzw. die Gruppe eindeutig identifiziert. Ein SID ist ein Zahlen-Array variabler Länge. In Textform wird der SID mit einem beginnenden „S“ dargestellt.

39.4 Zugriffsrechtelisten auslesen

Get-Acl

`Get-Acl` liefert abhängig vom Ressourcentyp Instanzen folgender .NET-Klassen:

- `System.Security.AccessControl.DirectorySecurity` (für Verzeichnisse)
- `System.Security.AccessControl.FileSecurity` (für Dateien)
- `System.Security.AccessControl.RegistrySecurity` (für Registrierungsdatenbankschlüssel)

`Get-Acl` erwartet als Parameter den Pfad der Ressource, deren Zugriffsrechteliste ermittelt werden soll, z.B.:

- `Get-Acl hklm:/software/www.IT-visions.de`
- `Get-Acl g:\daten\kunden`
- `Get-Acl g:\daten\kunden\Kundenliste.csv`

Die Standardausgabe erfolgt mit `Format-Table`. Die Ausgabe mit `Format-List` bietet sich an, damit die Ausgabe besser lesbar ist.

Die folgende Abbildung zeigt die Anwendung von `Get-Acl` auf ein Verzeichnis im Dateisystem. Die *Abbildung 39.5* zeigt die gleiche Zugriffsrechteliste im Windows Explorer.

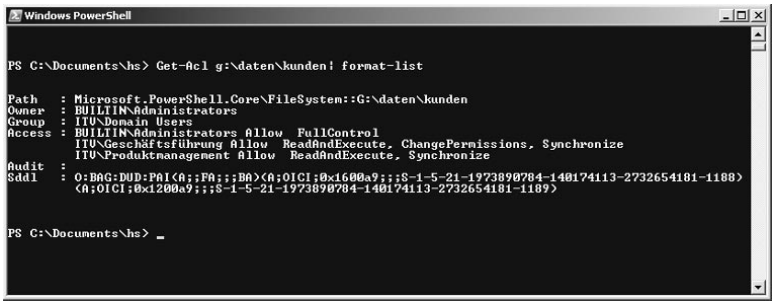


Abbildung 39.4
Beispiel für das Auslesen einer ACL

`Access` ist keine Eigenschaft der .NET-Klasse `ObjectSecurity`, sondern ein PowerShell Code Property, intern wird hier `GetAccessRules()` aufgerufen. Der Rückgabewert ist in beiden Fällen eine `AuthorizationRuleCollection`.



Abbildung 39.5
Tatsächliche Einstellungen

39.5 Einzelne Rechteeinträge auslesen

GetAccessRules()

Möchte man die einzelnen Rechteeinträge eines Systembausteins genauer betrachten, dann sollte man selbst über die Rechteliste iterieren. Die von Access bzw. `GetAccessRules()` gelieferte Liste des Typs `AuthorizationRuleCollection` enthält im Falle des Dateisystems Objekte vom Typ `FileSystemAccessRule`. Diese Objekte enthalten wiederum folgende Attribute:

- `IdentityReference`: Subjekt (Benutzer oder Gruppe), das Rechte hat
- `FileSystemRights`: Rechte
- `AccessControlType`: Rechteart (erlaubt oder verboten)
- `IsInherited`: zeigt an, ob die Regel geerbt ist
- `InheritanceFlags`: zeigt die Art der Vererbung nach unten an

Benutzerkonten können in zwei Formen ausgedrückt werden: im Klartext oder durch Security Identifiers (SIDs). Bei der Verwendung von `GetAccessRules()` muss man angeben, wie man die Benutzer sehen will: `[System.Security.Principal.NTAccount]` (Klartext) oder `[System.Security.Principal.SecurityIdentifier]` (SID). Davor besitzt die Methode noch zwei Parameter, mit denen man steuern kann, welche Regeln man sehen will: die explizit auf dem Objekt gesetzten Regeln (erste Parameter) und/oder die vererbten Regeln (zweite Parameter). Die expliziten ACEs erscheinen immer zuerst in der Liste.

Der Zugriff auf das Code Property Access ist gleichbedeutend mit `GetAccessRules ($true, $true, [System.Security.Principal.NTAccount])`. Möchte man andere Informationen, muss man `GetAccessRules()` explizit nutzen. In dem folgenden Beispiel werden beim zweiten Ausgeben der Liste nur die geerbten Regeln in SID-Form aufgeführt.

Listing 39.1 *Details aus ACEs auslesen [Einsatzgebiete/Sicherheitseinstellungen/Filesystem_ACL_Read.ps1]*

```
$a = Get-Acl g:\daten\kunden

# Rechte als NT-Konten holen
$aces =$a.access
# entspricht: $aces =$a.GetAccessRules($true, $true,
[System.Security.Principal.NTAccount])

Write-Host "Alle Regeln:" -F yellow
foreach ($ace in $aces)
{
Write-host $ace.IdentityReference.ToString() "hat Zugang"
$ACE.FileSystemRights $ACE.AccessControlType "Vererbt?"
$ACE.IsInherited
}

$aces =$a.GetAccessRules($true, $false,
```

```
[System.Security.Principal.SecurityIdentifier])

Write-Host  "Nur die expliziten Regeln, in SID-Form:"  -F yellow
foreach ($ace in $aces)
{
Write-host $ace.IdentityReference.ToString() "hat Zugang"
$ACE.FileSystemRights $ACE.AccessControlType "Vererbt?"
$ACE.IsInherited
}
```



Abbildung 39.6
Ausgabe des obigen Skripts

39.6 Besitzer auslesen

GetOwner()

Den Besitzer eines Systembausteins liest man über das Code Property `Owner` aus dem von `ObjectSecurity` abgeleiteten und von der PowerShell erweiterten Objekt aus, das `Get-Acl` zurückgibt. Alternativ kann man auch `GetOwner()` verwenden und hat dabei wieder die Wahl der Form. Zwischen den beiden Formen der Benutzerdarstellung kann man auch mit Hilfe der Translate()-Methode konvertieren.

Listing 39.2
Besitzerinformationen auslesen [Einsatzgebiete/Sicherheitseinstellungen/Filesystem_Owner.ps1]

```
"Besitzerinformationen:"
$a = Get-Acl g:\daten\kunden
$a.Owner
$a.GetOwner([System.Security.Principal.NTAccount]).Value
$a.GetOwner([System.Security.Principal.SecurityIdentifier]).Value

# Übersetzen zwischen Kontoname und SID
$konto = $a.GetOwner([System.Security.Principal.NTAccount])
$konto.Translate([system.security.principal.securityidentifier]).va
lue

# Übersetzen zwischen SID und Kontoname
$konto =
$a.GetOwner([System.Security.Principal.SecurityIdentifier])
$konto.Translate([system.security.principal.NTAccount]).value
```

39.7 Benutzer und SID

Translate()

Möchte man für einen beliebigen Benutzer dessen SID ermitteln, kann man auch eine Instanz von `System.Security.Principal.NtAccount` unter Angabe des Benutzernamens in Textform erzeugen und dann `Translate()` aufrufen.

Umwandeln zwischen Benutzername und SID

Das folgende Skript zeigt die Umwandlung eines Prinzipalnamens in einen SID und umgekehrt mit Hilfe der Methode `Translate()` in der Klasse `IdentityReference`, welche die Basisklasse für `NTAccount` und `SecurityIdentifier` ist.

Listing 39.3 *SID ermitteln [Einsatzgebiete/Sicherheitseinstellungen/SID.ps1]*

```
# Umwandlung zwischen Kontoname und SID

# Hole Konto
$Account = New-Object system.security.principal.ntaccount("itv\hs")
# Übersetze in SID
$SID =
$Account.Translate([system.security.principal.securityidentifier]).
value
$SID

# Übersetze in Kontonamen
$Account = New-Object
system.security.principal.securityidentifier("S-1-5-32-544")
$Name =
$Account.Translate([system.security.principal.ntaccount]).value
$Name
```

Well-Known Security Identifier verwenden

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. Im Active Directory sind die *Well-Known Security Principals* im *ConfigurationNamingContext* im 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`.

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
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 39.2 *SIDs der Well-Known Security Principals*

.NET 2.0 stellt eine Auflistung `System.Security.Principal.WellKnownSidType` bereit, die man zur Instanziierung der Klasse `SecurityIdentifier` einsetzen kann. Man umgeht damit die sprachspezifischen Unterschiede des Betriebssystems („Guests“/„Gäste“).

Listing 39.4 *Zugriff auf ein Konto über die SID [Einsatzgebiete/Sicherheitseinstellungen/Accounts_and_SID.ps1]*

```
# Bekannte Konten
$SID =
[System.Security.Principal.WellKnownSidType]::BuiltinAdministratorsSid
$Account = New-Object
system.security.principal.securityidentifier($SID, $null)
$Name =
$Account.Translate([system.security.principal.ntaccount]).value
$Name
```

Einige eingebaute Benutzer und Gruppen beinhalten den SID der Domäne in ihrem eigenen SID. In diesem Fall muss bei der Instanziierung der Klasse `SecurityIdentifier` der Domänen-SID mit angegeben werden. Leider schweigt sich die Dokumentation darüber aus, woher man den Domänen-SID mit .NET-Methoden bekommt. Auch im WWW findet man noch kein Beispiel dafür.

SDDL verwenden

SDDL

Eine andere Möglichkeit zum Zugriff auf eingebaute Benutzer und Gruppen besteht in der Verwendung der in der Security Descriptor Definition Language (SDDL) definierten Abkürzungen für die eingebauten Benutzer und Gruppen (siehe *Tabelle 39.3*).

Listing 39.5 *Ermitteln einer SID aus einem SDDL-Kürzel [Einsatzgebiete/Sicherheitseinstellungen/SID.ps1]*

```
# SDDL-Namen
$Account = New-Object
System.Security.Principal.SecurityIdentifier("BA")
$Account.Value
```

SDDL-Abkürzung	Bedeutung
"AO"	Account operators
"AN"	Anonymous logon
"AU"	Authenticated users
"BA"	Built-in administrators
"BG"	Built-in guests
"BO"	Backup operators
"BU"	Built-in users
"CA"	Certificate server administrators
"CG"	Creator group
"CO"	Creator owner
"DA"	Domain administrators
"DC"	Domain computers
"DD"	Domain controllers
"DG"	Domain guests
"DU"	Domain users
"EA"	Enterprise administrators
"ED"	Enterprise domain controllers
"WD"	Everyone
"PA"	Group Policy administrators
"IU"	Interactively logged-on user
"LA"	Local administrator
"LG"	Local guest
"LS"	Local service account
"SY"	Local system
"NU"	Network logon user
"NO"	Network configuration operators

Tabelle 39.3*: SDDL-Abkürzungen für eingebaute Benutzer und Gruppen*

SDDL-Abkürzung	Bedeutung
"NS"	Network service account
"PO"	Printer operators
"PS"	Personal self
"PU"	Power users
"RS"	RAS servers group
"RD"	Terminal server users
"RE"	Replicator
"RC"	Restricted code
"SA"	Schema administrators
"SO"	Server operators
"SU"	Service logon user

***Tabelle 39.3**: SDDL-Abkürzungen für eingebaute Benutzer und Gruppen (Forts.)*

39.8 Hinzufügen eines Rechteeintrags zu einer Zugriffsrechteliste

FileSystem-AccessRule

Das folgende Skript zeigt das Ergänzen eines Rechteeintrags zu einer Rechteliste einer Datei im Dateisystem. Neue Rechteobjekte vom Typ `FileSystemAccessRule` benötigen fünf Angaben:

- Kontoobjekt (`NTAccount`-Objekte oder `SecurityIdentifier`-Objekte)
- Zu vergebende Rechte (Werte aus der `FileSystemRights`-Aufzählung)
- Ziele der Vererbung (Werte aus der `InheritanceFlags`-Aufzählung)
- Art der Vererbung (Werte aus der `PropagationFlags`-Aufzählung)
- Art der Regel: Erlauben oder Verbieten (Werte aus der `AccessControlType`-Aufzählung)

Das folgende Skript gewährt einem Benutzer Leserechte auf ein Verzeichnis:

***Listing 39.6** ACE anfügen [Einsatzgebiete/Sicherheitseinstellungen/Filesystem_ACL_Write.ps1]*

```
# ACL schreiben: Lese- und Schreibrechte fuer einen Benutzer setzen

# Eingabedaten
$DIR = "g:\daten\kunden"
$BENUTZER = "HS"

# Hole ACL
$ACL = Get-Acl  $DIR
```

```
"ACL vorher:"
$acl | format-list

# ACE definieren
$Rights = [System.Security.AccessControl.FileSystemRights]
"ReadData, ReadExtendedAttributes, ReadAttributes, ReadPermissions"
$Access=[System.Security.AccessControl.AccessControlType]::Allow
$Inherit=[System.Security.AccessControl.InheritanceFlags]::Containe
rInherit `-bor
[System.Security.AccessControl.InheritanceFlags]::ObjectInherit
$Prop=[System.Security.AccessControl.PropagationFlags]::InheritOnly
$AccessRule = New-Object
System.Security.AccessControl.FileSystemAccessRule `
($BENUTZER,$Rights,$Inherit,$Prop,$Access)

# ACE an ACL anfügen
$ACL.AddAccessRule($AccessRule)

# ACL speichern
Set-Acl -AclObject $ACL -Path $DIR

# Kontrolle
$ACL = Get-Acl  $DIR
"ACL nachher:"
$acl | format-list
```

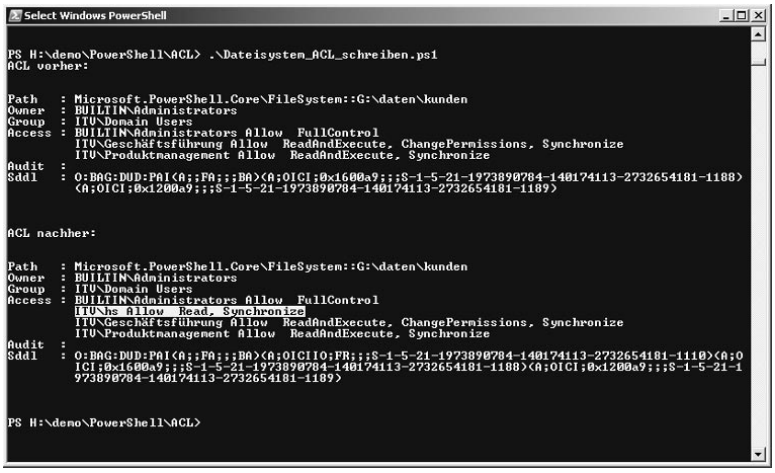


Abbildung 39.7 *Ausführung des Skripts, das einem Benutzer Leserechte gewährt*

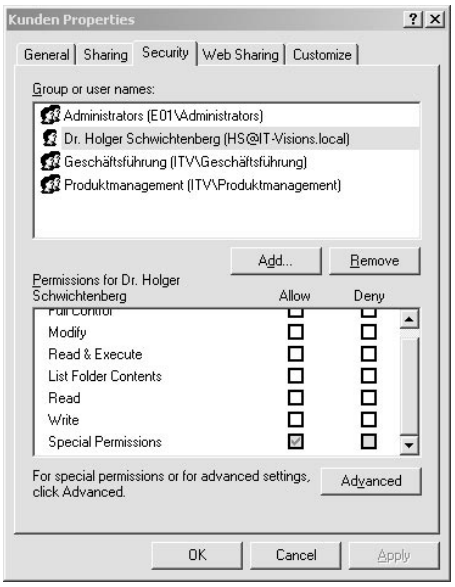


Abbildung 39.8
Ansicht der Rechte im Windows Explorer



Wenn in einem Parameter mehrere Flags zu setzen sind, sind diese mit einem binären Oder zu verknüpfen (Operator `-bor` in der PowerShell-Sprache).

```
$Rights= [System.Security.AccessControl.FileSystemRights]::Read `
-bor [System.Security.AccessControl.FileSystemRights]::
ReadExtendedAttributes `
-bor [System.Security.AccessControl.FileSystemRights]::
ReadAttributes `
-bor [System.Security.AccessControl.FileSystemRights]::
ReadPermissions
```

Prägnanter kann man die Aufzählungswerte auch in eine durch Kommata getrennte Zeichenkette schreiben.

```
$Rights = [System.Security.AccessControl.FileSystemRights]
"ReadData, ReadExtendedAttributes, ReadAttributes,
ReadPermissions"
```

39.9 Entfernen eines Rechteeintrags aus einer Zugriffsrechteliste

ACE

Zum Entfernen eines Rechteeintrags (Access Control Entry – ACE) aus der Zugriffsrechteliste dient die Methode `RemoveAccessRule()`, die von `NativeObjectSecurity` an alle Rechteklassen vererbt wird. Die Methode erwartet als Parameter ein Objekt vom Typ `AccessContolEntry`.

Möchte man alle Einträge zu einem Benutzer entfernen, kann man `PurgeAccessRules()` unter Angabe eines Benutzerkontoobjekts (nicht des Kontonamens!) verwenden.

Beispiel 1 Das folgende Skript löscht aus einer Zugriffsrechteliste alle Rechteeinträge zu einem bestimmten Benutzer.

Listing 39.7 ACL schreiben: Alle ACEs eines Benutzers löschen [Einsatzgebiete/Sicherheitseinstellungen/Filesystem_ACL_Delete.ps1]

```
# ACL schreiben: Alle ACEs eines Benutzers loeschen

# Eingabedaten
$DIR = "g:\daten\kunden"
$BENUTZER = "itv\HS"
$Count = 0

# Kontrollausgabe
$acl = Get-Acl  $DIR
"ACL vorher:"
$acl | format-list

# ACL holen
$acl = Get-Acl g:\daten\kunden

$Account = New-Object system.security.principal.ntaccount("itv\hs")
$acl.PurgeAccessRules($Account)

# ACL speichern
Set-acl -AclObject $ACL -Path $DIR

# Kontrollausgabe
$acl = Get-Acl  $DIR
"ACL nachher:"
$acl | format-list
```

Beispiel 2 Das folgende Skript löscht aus einer Zugriffsrechteliste alle Rechteeinträge, in denen das Lese- und Ausführungsrecht (`"ReadAndExecute"`) vergeben wurde.

Listing 39.8 Aus einer Zugriffsrechteliste alle Rechteeinträge löschen, in denen das Lese- und Ausführungsrecht vergeben wurde ("ReadAndExecute") [Einsatzgebiete/Sicherheitseinstellungen/Filesystem_ACL_ReadAllReadAndExecute.ps1]

```
# ACL schreiben: Aus einer Zugriffsrechteliste alle Rechteeinträge
löschen, in denen das Lese- und Ausführungsrecht ("ReadAndExecute")
steht
# Eingabedaten
$DIR = "g:\daten\kunden"
$BENUTZER = "itv\HS"
$Count = 0

# Kontrollausgabe
$acl = Get-Acl  $DIR
"ACL vorher:"
$acl | format-list

# Zugriff auf ACEs
```

```
$aces =$acl.GetAccessRules($true, $true,
[System.Security.Principal.NTAccount])

# Schleife über alle ACEs
foreach ($ace in $aces)
{
Write-host $ace.IdentityReference.ToString() "hat Zugang"
$ACE.FileSystemRights $ACE.AccessControlType "Vererbt?"
$ACE.IsInherited
# Selektives Löschen
if ($ace.FileSystemRights.ToString() -match "ReadAndExecute")
 {
  "...wird entfernt!"
  $Ergebnis = $acl.RemoveAccessRule($ace)
if ($Ergebnis) { echo "Wurde entfernt!"; $Count++ }
 }
}

# ACL speichern
Set-acl -AclObject $ACL -Path $DIR

echo ($Count.ToString() + " ACEs wurden entfernt!")

# Kontrollausgabe
$acl = Get-Acl  $DIR
"ACL nachher:"
$acl | format-list
```

39.10 Zugriffsrechteliste übertragen

Get-Acl, Set-Acl

Durch die Kombination von `Get-Acl` und `Set-Acl` kann man auf einfache Weise eine Zugriffsrechteliste von einem Dateisystemobjekt auf ein anderes übertragen.

```
# Übertragen einer ACL von einer Datei auf eine andere
Get-Acl g:\daten\kunden | Set-Acl g:\daten\lieferanten

# Übertragen einer ACL von einer Datei auf eine Menge von Dateien
$acl = Get-Acl g:\Daten\kunden
Get-ChildItem g:\Daten | foreach-Object { Set-acl $_.Fullname $acl;
"Übertragen auf $_" }
```

Listing 39.9 *Übertragen einer ACL zwischen zwei Dateien [Einsatzgebiete/Sicherheitseinstellungen/Filesystem_ACL_Transfer.ps1]*

39.11 Zugriffsrechteliste über SDDL setzen

Die Security Descriptor Definition Language (SDDL) ist ein Textformat zur Beschreibung von Access Control Lists (ACLs) mit einzelnen ACEs in Windows (eingeführt mit Windows 2000).

Ein Beispiel für eine SDDL-Zeichenkette ist:

O:BAG:DUD:PAI(A;;FA;;;BA)(A;OICI;0x1600a9;;;S-1-5-21-1973890784-140174113-2732654181-1188)(A;OICI;0x1200a9;;;S-1-5-21-1973890784-140174113-2732654181-1189)

Beispiel Das folgende Skript nutzt SDDL zur Übertragung einer Zugriffsrechteliste von einem Verzeichnis auf ein anderes. Zwischenzeitlich wird die Zugriffsrechteliste im Dateisystem gespeichert, so dass man das Auslesen und Setzen zeitlich entkoppeln könnte.

Listing 39.10 *Übertragen einer ACL zwischen zwei Dateien [Einsatzgebiete/Sicherheitseinstellungen/SDDL.ps1]*

```
# Übertragen einer ACL via SDDL

$QUELLE = "g:\daten\kunden"
$ZIEL = "g:\daten\lieferanten"

function replace-acl
{
Param (
  $sObject,
  $sSDDL
  )
  $acl = Get-Acl $sObject
  $acl.SetSecurityDescriptorSddlForm($sSDDL)

  Set-Acl -aclObject $acl $sObject
}

# SDDL lesen und in Textdatei speichern
(Get-Acl $QUELLE).SDDL > g:\Daten\acl.txt

# SDDL aus Textdatei lesen
$sddl = Get-Content g:\Daten\acl.txt
replace-acl $ZIEL $sddl

"Folgende Rechte wurden übertragen: " + $sddl
```

40 Active Directory

WPS 1.0

Die PowerShell Version 1.0 stellte keine Commandlets für den Zugriff auf die Windows-Benutzerdatenbank „SAM“, das Active Directory oder andere Verzeichnisdienste bereit. In der Beta-Phase gab es einen Active-Directory-Navigationsprovider, der wurde jedoch bis zur Version 1.0 entfernt.

PSCX

Ein solcher Provider zur Navigation im Active Directory ist derzeit im Rahmen der PowerShell Community Extensions [CODEPLEX01] verfügbar. Dort gibt es auch das Commandlet `Get-ADObject` zur Suche im Active Directory.

.NET-Klassen

Mit der PowerShell 1.0 (ohne PSCX) ist ein Zugriff auf Verzeichnisdienste nur mit klassischen Programmiertechniken möglich. Hier sind die .NET-Klassen aus dem Namensraum `System.DirectoryServices` der .NET-Klassenbibliothek und zum Teil auch die COM-Komponente Active Directory Service Interfaces (ADSI) zu verwenden. Wenige Funktionen stehen auch mit WMI zur Verfügung. Hier boten die Active-Directory-Erweiterungen von *www.IT-Visions.de* eine Verbesserung.

WPS 2.0

In Windows Server 2008 R2 und Windows 7 (mit RSAT) gibt es ein PowerShell 2.0-Modul für das Active Directory („ADPowerShell“). Die „Remote Server Administration Tools (RSAT)“ können hier heruntergeladen werden:

Remote Server Administration Tools for Windows 7

http://www.microsoft.com/downloads/details.aspx?familyid=7D2F6AD7-656B-4313-A005-4E344E43997D&displaylang=en



ADPowerShell ist die beste Möglichkeit, die Sie nutzen sollten, wenn Sie eines der von ADPowerShell unterstützten Betriebssysteme verwenden.



Fallbeispiel „FBI“

Dieses Kapitel verwendet als Fallbeispiel die Domänen „FBI.net“ (Windows Server 2003) und „FBI.org“ (Windows Server 2008).

System.DirectoryServices

Dieses Beispiel dreht sich komplett um das Active Directory für die Fernsehserie „Akte X“ (engl. X-Files). Die Domänen heißen mit dem NETBIOS-Namen „FBI“. Die Domänencontroller heißen „XFilesServer11“ und „XFilesServer2“. Die PCs sind mit „F171“ bis „F179“ benannt. Als Organisationseinheiten und Benutzer existieren bzw. werden im Rahmen dieses Kapitels angelegt:

- Organisationseinheit „Agents“ mit Benutzern wie „Fox Mulder“, „Dana Scully“, „John Doggett“ und „Monica Reyes“
- Organisationseinheit „Directors“ mit Benutzern wie „Walter Skinner“ und „Alvin Kersh“
- Organisationseinheit „Conspirators“ mit „Smoking Man“ und „Deep Throat“
- Organisationseinheit „Aliens“ mit zahlreichen Außerirdischen

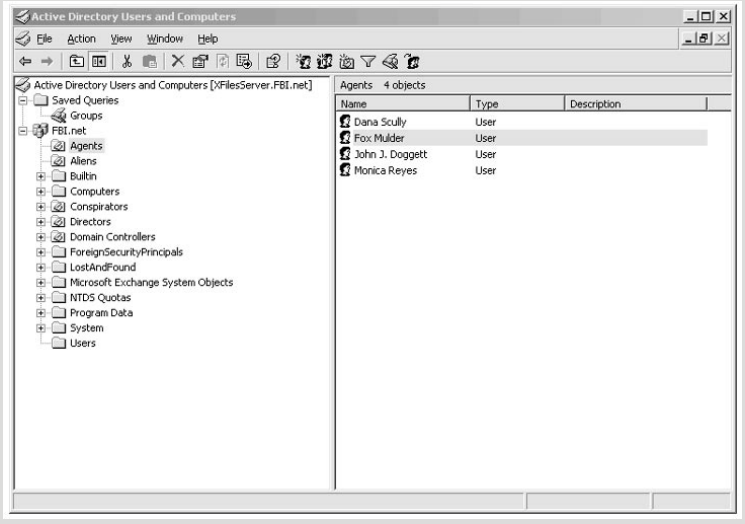


***Abbildung 40.1**: Das Active Directory für das „FBI“*

40.1 Benutzer- und Gruppenverwaltung mit WMI

Die Möglichkeiten der Benutzerverwaltung mit WMI sind leider beschränkt. ADSI bzw. `System.DirectoryServices` bieten hier wesentlich mehr.

Win32_Account

Der folgende Befehl liefert eine Objektliste der erreichbaren Benutzer und Gruppen:

```
Get-Wmiobject Win32_Account
```

Nur die Benutzerkonten erreicht man mit:

```
Get-Wmiobject Win32_UserAccount
```

Nur die Gruppen erreicht man mit:

```
Get-Wmiobject Win32_Group
```

Natürlich kann man damit gezielt Objekte herausfiltern:

```
# Name und Domäne der Benutzerkonten, deren Kennwort niemals
verfällt
Get-Wmiobject Win32_useraccount | Where-Object {$_.Kennwortexpires
-eq 0 } | Select-Object Name,Domain
```

Dies kann man alternativ auch so ausdrücken:

```
Get-Wmiobject Win32_Useraccount -filter "Kennwortexpires='false'" |
Select-Object Name,Domain
```

Win32_ Desktop

Die WMI-Klasse `Win32_Desktop` enthält Einstellungen der Benutzer. Mit dem folgenden Befehl bringt man in Erfahrung, ob der Benutzer „FBI\FoxMulder“ einen Bildschirmschoner auf dem Computer „AgentPC04“ aktiviert hat.

```
Get-Wmiobject Win32_Desktop -computer AgentPC04 | where { $_.Name -
eq "DBI\FoxMulder" } | select screensaveractive
```

40.2 Einführung in System.DirectoryServices

Die Klassen des .NET-Namensraums `System.DirectoryServices` sind eine Kapselung des Active Directory Service Interface (ADSI). Leider sind in der .NET-Bibliothek nicht alle Funktionen gekapselt, so dass auch ADSI in der PowerShell eine Rolle spielt.

Die Klassen im Namensraum `System.DirectoryServices` funktionieren nur, wenn auch die ADSI-COM-Komponente installiert ist.



Auf die ADSI-COM-Komponente wird in diesem Buch mit dem Begriff „klassisches ADSI“ Bezug genommen.

Architektur

Allgemeine Mechanismen

Die Klassen im .NET-Namensraum `System.DirectoryServices` bieten nur sehr allgemeine Mechanismen für den Zugriff auf Verzeichnisdienste. Es gibt keine spezifischen Klassen mehr für einzelne Verzeichnisdienste, wie sie in der ADSI-COM-Komponente vorhanden

sind. Bestimmte Operationen (z.B. Ändern des Kennworts in einem Benutzerobjekt) müssen daher direkt oder indirekt über die ADSI-COM-Komponente aufgerufen werden.

Zugriffsmöglichkeiten

Die folgende Grafik zeigt die Architektur von ADSI unter .NET. Ein .NET-Programm (Managed Code) hat drei Möglichkeiten, auf einen Verzeichnisdienst zuzugreifen:

1. Verwendung von Objekten im Namensraum `System.DirectoryServices` zur Ausführung von Verzeichnisdienstoperationen
2. Verwendung von Objekten im Namensraum `System.DirectoryServices` für den Aufruf von Operationen in der ADSI-COM-Komponente
3. Direkte Verwendung der ADSI-COM-Komponente via COM-Interoperabilität

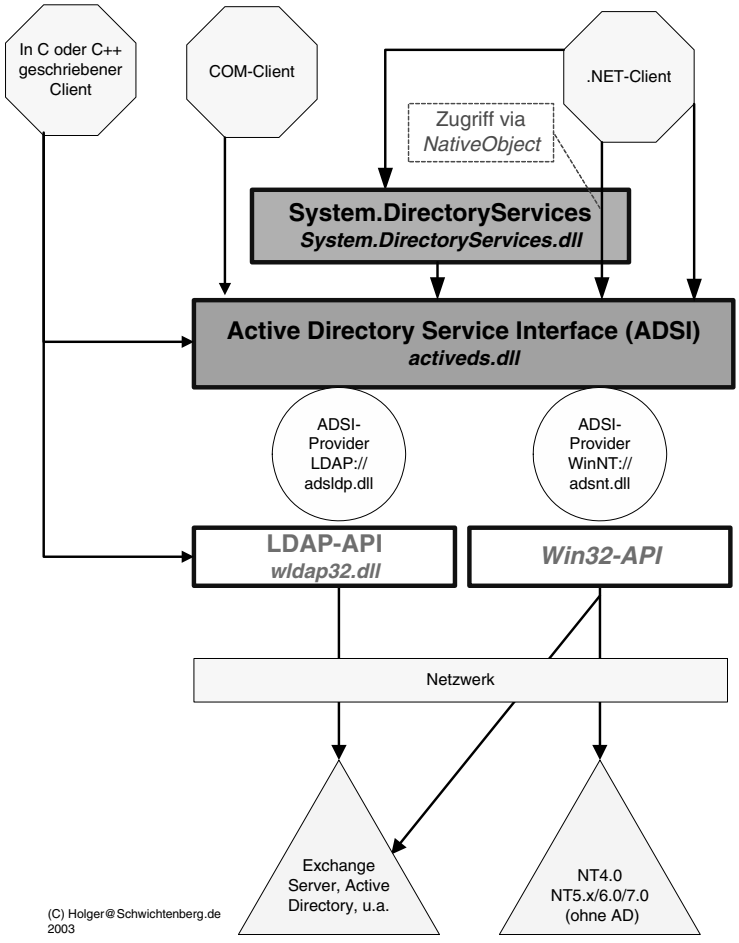


Abbildung 40.2 *Programmierschnittstellen für das Active Directory*

Weiterreichen an ADSI

Kapselung

Den Beweis dafür, dass alle Aufrufe in `System.DirectoryServices` in ADSI umgesetzt werden, liefern die Fehlermeldungen der .NET-Klassenbibliothek. Zum Beispiel liefert die Klasse `DirectoryEntry` beim Aufruf von `CommitChanges()` folgende Fehlermeldung, wenn das anzulegende Objekt bereits vorhanden ist:

```
System.Runtime.InteropServices.COMException (0x80071392):
Das Objekt ist bereits vorhanden.
at System.DirectoryServices.Interop.IAds.SetInfo()
at System.DirectoryServices.DirectoryEntry.CommitChanges()
```

Eigenschaftenzwischenspeicher

Dies bedeutet nichts anderes, als dass der Aufruf `CommitChanges()` in der Klasse `DirectoryEntry` intern weitergereicht wurde an die Methode `SetInfo()` in der Schnittstelle `System.DirectoryServices.Interop.IADs`. Dabei ist `SetInfo()` die aus der klassischen ADSI-COM-Komponente bekannte Methode, um den Eigenschaftenzwischenspeicher (engl. Property Cache) an den Verzeichnisdienst zurückzuliefern und damit alle Änderungen persistent zu machen.



Der Namensraum `System.DirectoryServices.Interop` ist undokumentiert und im Objektkatalog von Visual Studio nicht sichtbar. In diesem Namensraum sind die aus dem klassischen ADSI bekannten Schnittstellen `IADs`, `IADsContainer` etc. definiert. Da in .NET eine Instanziierung von Schnittstellen nicht mehr möglich ist, mussten die Schnittstellen zu Klassen zusammengefasst werden.

Objektmodell

Die Klassen im Namensraum `System.DirectoryServices` lassen sich in zwei Gruppen einteilen:

- Allgemeine Klassen für den Zugriff auf Blätter und Container
- Klassen für die Ausführung von LDAP-Suchanfragen

Allgemeine Klassen

Blätter und Container

Die beiden zentralen Klassen in diesem Namensraum sind `DirectoryEntry` und `DirectoryEntries`.

Klasse „DirectoryEntry“

Property Objektmenge

Die Klasse `DirectoryEntry` repräsentiert einen beliebigen Verzeichniseintrag, egal ob es sich um ein Blatt oder einen Container handelt. Diese Klasse besitzt ein Attribut `Children` vom Typ `DirectoryEntries`. Diese Objektmenge ist nur dann gefüllt, wenn das Objekt ein Container ist, also Unterobjekte besitzt. Die Objektmenge existiert aber auch in einem Blattobjekt; sie ist dann allerdings leer.

PropertyCollection

Die `DirectoryEntry`-Klasse besitzt im Attribut `Property` eine Objektmenge vom Typ `PropertyCollection`, welche die Menge der Verzeichnisattribute des Verzeichnisobjekts repräsentiert. Die `PropertyCollection` verfügt über drei untergeordnete Objektmengen:

- `PropertyNames` zeigt auf ein `KeysCollection`-Objekt, das Zeichenketten mit den Namen aller Verzeichnisattribute enthält.
- `Values` zeigt auf eine `ValuesCollection`, die wiederum einzelne Objektmengen vom Typ `PropertyValueCollection` enthält. Dies ist notwendig, da jedes Verzeichnisattribut mehrere Werte haben kann. Die `ValuesCollection` repräsentiert die Menge der Werte aller Verzeichnisattribute, die `PropertyValueCollection` steht für die einzelnen Werte eines Verzeichnisattributs.
- Das Attribut `Item(ATTRIBUTNAME)` liefert für einen als Parameter zu übergebenden Attributnamen die zugehörige `PropertyValueCollection`.



Der Zugriff über das Attribut `Values` kommt in der Regel nicht vor, da man normalerweise die Werte ohne die Namen der Attribute benötigt. Der normale Weg ist entweder die direkte Verwendung von `Item()`, wenn der Attributname bekannt ist, oder aber die Iteration über `PropertyNames` und darauffolgend die Verwendung von `Item()`, wenn alle Attribute mit ihren Werten aufgelistet werden sollen.

NativeObject

Jedes `DirectoryEntry`-Objekt besitzt ein Attribut mit Namen `NativeObject`, das einen Verweis auf das zugehörige ADSI-COM-Objekt liefert. Damit ist ein schneller Wechsel zur klassischen ADSI-Programmierung möglich.

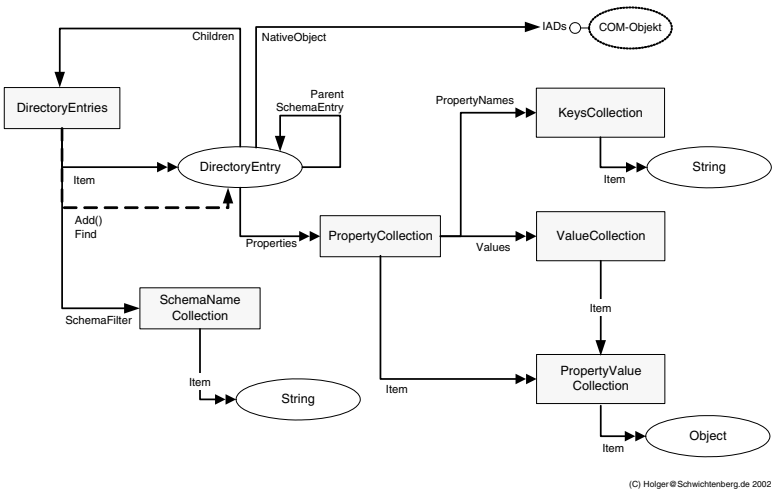


Abbildung 40.3 *Objektmodell der Klassen im Namensraum „System.DirectoryServices“ – Teil 1*

Klasse „DirectoryEntries“

Containerobjekte

Die Klasse `DirectoryEntries` unterstützt die Schnittstelle `IEnumerable` und ermöglicht daher die Auflistung ihrer Mitglieder über eine `foreach`-Schleife. Die Menge kann gefiltert werden, indem über die `SchemaNameCollection` eine Menge von Verzeichnisdienstklassen spezifiziert wird, die berücksichtigt werden sollen. Die Methode `Find()` liefert ein `DirectoryEntry`-Objekt. Wenn das anhand des Namens spezifizierte Objekt nicht in diesem Container vorhanden ist, gibt es eine *InvalidOperationException*.

Die Klasse `DirectoryEntries` kann nicht instanziiert werden. Sie erhalten ein `DirectoryEntries`-Objekt immer nur über das Attribut `Children` eines `DirectoryEntry`-Objekts.

Klassen für die Ausführung von Suchanfragen

LDAP-Suchanfragen

LDAP-Suchanfragen wurden in ADSI über die ActiveX Data Objects (ADO) bzw. einen OLEDB-Provider ausgeführt. In .NET gibt es nun eigene Klassen für die Ausführung von LDAP-Suchanfragen, die unabhängig von ADO.NET sind und direkt auf die LDAP-Implementierung von Windows zugreifen.

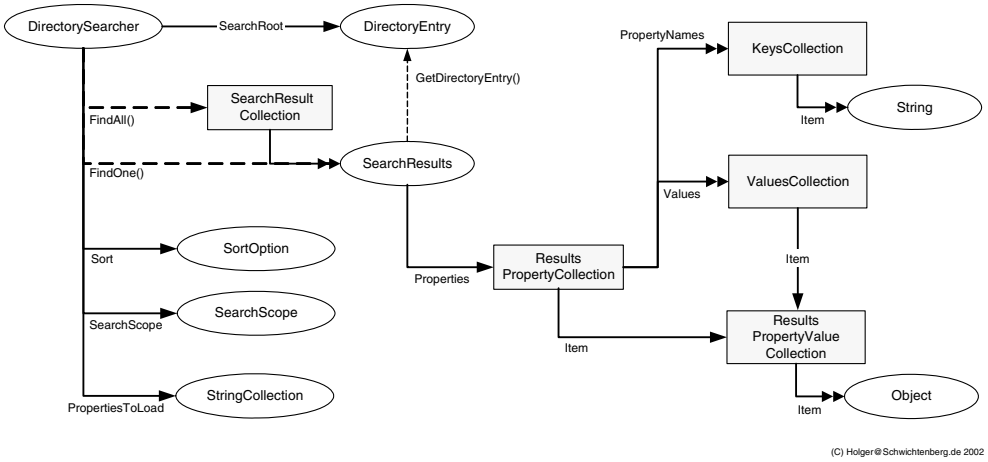


***Abbildung 40.4**: Objektmodell der Klassen im Namensraum „System.DirectoryServices“ – Teil 2*

Vergleich zwischen System.DirectoryServices und ADSI

Die folgende Tabelle zeigt, dass es für viele Schnittstellen aus der ADSI-COM-Komponente keine entsprechende spezifische Klasse in `System.DirectoryServices` mehr gibt.

Tabelle 40.1 System.DirectoryServices vs. ADSI

Verzeichniseintrag im Active Directory	ADSI in COM	ADSI in .NET (System.DirectoryServices)
Blatt	Schnittstelle `IADs`	Klasse `DirectoryEntry`
Container/Ast	Schnittstelle `IADsContainer`	Klasse `DirectoryEntries`
Klasse „`User`“	Schnittstelle `IADsUser`	`--- (DirectoryEntry)`
Klasse „`Computer`“	Schnittstelle `IADsComputer`	`--- (DirectoryEntry)`
Klasse „`Group`“	Schnittstelle `IADsGroup`	`--- (DirectoryEntry)`
LDAP-Suche	Klassen `ADODB.Connection` und `ADODB.RecordSet`	Klassen `DirectorySearcher` und `SearchResultCollection`

Unzulänglichkeiten der Implementierung

ADSI vs. System.DirectoryServices.DirectoryEntry

Leider hat Microsoft zwischen dem Release Candidate 1 und dem Release Candidate 2 der Windows PowerShell einen fundamentalen Richtungswechsel vollzogen, was die Unterstützung für Active Directory und andere Verzeichnisdienste angeht. Dieser Richtungswechsel kam nicht nur unerwartet, sondern führt nun auch in die falsche Richtung, weshalb an dieser Stelle massive Kritik an Microsoft angebracht ist.

Bis Release Candidate 1 der PowerShell 1.0 musste man für diese Scripting-Aufgaben direkt die .NET-Klassen aus dem .NET-Namensraum `System.DirectoryServices` verwenden. Diese Klassen basieren intern auf COM-Schnittstellen des Active Directory Services Interface (ADSI) und in einigen Fällen musste man für das Scripting auf ADSI „durchgreifen“.

Ab Release Candidate 2 der PowerShell 1.0 hat Microsoft eine Vereinfachung einführen wollen mit dem eigenen PowerShell-Typ `[ADSI]`. Die Absicht ist gut, die Umsetzung ist jedoch eine absolute Katastrophe:

1. [ADSI] instanziiert den Typ `System.DirectoryServices.DirectoryEntry`, bietet aber nur Attribute und keine Methoden dieser Klasse an. Die Methoden werden über das Extended Type System (ETS) versteckt.

2. Das erzeugte PowerShell-Objekt bietet stattdessen die Methoden der zu Grunde liegenden ADSI-COM-Klasse an.
3. Das Analyse-Commandlet `Get-Member` zeigt weder die einen noch die anderen Methoden an.
4. Auch bei direkter Instanziierung von `System.DirectoryServices.DirectoryEntry` gilt das obige Methodenchaos.
5. Die Methoden der Klasse `System.DirectoryServices.DirectoryEntry` stehen über das Unterobjekt `PSBase` zur Verfügung.
6. `DirectoryEntry`-Objekte können in der PowerShell-Pipeline nicht mit den üblichen Commandlets `Select-Object`, `Format-Table` etc. weiterverarbeitet werden. Möglich ist nur der objektbasierte Stil.

Konzeptionelles Chaos

Das ist eine sehr unlogische Implementierung. Schon im Windows Script Host (WSH) war das Verzeichnisdienst-Scripting nicht einfach, jetzt wird es noch schwieriger zu erlernen. Die folgende Abbildung dokumentiert noch einmal das konzeptionelle Chaos:

- Ein Verzeichnisdienstobjekt in einem Verzeichnisdienst besitzt eigentlich nur Attribute.
- Verzeichnisdienstoperationen werden durch das jeweilige Protokoll (z.B. LDAP) bereitgestellt. Ein ADSI-COM-Objekt kapselt diese Operationen für jeweils ein Verzeichnisobjekt in Methoden.
- Ein .NET-Objekt des Typs `DirectoryEntry` kapselt das ADSI-COM-Objekt, bietet dabei aber andere Methoden an (die intern wieder auf ADSI aufsetzen). Das Objekt `DirectoryEntry` eröffnet über das Unterobjekt `NativeObject` einen direkten Zugang zu den ADSI-Methoden.
- Das PowerShell-Objekt, das wiederum eine Kapsel um das `DirectoryEntry`-Objekt darstellt, verwendet nun aber nicht die Methoden von `DirectoryEntry`, sondern die Methoden des inneren ADSI-Objekts.
- Das PowerShell-Objekt bietet über das Unterobjekt `PSBase` einen Zugang zu den Methoden des `DirectoryEntry`-Objekts.

Aruk Kumaravel, Windows PowerShell Development Manager bei Microsoft, gibt in [Kumaravel01] zu, dass es unklug war, Methoden zu verstecken: „In retrospect, maybe we should have exposed these."

Außerdem ist übrigens noch sehr kritisch anzumerken, dass Microsoft eine solch fundamentale Änderung zwischen einem RC1 und einem RC2 vornimmt. Alle bis dahin geschriebenen PowerShell-Skripte für das Active Directory konnte man damit über den Haufen werfen. Eine solche Änderung erwartet man in der Beta-Phase, nicht aber kurz vor dem Erscheinen.

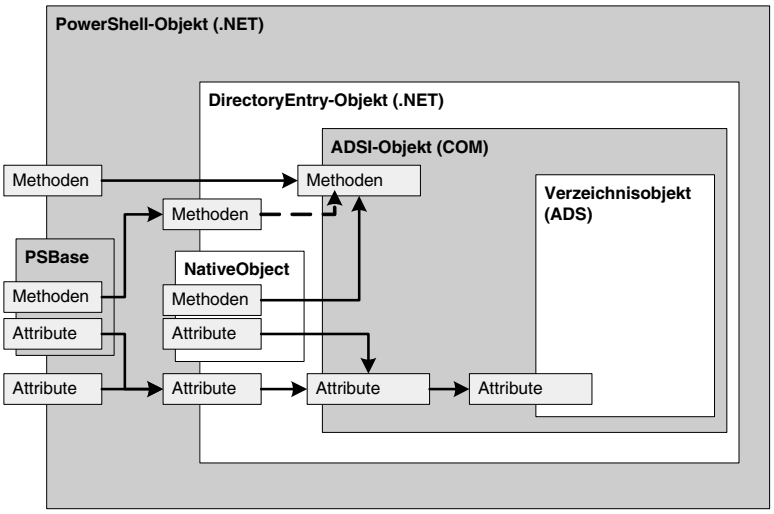


Abbildung 40.5 *Chaos bei den Verzeichnisdienstoperationen*

Objektidentifikation in Verzeichnisdiensten (Verzeichnisdienstpfade)

Zur Programmieren mit Verzeichnisdiensten ist es zwingend notwendig, die einzelnen Einträge im Verzeichnisdienst identifizieren zu können.

Verzeichnispfade

ADSI bedient sich für Pfadangaben auch unter .NET der sogenannten COM-Moniker (engl. für „Spitzname"), um einzelne Einträge in verschiedenen Verzeichnisdiensten anzusprechen und einen Zeiger auf das Stellvertreterobjekt zu erhalten. Der Moniker hat die Form

```
<Namenraum-ID>:<providerspezifischer Teil>
```

und wird in diesem Zusammenhang Verzeichnispfad (oder ADSI-Pfad) genannt.



Bei der Namensraum-ID werden Groß- und Kleinschreibung berücksichtigt, also muss man „LDAP" statt „ldap" oder „Ldap" verwenden!

DN und RDN

Der providerspezifische Teil des Verzeichnisdienstpfads enthält den Distinguished Name (DN) des Verzeichnisobjekts und dazu einen Servernamen (siehe folgende Tabelle).

Namensraum	Beispiele für Verzeichnispfade
Active Directory (via LDAP)	*LDAP://server/cn=Agents,dc=FBI,dc=NET* *LDAP://XFilesServer101.FBI.net/cn=Fox Mulder, OU=Agents,dc=FBI,dc=NET*
NT 4.0-Domänen und lokale Windows-Benutzerdatenbanken (SAM)	*WinNT://Domaene/Computer/Benutzer* *WinNT://Computername/Gruppenname* *WinNT://domaene/benutzer*
Novell 3.x	*NWCOMPAT://NWServer/Druckername*
Novell 4.x (NDS)	*NDS://Server/O=FBI/OU=Washington/cn=Agents*
Internet Information Services (IIS)	*IIS://ComputerName/w3svc/1*

***Tabelle 40.2**: Beispiele für ADSI-Pfade in verschiedenen Verzeichnisdiensten*

Objektidentifikation im Active Directory

Active Directory

Für die Adressierung der Einträge in einem Active Directory werden Verzeichnispfade der Form *LDAP://server:port/DN* verwendet. Dabei sind alle Bestandteile optional:

- Ohne Servername wird der sogenannte *Locator Service* verwendet. Beim serverlosen Binden sucht der Active Directory Locator Service mit Hilfe des Domain Name Service (DNS) den besten Domänencontroller 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.



Beim Active Directory sollten Sie immer den Namen des „nächstgelegenen" Domänencontrollers als Servernamen verwenden. Den Servernamen des Domänencontrollers ermitteln Sie über das Commandlet `Get-DomainController` (enthalten in den PSCX). Das Binden ohne Angabe eines Servers (serverloses Binden) ist möglich, aber aus Leistungsgesichtspunkten nicht empfehlenswert.

Bindung über GUIDs

Bei der Adressierung über einen Textpfad besteht die Gefahr, dass Verzeichnisobjekte zwischenzeitlich umbenannt wurden. Active Directory ermöglicht daher die Bindung über einen GUID, der für ein Verzeichnisobjekt unveränderlich ist. Der GUID muss natürlich für ein Objekt bekannt sein.

```
LDAP://XFilesServer1/<GUID=228D9A87C30211CF9AA400AA004A5691>
```

Well-Known Objects

Für die Standardcontainer in einem Active Directory gibt es eine besondere Unterstützung. Für diese sogenannten *Well-Known Objects* besteht ein vordefinierter GUID (Well-Known GUID), der in jedem Active Directory gleich ist.

```
LDAP://<WKGUID=a9d1ca15768811d1aded00c04fd8d5cd,dc=fbi,dc=net>
```

Bitte beachten Sie, dass hierbei die Ansprache über `WKGUID=` erfolgt und der dahinter angegebene GUID nicht der wirkliche GUID des Objekts ist. Auch die Standardcontainer erhalten bei der Installation eines Active Directorys einen individuellen GUID; der WKGUID ist ein allgemeingültiger Alias.

Tabelle 40.3 *Liste der Well-Known Objects*

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

Überblick über die Programmiermechanismen

Dieses Kapitel dokumentiert die wichtigsten Mechanismen der Verzeichnisdienstprogrammierung mit `System.DirectoryServices`.

Bindung an einen Verzeichniseintrag

Instanziierung von DirectoryEntry

Voraussetzung für den Zugriff auf Objekte des Verzeichnisdienstes ist die Bindung eines ADSI-Objekts an einen Verzeichniseintrag. Während unter dem klassischen ADSI der Bindungsvorgang über die Methode `GetObject()` stattfand, wird dies in `System.DirectoryServices` über einen Parameter bei der Instanziierung der Klasse `DirectoryEntry` erledigt.

```
$o = New-Object system.directoryservices.directoryEntry("LDAP://
XFilesServer1")
```

[ADSI]

Hierfür gibt es auch eine Kurzform über den integrierten PowerShell-Datentyp `[ADSI]`:

```
$o = [ADSI] "LDAP://XFilesServer1"
```

Nach dieser Operation enthält die Variable `$o` die Instanz der Klasse `DirectoryEntry`. Beim Zugriff auf `$o` erscheint an der Konsole der relative Pfad.



Abbildung 40.6
Zugriff auf einen Active-Directory-Eintrag

Ohne Angabe eines LDAP-Pfads wird bei der Instanziierung von `DirectoryEntry` eine Verbindung zum Standardnamenskontext (Default Naming Context) des Active Directory aufgebaut, zu dem der Computer gehört.

```
New-Object System.DirectoryServices.DirectoryEntry
```

Impersonifizierung

Wechsel des Benutzerkontextes

Im Standard meldet sich die Klasse `DirectoryEntry` unter dem Benutzerkonto beim Active Directory an, das den Befehl bzw. das Skript gestartet hat. Es ist aber möglich, durch Impersonifizierung einen anderen Benutzer für die Kommunikation mit dem Active Directory zu verwenden, wenn der startende Benutzer nicht genug Rechte besitzt.

Die Klasse `DirectoryEntry` verwendet den ADSI-Impersonifizierungsmodus durch Angabe eines Benutzernamens und eines Kennworts bei der Instanziierung der Klasse `DirectoryEntry` als zweiten und dritten Parameter. Bitte bedenken Sie, dass bei diesem Befehl das Kennwort im Klartext verwendet wird.

```
$o = New-Object system.directoryservices.directoryEntry("LDAP://
XFilesServer1/CN=Fox Mulder,OU=Agents,DC=FBI,DC=net", "FoxMulder",
"I+love+Scully")
```

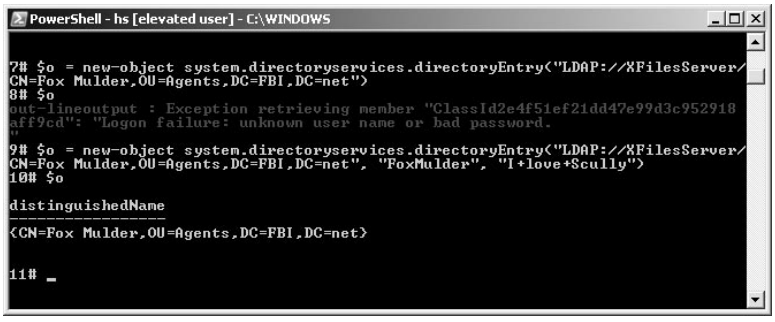


Abbildung 40.7
Zugriff ohne und mit Impersonifizierung

Prüfung auf Existenz eines Verzeichniseintrags

Exists()

Das klassische ADSI hatte keine eingebaute Methode, um die Existenz eines Verzeichnisobjekts zu überprüfen. Man war dort auf die (zeitaufwändige) Methode „Versuch und Irrtum" [WPE01] angewiesen. Unter .NET bietet die Klasse `DirectoryEntry` die statische Methode `Exists()` an, mit der sich prüfen lässt, ob ein anhand seines ADSI-Pfads spezifiziertes Verzeichnisobjekt existiert.

```
$janein = [system.directoryservices.directoryEntry]::Exists("LDAP://
/XFilesServer1/CN=Fox Mulder,OU=Agents,DC=FBI,DC=net")
```

Dies kann man abkürzen mit:

```
$janein = [ADSI]::Exists("LDAP://XFilesServer1/CN=Fox
Mulder,OU=Agents,DC=FBI,DC=net")
```

Verzeichnisattribute lesen

Attribute lesen über die Properties-Objektmenge

Eigentlich ist das Objektmodell von `System.DirectoryServices` sehr kompliziert: In einem `DirectoryEntry`-Objekt sind die einzelnen Werte nur verschachtelt über die Mengen Properties und PropertyValue-Objektmenge erreichbar. Die PowerShell kennt aber diesen Mechanismus und macht es dem Benutzer daher einfacher, er kann schreiben:

```
$xy = $obj.Attributname
```

Auch mehrwertige Attribute können so ausgelesen werden.

Im folgenden Beispiel werden Daten über einen Benutzer ausgelesen:

Listing 40.1 *Auslesen eines Verzeichnisobjekts [ADS_Einzelobjekte.ps1]*

```
New-Object system.directoryservices.directoryEntry("LDAP://
XFilesServer1/CN=Fox Mulder,OU=Agents,DC=FBI,DC=net")
"Name: "+ $o.sn
"Ort: " + $o.l
"Telefon: " +$o.Telephonenumber
"Weitere Rufnummern: " +$o.OtherTelephone
```



Der Zugriff auf ein Verzeichnisattribut, das es nicht gibt, führt nicht zum Fehler. Achten Sie also auf die genaue Schreibweise!



Zum Auslesen des Verzeichnispfads eines Verzeichniseintrags, auf den Sie bereits einen Verweis in der Form einer Variable besitzen, müssen Sie auf `.psbase.path` zugreifen, z.B. `$o.psbase.path`.

ADSI Property Cache

Zwischenspeicherung

Da ADSI-Objekte nur Stellvertreter für Verzeichniseinträge sind, werden die Attributwerte in einem Eigenschaftenzwischenspeicher verwaltet. Beim ersten Zugriff auf ein Attribut lädt ADSI alle Attributwerte in den Eigenschaftenzwischenspeicher. Schreibzugriffe sind durch Zuweisungen an die Attribute möglich.

GetInfo(), SetInfo()

Alle Schreibzugriffe müssen mit einem Aufruf der Methode `CommitChanges()` (`SetInfo()` unter klassischem ADSI) abgeschlossen werden. Erst dann wird der Eigenschaftenzwischenspeicher 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 Eigenschaftenzwischenspeicher gibt es eine Methode: `RefreshCache()` (entspricht `GetInfo()` unter klassischem ADSI). Das Programm sollte sie explizit aufrufen, wenn nicht sicher ist, ob die Werte im Eigenschaftenzwischenspeicher noch aktuell sind. Mit `RefreshCache()` können auch Änderungen verworfen werden, wenn zwischen den Änderungen und dem `RefreshCache()` kein `CommitChanges()` steht. Durch Angabe eines Arrays mit Attributnamen bei `RefreshCache(ARRAY_OF_STRING)` können vor einem ersten Attributzugriff gezielt einzelne Werte in den Eigenschaftenzwischenspeicher gelesen werden, um zur Verringerung der Netzwerklast die Übertragung aller Attribute zu vermeiden.

Abschalten des Property Cache

Im Gegensatz zu ADSI bietet `System.DirectoryServices` die Möglichkeit, den Eigenschaftenzwischenspeicher auszuschalten. Dazu ist nach der Instanziierung des `DirectoryEntry`-Objekts folgender Befehl notwendig:

```
$o.PSBase.UsePropertyCache = 0
```



Die Abschaltung des Eigenschaftenzwischenspeichers funktioniert nicht beim Anlegen von Verzeichnisobjekten von Verzeichnisklassen, die Pflichtattribute haben, da der Verzeichnisdienst den Eintrag erst erzeugt, wenn alle Pflichtattribute übergeben wurden.

Verzeichnisattribute schreiben

Attribute schreiben über die Properties-Objektmenge

Das Beschreiben eines Verzeichnisattributs ist ebenso einfach. Man weist dem betreffenden Verzeichnisattribut entweder einen einfachen Wert oder ein Array von Werten (bei einem mehrwertigen Attribut) zu.

Wichtig ist nur, dass am Ende der Eigenschaftenzwischenspeicher (Property Cache) auch geschrieben wird. Hier gibt es aufgrund des Methodenchaos nun zwei Alternativen:

1. Aufruf der COM-Methode `SetInfo()`
2. Aufruf der .NET-Methode `CommitChanges()` über das Unterobjekt `PSBase`

Die Methode heißt in der .NET-Welt nicht `SetInfo()`, sondern `CommitChanges()`.

Listing 40.2 *Ändern eines Verzeichnisobjekts [ADS_Einzelobjekte.ps1]*

```
$o.Telephonenumber = "+49 201 7490700"
$o.OtherTelephone = "+01 111 222222","+01 111 333333","+49 111
44444"
$o.SetInfo()
# oder:
$o.PSBase.CommitChanges()
```

40.3 Basiseigenschaften

Mitglieder der DirectoryEntry-Klasse

Die Metaklasse `DirectoryEntry` besitzt einige wenige Attribute, die Basiseigenschaften eines Verzeichnisdienstobjekts enthalten. Dies sind:

- `Name`: Relative Distinguished Name des Objekts
- `Path`: Distinguished Name des Objekts
- `SchemaClassName`: Name der Verzeichnisdienstklasse im Schema des Verzeichnisdienstes
- `Guid`: Global Unique Identifier (GUID) des Metaobjekts
- `NativeGuid`: der Global Unique Identifier (GUID) für das Verzeichnisdienstobjekt
- `Children`: Liste der untergeordneten Objekte
- `UsePropertyCache`: Flag, das anzeigt, ob der Eigenschaftenzwischenspeicher verwendet werden soll



In der aktuellen endgültigen Version der PowerShell kann man diese allgemeinen Attribute leider nicht direkt abrufen, sondern nur über `PSBase`.

```
$o = New-Object system.directoryservices.directoryEntry("LDAP://
XFilesServer1/CN=Fox Mulder,OU=Agents,DC=FBI,DC=net",
"FoxMulder", "I+love+Scully")
"Klasse: " + $o.PSBase.SchemaClassName
"GUID: " + $o.PSBase.Guid
```

Listing 40.3*: Zugriff auf Basiseigenschaften eines Verzeichnisobjekts [Einsatzgebiete\VerzeichnisdiensteADS_User_Misc.ps1]*

Zugriff auf Containerobjekte

Containerobjekte

Die Bindung an Containerobjekte und der Zugriff auf deren Verzeichnisattribute erfolgen vollkommen identisch zum Zugriff auf Blattobjekte, also über die Klasse `DirectoryEntry`. Sollen die Unterobjekte des Containers aufgelistet werden, muss jedoch das Unterobjekt

Children angesprochen werden, das ein DirectoryEntries-Objekt liefert. Das DirectoryEntries-Objekt enthält eine Instanz der Klasse DirectoryEntry für jeden untergeordneten Verzeichniseintrag.

Wieder ist zu beachten, dass das Unterobjekt Children nicht direkt, sondern nur über die PSBase zur Verfügung steht.

Listing 40.4 Liste der Unterobjekte eines Containers [ADS_Container_List.ps1]

```
$pfad= "LDAP://XFilesServer1/OU=Agents,DC=FBI,DC=net"
$con = New-Object system.directoryservices.directoryEntry($pfad)
$con.PSBase.Children
```

Eigentlich besitzt die DirectoryEntries-Menge keinen numerischen Index. Die PowerShell macht jedoch mit einem Trick den Zugriff auf die Elemente der Liste möglich.

```
"Das zweite Element ist " +
@($con.PSBase.Children)[1].distinguishedName
```

Alternativ kann man mit Find() auch ein Element in dem Container anhand seines CN suchen:

```
"Suche nach einem Element " + $con.PSBase.Children.find("cn=Dr.
Holger Schwichtenberg").distinguishedName
```

Verzeichnisobjekt anlegen

Add()

Ein Verzeichnisobjekt wird über den übergeordneten Container angelegt, weil nur dieser weiß, ob er eine bestimmte Verzeichnisklasse als Unterobjekt überhaupt zu akzeptieren bereit ist. Die Methode Add() der .NET-Klasse DirectoryEntries erwartet im ersten Parameter den Relative Distinguished Name (RDN) des neuen Objekts und im zweiten Parameter den Namen der Verzeichnisdienstklasse, die als Schablone für das Objekt verwendet werden soll. Nach dem Setzen eventuell vorhandener Pflichtattribute muss noch CommitChanges() aufgerufen werden.

Listing 40.5 Anlegen einer Organisationseinheit [ADS_OU_DeleteAndCreate.ps1]

```
"Anlegen einer OU..."
$pfad= "LDAP://XFilesServer1/DC=FBI,DC=net"
$con = New-Object system.directoryservices.directoryEntry($pfad)
$ou = $con.PSBase.Children.Add("ou=Directors","organizationalUnit")
$ou.PSBase.CommitChanges()
$ou.Description = "FBI Directors"
$ou.PSBase.CommitChanges()
"OU wurde angelegt!"
```

Verzeichnisobjekt löschen

Remove()

Ein Objekt wird entweder durch einen Methodenaufruf auf sich selbst (DeleteTree()) oder über die Ausführung von Remove() auf einem übergeordneten Containerobjekt gelöscht. Dabei ist als Parameter das DirectoryEntry-Objekt, welches das zu löschende Verzeich-

nisobjekt repräsentiert, anzugeben. Der Aufruf von `CommitChanges()` ist nicht nötig. `DeleteTree()` hat den Vorteil, dass es rekursiv auch alle Unterobjekte löscht.

Listing 40.6 *Löschen einer Organisationseinheit [ADS_OU_DeleteAndCreate.ps1]*

```
$oupfad= "LDAP://XFilesServer1/ou=Directors,DC=FBI,DC=net"
$ou = New-Object system.directoryservices.directoryEntry($oupfad)
if ([system.directoryservices.directoryEntry]::Exists($oupfad))

{
"OU existiert schon und wird jetzt erst gelöscht!"
$ou.PSBase.DeleteTree()
}
```

40.4 Benutzer- und Gruppenverwaltung im Active Directory

Dieses Kapitel liefert Ihnen einige Beispiele zur Verwendung der Klassen des Namensraums `System.DirectoryServices` zum Zugriff auf das Microsoft Active Directory.

Die Active-Directory-Verzeichnisklasse „user“

Ein Benutzerobjekt im Active Directory (AD-Klasse `"user"`) besitzt zahlreiche Verzeichnisattribute. Ein Pflichtattribut, das alle Benutzerobjekte besitzen, ist `SAMAccountName`, das den NT3.51/NT 4.0-kompatiblen Anmeldenamen enthält.

LDAP-Namen

Die folgende Tabelle zeigt weitere Verzeichnisattribute eines Benutzerobjekts im Active Directory. Es gibt sowohl unglaublich kurze Namen, wie z.B. „l“ für Stadt, als auch unglaublich lange Namen wie „physicalDeliveryOfficeName“ für das Büro.

Name	Pflicht	Mehrwertig	Datentyp (Länge)
`cn`	Ja	Nein	DirectoryString (1–64)
`nTSecurityDescriptor`	Ja	Nein	ObjectSecurityDescriptor (0–132096)
`objectCategory`	Ja	Nein	DN
`objectClass`	Ja	Ja	OID
`objectSid`	Ja	Nein	OctetString (0–28)
`sAMAccountName`	Ja	Nein	DirectoryString (0–256)
`accountExpires`	Nein	Nein	INTEGER8
`accountNameHistory`	Nein	Ja	DirectoryString

Tabelle 40.4*: Ausgewählte Attribute der Active-Directory-Klasse „user“*

Name	Pflicht	Mehrwertig	Datentyp (Länge)
badPwdCount	Nein	Nein	INTEGER
comment	Nein	Nein	DirectoryString
company	Nein	Nein	DirectoryString (1–64)
createTimeStamp	Nein	Nein	GeneralizedTime
department	Nein	Nein	DirectoryString (1–64)
description	Nein	Ja	DirectoryString (0–1024)
desktopProfile	Nein	Nein	DirectoryString
displayName	Nein	Nein	DirectoryString (0–256)
displayNamePrintable	Nein	Nein	PrintableString (1–256)
distinguishedName	Nein	Nein	DN
division	Nein	Nein	DirectoryString (0–256)
employeeID	Nein	Nein	DirectoryString (0–16)
employeeType	Nein	Nein	DirectoryString (1–256)
expirationTime	Nein	Nein	UTCTime
dacsimileTelephoneNumber	Nein	Nein	DirectoryString (1–64)
givenName	Nein	Nein	DirectoryString (1–64)
homeDirectory	Nein	Nein	DirectoryString
HomeDrive	Nein	Nein	DirectoryString
homeMDB	Nein	Nein	DN
initials	Nein	Nein	DirectoryString (1–6)
internationalISDNNumber	Nein	Ja	NumericString (1–16)
l	Nein	Nein	DirectoryString (1–128)
lastLogoff	Nein	Nein	INTEGER8
LastLogon	Nein	Nein	INTEGER8
logonCount	Nein	Nein	INTEGER
LogonHours	Nein	Nein	OctetString
logonWorkstation	Nein	Nein	OctetString
manager	Nein	Nein	DN
middleName	Nein	Nein	DirectoryString (0–64)
Mobile	Nein	Nein	DirectoryString (1–64)
name	Nein	Nein	DirectoryString (1–255)
objectGUID	Nein	Nein	OctetString (16–16)
objectVersion	Nein	Nein	INTEGER
otherFacsimileTelephoneNumber	Nein	Ja	DirectoryString (1–64)

***Tabelle 40.4**: Ausgewählte Attribute der Active-Directory-Klasse „user“ (Forts.)*

Name	Pflicht	Mehrwertig	Datentyp (Länge)
OtherHomePhone	Nein	Ja	DirectoryString (1–64)
physicalDeliveryOfficeName	Nein	Nein	DirectoryString (1–128)
PostalAddress	Nein	Ja	DirectoryString (1–4096)
postalCode	Nein	Nein	DirectoryString (1–40)
postOfficeBox	Nein	Ja	DirectoryString (1–40)
profilePath	Nein	Nein	DirectoryString
sAMAccountType	Nein	Nein	INTEGER
scriptPath	Nein	Nein	DirectoryString
street	Nein	Nein	DirectoryString (1–1024)
streetAddress	Nein	Nein	DirectoryString (1–1024)
TelephoneNumber	Nein	Nein	DirectoryString (1–64)
title	Nein	Nein	DirectoryString (1–64)
userWorkstations	Nein	Nein	DirectoryString (0–1024)
whenChanged	Nein	Nein	GeneralizedTime
whenCreated	Nein	Nein	GeneralizedTime
wWWHomePage	Nein	Nein	DirectoryString (1–2048)

***Tabelle 40.4**: Ausgewählte Attribute der Active-Directory-Klasse „user“ (Forts.)*

Einige mehrwertige Eingabefelder aus den Dialogen des MMC-Snap-In „Active Directory-Benutzer und -Computer“ werden im Active Directory in mehr als einem Attribut gespeichert. Ein gutes Beispiel dafür ist die Liste der Telefonnummern. Die Haupttelefonnummer ist in dem einwertigen Attribut `telephoneNumber` gespeichert, während die weiteren Telefonnummern in dem mehrwertigen Attribut `otherTelephone` stehen. Andere Fälle dieser Art sind:

- `mobile/otherMobile`,
- `mail/otherMailbox` und
- `logonWorkstation/otherLoginWorkstations`.



Übrigens handelt es sich bei den beiden letztgenannten Attributen nicht um Tippfehler des Buchautors (Login – Logon), sondern um Inkonsistenzen im Active Directory, für die man die Verantwortlichen in Redmond suchen muss.



Eine komplette Liste aller Verzeichnisattribute findet man in der Dokumentation des Active-Directory-Schemas [MSDN09]. Dabei sind im Skript die LDAP-Namen der Eigenschaften zu verwenden, die in der Dokumentation als „LDAP-Display-Name“ eingetragen sind (siehe *Abbildung 40.8*).

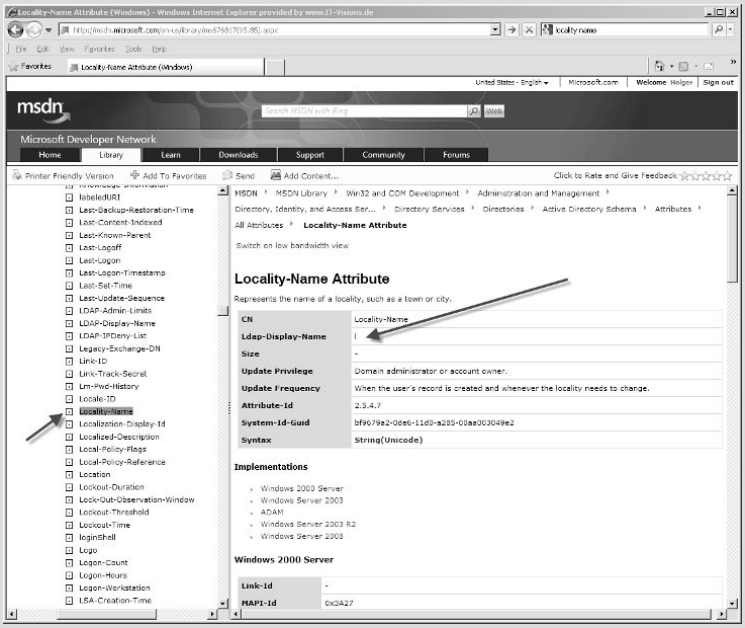


***Abbildung 40.8**: Dokumentation des Active-Directory-Schemas*

Der LDAP-Eigenschaftsname ist leider zum Teil sehr weit entfernt von den Namen in der MMC-Konsole. Das Dokument „User Object User Interface Mapping“ [MSDN10] hilft beim Auffinden der richtigen LDAP-Namen. Eine andere Möglichkeit ist, mit dem Werkzeug „ADSI Edit“ aus den „Support Tools“ für Windows Server direkt auf das „rohe“ Verzeichnis zu blicken und dort die LDAP-Namen herauszusuchen.

Benutzerkonto anlegen

„user“-Verzeichnisobjekt anlegen

Da das Anlegen eines Objekts vom übergeordneten Container ausgeht, muss im ersten Schritt der Container an `DirectoryEntry` gebunden werden. Die Erzeugung eines neuen Objekts erfolgt mit `Add()`, wobei im ersten Parameter der RDN des neuen Objekts und im zweiten Parameter der AD-Klassenname `user` anzugeben sind.

CommitChanges()

Das Setzen der Eigenschaft `SAMAccountName` ist Pflicht. Sofern der Eigenschaftenzwischenspeicher nicht ausgeschaltet wurde, muss nach dem Setzen aller Eigenschaften `CommitChanges()` ausgeführt werden, da sonst das Benutzerobjekt nicht angelegt wird.

Im Standard ist ein neues Benutzerkonto im Active Directory deaktiviert. Die einfachste Möglichkeit zur Aktivierung ist der Zugriff auf das Attribut `AccountDisabled` in der COM-Schnittstelle `IADsUSer`.

Beispiel

In der folgenden Routine wird ein Benutzerkonto „Walter Skinner" mit dem Anmeldenamen „WalterSkinner" angelegt. Als optionales Attribut werden nur die Stadt (`l`) und die Beschreibung (`Description`) gesetzt.

Listing 40.7 Anlegen eines User-Objekts im Active Directory [ADS_User_Create.ps1]

```
# ADS-Benutzer anlegen
$pfad= "LDAP://XFilesServer1/OU=Directors,DC=FBI,DC=net"
$name = "Walter Skinner"
$NTname = "WalterSkinner"
$ou = New-Object DirectoryServices.DirectoryEntry($pfad)
$user = $ou.PSBase.Children.Add("CN=" + $name,'user')
$user.PSBase.CommitChanges()
$user.SAMAccountName = $NTname
$user.l = "Washington"
$user.Description = "FBI Director"
$user.PSBase.CommitChanges()
"Benutzer wurde angelegt: " + $user.PBase.Path
```

Kennwort des Benutzers setzen

Kennwort festlegen mit SetKennwort()

Das Kennwort eines Benutzerkontos kann erst gesetzt werden, nachdem das Benutzerkonto im Verzeichnisdienst angelegt wurde. Auch bei dieser Operation ist unter .NET die Impersonifizierung notwendig. Hier kommt dem Nutzer nun einmal zugute, dass die PowerShell die ADSI-Methoden und nicht die COM-Methoden veröffentlicht, denn die Methode zum Setzen des Kennworts (`SetPassword()`) gibt es nicht in der .NET-Ebene. Als Parameter ist das neue Kennwort in Form einer Zeichenkette zu übergeben. Erst nach der Vergabe eines Kennworts sollte man den Benutzer aktivieren.

Listing 40.8 Kennwort für ein AD-Benutzerkonto setzen [ADS_User_Create.ps1]

```
$user.SetKennwort("secret-123")
"Kennwort wurde gesetzt"
$user.Accountdisabled = $false
$user.PSBase.CommitChanges()
```

Benutzerauthentifizierung

Benutzername und Kennwort überprüfen

Leider gibt es keine eingebaute Methode, die eine Authentifizierung mit Benutzername und Kennwort gegen das Active Directory ermöglicht. Um dies zu realisieren, bleibt nur die Versuch-und-Irrtum-Methode [WPE01]: Man versucht einen Zugriff auf das Active Direc-

tory unter Anwendung der Impersonifizierung mit den zu prüfenden Anmeldedaten. Ist ein Zugriff auf das Attribut `NativeGuid` möglich, dann stimmen die Daten. Wenn die Daten nicht stimmen, erhält man eine Fehlermeldung. Dies ist in der nachfolgenden Hilfsroutine `Authenticate-User()` realisiert.

Listing 40.9 *Authentifizierung beim ADS [ADS_Authentication.ps1]*

```
Function Authenticate-User {

trap [System.Exception] { "Fehler!"; return $false; }
"Versuche, Benutzer " + $args[1] + " mit dem Kennwort " + $args[2]
+ " zu authentifizieren bei " + $args[0] + "..."
$o = New-Object
system.directoryservices.directoryEntry([string]$args[0],
[String]$args[1], [String]$args[2])

$o.PSBase.NativeGUID
return $true
}

$o = New-Object system.directoryservices.directoryEntry("LDAP://
E02")
$o.get_NativeGUID()
$e = Authenticate-User "LDAP://XFilesServer1" "fbi\foxmulder" "I+
love+Scully"
$e
if ($e) { "Benutzer konnte authentifiziert werden!" }
else { "Benutzer konnte NICHT authentifiziert werden!" }
```

Benutzerkonto löschen

Um einen Benutzer zu löschen, kann man wieder `DeleteTree()` einsetzen, auch wenn ein Benutzer ein Blattobjekt ist, d.h. keine Unterobjekte besitzt.

Listing 40.10 *Löschen eines Benutzers [ADS_User_Create.ps1]*

```
$pfad= "LDAP://XFilesServer1/CN=Fox Mulder,OU=Agents,DC=FBI,DC=net"
$benutzer = New-Object
system.directoryservices.directoryEntry($pfad)
if ([system.directoryservices.directoryEntry]::Exists($pfad))
{
"Benutzer existiert schon und wird jetzt erst gelöscht!"
$benutzer.PSBase.DeleteTree()
}
```

Benutzerkonto umbenennen

Rename()

Für das Umbenennen eines Verzeichnisdienstobjekts bietet die Klasse `DirectoryEntry` mit der Methode `Rename()` ein sehr einfaches Verfahren. Unter klassischem ADSI musste dazu die `IADsContainer`-Methode `MoveHere()` verwendet werden.

Beispiel Im folgenden Beispiel wird das Benutzerkonto „Dana Scully“ in „Dana Mulder“ umbenannt (auch wenn es bisher weder in der Serie noch in den Filmen zu dieser Hochzeit kam).

Listing 40.11 *Umbenennen eines AD-Benutzerkontos [ADS_User_Misc.ps1]*

```
# Benutzer umbenennen
$pfad= "LDAP://XFilesServer1/CN=Dana
Scully,OU=Directors,DC=FBI,DC=net"
$user = New-Object system.directoryservices.directoryEntry($pfad)
$user.PSBase.Rename("cn=Dana Mulder")
"Benutzer wurde umbenannt!"
```

Benutzerkonto verschieben

MoveTo() Als Äquivalent zur COM-Methode `IADSContainer.MoveHere()` gibt es in der FCL-Klasse `DirectoryEntry` die Methode `MoveTo()`. Sie verschiebt ein Verzeichnisobjekt in einen anderen Container. Der Zielcontainer ist in Form eines zweiten `DirectoryEntry`-Objekts als Parameter zu übergeben.

Beispiel In der folgenden Routine wird der Benutzer „Fox Mulder“ aus der Organisationseinheit „Agents“ in den Standardbenutzercontainer „Users“ verschoben (diese Degradierung kam in der Serie tatsächlich vor).

Listing 40.12 *Verschieben eines AD-Benutzerkontos [ADS_User_Misc.ps1]*

```
# Benutzer verschieben
$pfad= "LDAP://XFilesServer1/CN=Walter Fox
Mulder,OU=Agents,DC=FBI,DC=net"
$ziel = "LDAP://XFilesServer1/CN=Users,DC=FBI,DC=net "
$user = New-Object system.directoryservices.directoryEntry($pfad)
$user.PSBase.MoveTo($ziel)
"Objekt verschoben!"
```

Gruppenverwaltung

Member In einem Verzeichnisobjekt des Typs `group` existiert ein Attribut `Member` mit den LDAP-Pfaden zu den Gruppenmitgliedern. Zum Anzeigen der Mitglieder einer Gruppe braucht man daher nur einen Einzeiler. Der folgende Befehl zeigt die Mitglieder der Gruppe aller FBI-Agenten:

```
(New-Object  directoryservices.directoryentry
("LDAP://XFilesServer1/CN=All Agents,DC=FBI,DC=net")).member
```

Dieser Befehl liefert aber nur die direkten Mitglieder. Wenn eine Gruppe aber eine andere Gruppe enthält, dann gibt es auch indirekte Mitglieder. Die im folgenden Listing implementierte Funktion `Get-Members` liefert rekursiv alle direkten und indirekten Mitglieder einer Gruppe im Active Directory.

Listing 40.13
Auflisten indirekter Gruppenmitglieder [ADS_Group_Create.ps1]

```
"Direct Group Members:"
$gruppe = New-Object directoryservices.directoryentry("LDAP://
xfilesserver/CN=All FBI Employees,DC=FBI,DC=net")
$gruppe.member

function Get-Members ($group){
  if ($group.objectclass[1] -eq 'group') {
   "-- Gruppe $($group.cn)"
    $Group.member | foreach-Object {
      $de = New-Object directoryservices.directoryentry("LDAP://
xfilesserver/" + $_)

      if ($de.objectclass[1] -eq 'group') {
        Get-Members $de
      }
      Else {
        $de.distinguishedName
      }
    }
  }
  Else {
    Throw "$group is not a group."
  }
}
""
"All Members (including non-direct):"
Get-Members(New-Object directoryservices.directoryentry("LDAP://
xfilesserver/CN=All FBI Employees,DC=FBI,DC=net"))
```

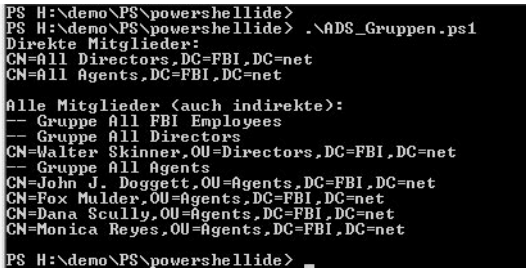


Abbildung 40.9
Auflisten direkter und indirekter Gruppenmitglieder

Anlegen und Befüllen einer Gruppe

Das Anlegen einer Gruppe erfolgt analog zum Anlegen eines Benutzers.

Add()

Beachten Sie beim Anlegen von Gruppen im Vergleich zum Anlegen von Benutzern den anderen Klassennamen (`group`).

Listing 40.14
Gruppe anlegen [ADS_Gruppe-Anlegen.ps1]

```
"Anlegen einer Gruppe..."
$pfad= "LDAP://XFilesServer1/DC=FBI,DC=net"
$con = New-Object system.directoryservices.directoryEntry($pfad)
$ou = $con.PSBase.Children.Add("cn=All Directors","group")
$ou.PSBase.CommitChanges()
```

```
$ou.samaccountname = "AllDirectors"
$ou.Description = "Group for FBI Directors"
$ou.PSBase.CommitChanges()
"Gruppe wurde angelegt!"
```

Für die Zuordnung von Benutzern zu Gruppen gibt es in der Klasse `DirectoryEntry` keine spezifischen Methoden. Hier ermöglicht das PowerShell-Objekt wieder den Zugang zu den in der COM-Schnittstelle `IADsGroup` definierten Methoden `Add()` und `Remove()`.

Listing 40.15 *Hinzufügen von Benutzern in Gruppen [ADS_Group_Members.ps1]*

```
# Hinzufuegen eines Gruppenmitglieds
$pfad= "LDAP://XFilesServer1/cn=All Directors,DC=FBI,DC=net"
$gr = New-Object system.directoryservices.directoryEntry($pfad)
$Benutzer = "LDAP://XFilesServer1/CN=Walter
Skinner,OU=Directors,DC=FBI,DC=net"
$gr.Add($Benutzer)
"Benutzer " + $Benutzer + " wurde der Gruppe " + $gr + "
hinzugefuegt
```

Listing 40.16 *Entfernen von Benutzern aus Gruppen [ADS_Group_Members.ps1]*

```
# Entfernen eines Gruppenmitglieds
$pfad= "LDAP://XFilesServer1/cn=All Directors,DC=FBI,DC=net"
$gr = New-Object system.directoryservices.directoryEntry($pfad)
$Benutzer = "LDAP://XFilesServer1/CN=Walter
Skinner,OU=Directors,DC=FBI,DC=net"
$gr.Remove($Benutzer)
"Benutzer " + $Benutzer + " wurde aus der Gruppe " + $gr + "
entfernt!"
```

Testen der Gruppenmitgliedschaft

Die Prüfung, ob ein Benutzer Mitglied einer Gruppe ist, erfolgt am einfachsten mit dem Commandlet `Test-UserGroupMembership` aus den PowerShell Community Extensions.

```
Test-UserGroupMembership -Identity hs -GroupName Administrators
```

40.5 Verwaltung der Organisationseinheiten

organizational-Unit

Das Erstellen und Löschen von Organisationseinheiten (Verzeichnisdienstklasse `organizationalUnit`) wurde bereits innerhalb des Überblicks über die Programmiertechniken gezeigt.

Beachten Sie beim Anlegen von Organisationseinheiten im Vergleich zum Anlegen von Benutzern den anderen Klassennamen (`organizationalUnit`) im ersten Parameter und den anderen Attributnamen (`OU`) im ersten Parameter bei `Add()`.

Listing 40.17
Skript zum Neuanlegen einer OU [ADS_OU_DeleteAndCreate.ps1]

```
# Skript zum Neuanlegen einer OU (Die OU wird geloescht, wenn sie
schon existiert!)

$oupfad= "LDAP://XFilesServer1/ou=Directors,DC=FBI,DC=net"
$ou = New-Object system.directoryservices.directoryEntry($oupfad)
if ([system.directoryservices.directoryEntry]::Exists($oupfad))
{
"OU existiert schon und wird jetzt erst gelöscht!"
$ou.PSBase.DeleteTree()
}

"Anlegen einer OU..."
$pfad= "LDAP://XFilesServer1/DC=FBI,DC=net"
$con = New-Object system.directoryservices.directoryEntry($pfad)
$ou = $con.PSBase.Children.Add("ou=Directors","organizationalUnit")
$ou.PSBase.CommitChanges()
$ou.Description = "FBI Directors"
$ou.PSBase.CommitChanges()
"OU wurde angelegt!"
```

40.6 Suche im Active Directory

Suche via LDAP

Im Active Directory können – wie in anderen LDAP-basierten Verzeichnisdiensten auch – Einträge, die bestimmten Kriterien entsprechen, containerübergreifend gesucht werden.

LDAP-Suchanfragen

Für LDAP-Suchanfragen existiert eine spezielle Syntax nach [RFC1960] und [RFC2254]. Dabei sind anzugeben:

- *Wurzel* ein LDAP-Pfad inkl. *LDAP://*. Der Pfad kann sowohl in Little-Endian- als auch in Big-Endian-Form angegeben werden.

  Beispiel: `LDAP://XFilesServer101/dc=FBI,dc=net`
- *Filter* eine Bedingung in umgekehrt polnischer Notation (UPN oder Postfix-Notation). Diese Notation zeichnet sich dadurch aus, dass die Operatoren am Anfang stehen. Erlaubte Operationen sind `&` (und), `|` (oder) und `!` (nicht). Zum Vergleich stehen =, <= und >= zur Verfügung, nicht aber < und >.

  Beispiel: `(&(objectclass=user)(name=h*))`
- *Attribute* eine 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`
- *Geltungsbereich* eine der in der folgenden Tabelle genannten Konstanten.

Tabelle 40.5 Suchtiefen bei LDAP-Suchabfragen

Konstante (LDAP-Syntax)	Erläuterung
BASE	Es wird nur auf der Ebene des angegebenen Eintrags gesucht. Die Ergebnismenge umfasst keinen oder einen Datensatz.
ONELEVEL	Es wird in den Einträgen gesucht, die dem angegebenen Eintrag untergeordnet sind.
SUBTREE	Es werden alle darunterliegenden Ebenen durchsucht.

EXKURS

In dem Active-Directory-MMC-Snap-In „Benutzer und Computer" gibt es ab Windows Server 2003 einen neuen Ast „Gespeicherte Abfragen", mit dem LDAP-Abfragen entworfen und ausgeführt werden können.

Programmierschnittstellen für die Suche

Im klassischen ADSI wurde die Suchfunktionalität durch einen OLEDB-Provider gekapselt. Dieser steht grundsätzlich auch in ADO.NET über den Managed Provider für OLEDB noch zur Verfügung. Allerdings bietet der Namensraum `System.DirectoryServices` eine elegantere Möglichkeit zur Ausführung von LDAP-Suchanfragen.

Während der OLEDB-Provider für ADSI-Anfragen sowohl LDAP-Query-Syntax als auch SQL-Befehle unterstützt, können mit den in der .NET-Klassenbibliothek eingebauten Klassen nur LDAP-Query-Syntaxanfragen gestellt werden.

Ebenso wie mit dem OLEDB-Provider lassen sich auch mit den FCL-Klassen nur LDAP-fähige Verzeichnisdienste abfragen. Die LDAP-Query-Syntax ist ein Standard ([RFC1960] und [RFC2254]) und daher nicht anders als bei der COM-Implementierung.

Ausführung einer Abfrage in der PowerShell

Suche definieren

Eine LDAP-Abfrage wird mit .NET-Klassen in folgenden Schritten ausgeführt:

- Instanziierung der Klasse `DirectorySearcher`
- Festlegung des Ausgangspunkts der Anfrage durch Zuweisung eines Zeigers auf ein `DirectoryEntry`-Objekt, das an den Ausgangspunkt gebunden ist, an das Attribut `SearchRoot`
- Setzen des Filterteils der LDAP-Abfrage im Attribut `Filter`

- Festlegung der Attribute durch Füllen der Objektmenge `PropertiesToLoad`
- Festlegung des Geltungsbereichs in dem Attribut `SearchScope`
- Starten der Anfrage durch die Methode `FindAll()`
- `FindAll()` liefert eine Objektmenge vom Typ `SearchResultCollection` zurück.

Suche starten

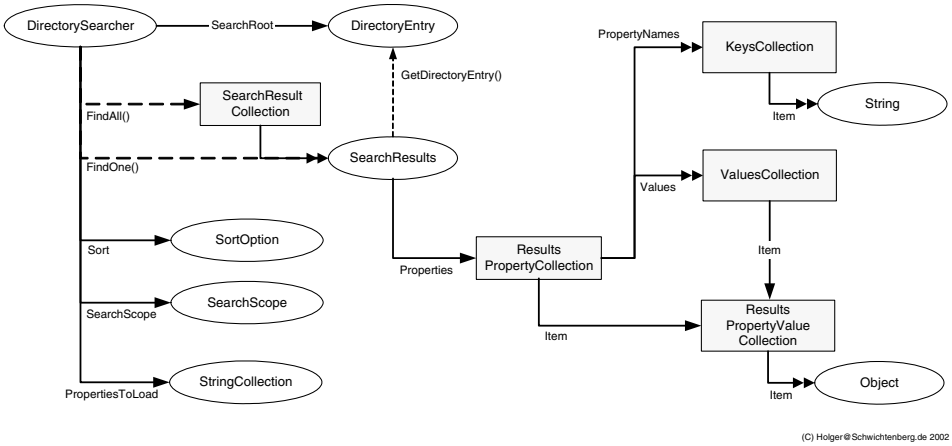


***Abbildung 40.10**: Objektmodell für LDAP-Suche*

- Die `SearchResultCollection` enthält einzelne `SearchResult`-Objekte.
- Von einem `SearchResult`-Objekt kann man entweder lesend auf die abgefragten Attribute zugreifen oder aber man lässt sich von der Methode `GetDirectoryEntry()` ein `DirectoryEntry`-Objekt für den gefundenen Verzeichniseintrag liefern. Das so ermittelte `DirectoryEntry`-Objekt ermöglicht auch den Schreibzugriff.

Ergebnis auswerten

Beispiel

- Die `SearchResultCollection` enthält einzelne `SearchResult`-Objekte.
- Von einem `SearchResult`-Objekt kann man entweder lesend auf die abgefragten Attribute zugreifen oder aber man lässt sich von der Methode `GetDirectoryEntry()` ein `DirectoryEntry`-Objekt für den gefundenen Verzeichniseintrag liefern. Das so ermittelte `DirectoryEntry`-Objekt ermöglicht auch den Schreibzugriff.

Suchbeispiel

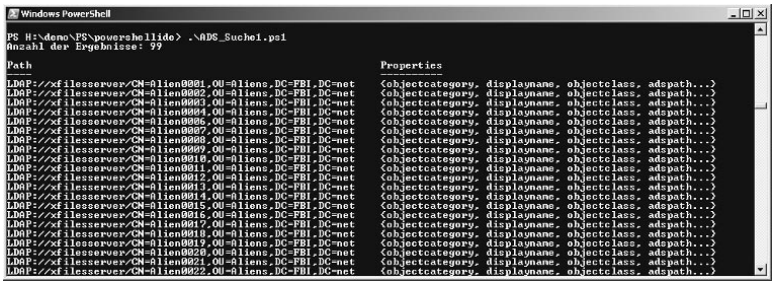
In dem folgenden Beispiel werden im ganzen Active Directory alle Benutzerkonten gesucht, deren Verzeichnisnamen mit dem Buchstaben „Alien“ beginnen.

Listing 40.18 *Ausführen einer LDAP-Suche im AD [ADS_Search_NamePattern.ps1]*

```
$Wurzel = New-Object
system.directoryservices.directoryEntry("LDAP://XFilesServer1/
DC=FBI,DC=net", "FoxMulder", "I+love+Scully")
$Filter = "(&(objectclass=user)(name=alien*))"
$Attribute =
"CN","ObjectClass","ObjectCategory","distinguishedName","lastLogonT
imestamp",
"description","department","displayname"

# Suche zusammenstellen
$Searcher = New-Object DirectoryServices.DirectorySearcher($Wurzel)
$searcher.PageSize = 900
$searcher.Filter = $Filter
$searcher.SearchScope = "subtree"
$Attribute | foreach {[void]$searcher.PropertiesToLoad.Add($_)}
# Suche ausführen
$ergebnis = $searcher.findAll()
"Anzahl der Ergebnisse: " + $ergebnis.Count
$ergebnis
```

Abbildung 40.11 *Suchergebnisse*



Suche nach einem Benutzer mit seinem Anmeldenamen

LDAP-Pfad aus SAMAccount-Name ermitteln

Wenn für einen Benutzer dessen NT 4.0-kompatibler Anmeldename, aber nicht der Pfad des Verzeichnisdiensteintrags bekannt ist, dann hilft nur die Suche im Active Directory mit einer ADSI-Suchanfrage über das Attribut `SAMAccountName`. Wichtig ist dabei, dass hier nur der Benutzername, nicht auch der NT 4.0-kompatible Domänenname anzugeben ist.

Listing 40.19 *Verzeichnisdiensteintrag zu einem Benutzer suchen, dessen SAMAccountName bekannt ist [ADS_Search_SamAccountName.ps1]*

```
$Benutzername = "FoxMulder"
"Suche Benutzer " + $benutzername + "..."
$Wurzel = New-Object
system.directoryservices.directoryEntry("LDAP://XFilesServer1/
DC=FBI,DC=net", "FoxMulder", "I+love+Scully")
$Filter = "(SAMAccountName=" + $benutzername +")"
$Attribute =
"CN","ObjectClass","ObjectCategory","distinguishedName","lastLogonT
imestamp","description","department","displayname"
```

```
# Suche zusammenstellen
$Searcher = New-Object DirectoryServices.DirectorySearcher $Wurzel
$searcher.PageSize = 900
$searcher.Filter = $Filter
$searcher.SearchScope = "subtree"
$Attribute | foreach {[void]$searcher.PropertiesToLoad.Add($_)}
# Suche ausführen
$searcher.findAll()
```

Tipps und Tricks zur Suche

Dieses Unterkapitel enthält Tipps und Tricks zur Suche im Active Directory. **Tipps**

Verwendung indizierter Attribute

Sie sollten möglichst viele indizierte Attribute in Suchanfragen verwenden. Welche Attribute indiziert sind, erfahren Sie in der Dokumentation des Active Directory. Die folgende Abbildung zeigt, wo Sie die Dokumentation der Active-Directory-Attribute im Active-Directory-Schema in der MSDN-Bibliothek finden. Der Eintrag „Is Indexed: True“ zeigt indizierte Attribute an.

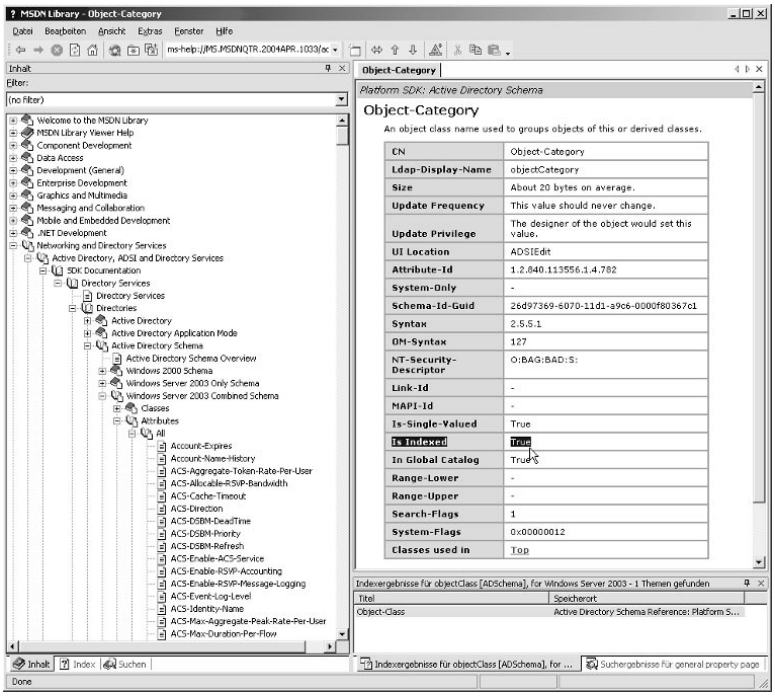


Abbildung 40.12 *Dokumentation der AD-Attribute in der MSDN-Entwicklerbibliothek*

Vermeidung mehrwertiger Attribute

Die Anfrage

```
(&(objectClass=user)(name=f*))
```

ist korrekt, aber aus Leistungsgründen nicht optimal. Schneller ist folgende Anfrage:

```
(objectCategory=person)(objectClass=user)(name=f*))
```

Sie werden feststellen, dass die zweite, längere Abfrage wesentlich schneller ausgeführt wird. Dabei ist die Reihenfolge der Attribute in der Bedingung beliebig; das Active Directory optimiert selbst.

Auffällig in dieser modifizierten Anfrage ist, dass neben der `objectClass` auch ein Bezug auf das Attribut `objectCategory` in der Anfrage enthalten ist. Der Grund dafür liegt darin, dass `objectClass` ein mehrwertiges Attribut ist, das die komplette Vererbungshierarchie der Verzeichnisklasse abbildet. Beispielsweise ist dort für ein `user`-Objekt „top, person, organizationalPerson, user“ abgelegt. Bei einem `computer`-Objekt erkennt man interessanterweise, dass ein Computer eine Spezialisierung eines Benutzers ist, weil `objectClass` für einen Computer enthält: „top, person, organizationalPerson, user, computer“. Eine Suche über ein mehrwertiges Attribut ist sehr zeitaufwändig. Leider existiert im Active Directory kein Attribut, das den Klassennamen in einem einwertigen Attribut enthält.

objectCategory

Neben der Klasse existiert aber auch eine Kategorisierung der Verzeichnisobjekte. Kategorien sind `person`, `group`, `computer` und `organizationalUnit`. `Person` umfasst die Klassen `user` und `contact`. Die Kategorie eines Verzeichnisobjekts ist in `objectCategory` abgelegt und `objectCategory` ist ein indiziertes Attribut, das eine sehr schnelle Suche ermöglicht. Aus diesem Grund ist es sinnvoll, sowohl `objectClass` als auch `objectCategory` in die Bedingungen aufzunehmen.

Die wichtigsten Bedingungen

Die folgende Liste zeigt die korrekten Bedingungen für eine schnelle Suche für verschiedene Verzeichnisklassen:

- Kontakte: `(&(objectclass=contact) (objectcategory=person)`
- Benutzer: `(&(objectclass=user) (objectcategory=person)`
- Gruppen: `(&(objectclass=group) (objectcategory=group)`
- Organisationseinheiten: `(&(objectclass=organizationalUnit)`
- `(objectcategory=organizationalUnit)`
- Computer: `(&(objectclass=user) (objectcategory=computer)`

Vermeidung des Sternoperators Als weiteren Tipp zur Optimierung von Active-Directory-Suchanfragen sollten Sie die Verwendung von Platzhaltern (Stern-Operator `*`) am Anfang einer Zeichenkette vermeiden.

Begrenzung für Suchanfragen

Das Active Directory begrenzt in der Standardkonfiguration die Anzahl der Suchergebnisse auf 1000. Diese Einstellung können Sie in den Domänenrichtlinien ändern.

```
C:\> ntdsutil
ntdsutil: ldap policies
ldap policy: connections
server connections: connect to server SERVERNAME
Connected to SERVERNAME using credentials of locally logged on user
server connections: q
ldap policy: show values

Policy                          Current(New)

…MaxPageSize                     1000…
ldap policy: set maxpagesize to ##### (for example, 50000)
ldap policy: commit changes
ldap policy: q
ntdsutil: q
Disconnecting from SERVERNAME ...
```

Listing 40.20 *Änderung der Domänenrichtlinie für die Suchbegrenzung durchntdsutil.exe*

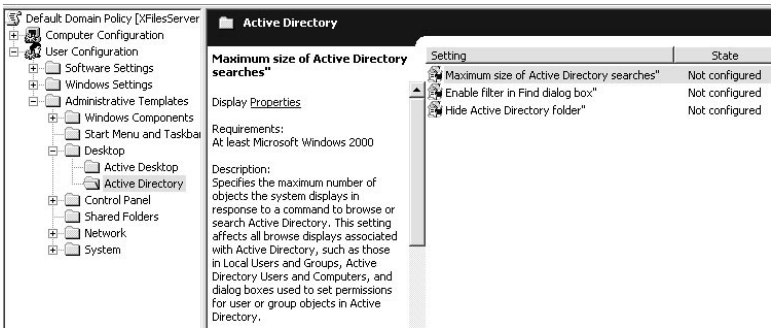


Abbildung 40.13 *Änderung der Domänenrichtlinie für die Suchbegrenzung durch die MMC*

Weitere Beispiele für LDAP-Suchanfragen

Die folgende Liste enthält weitere Beispiele für mögliche Filter bei der Suche nach Benutzerkonten:

- Alle Benutzer, deren Name mit s beginnt:

  ```
  (&(objectCategory=person)(objectClass=user)(name=s*))
  ```

- Alle Benutzer, für die es keine Beschreibung gibt:

  ```
  (&(objectCategory=computer)(!description=*))
  ```

- Alle Benutzer, die deaktiviert sind:

  ```
  (&(objectCategory=person)(objectClass=user)(userAccountControl:
  1.2.840.113556.1.4.803:=2))
  ```

Die Herausforderung in diesem Fall besteht darin, dass die Deaktivierungsinformation in einem einzelnen Bit in `userAccountControl` abgelegt ist. Ein Vergleich auf einen bestimmten Wert nur mit dem Gleichheitszeichen würde nicht zum Ziel führen. Notwendig ist ein bitweises UND. Leider wird dies in LDAP kompliziert durch die Angabe „1.2.840.113556.1.4.803“ ausgedrückt. Ein bitweises ODER wäre der Wert „1.2.840.113556.1.4.804“.

- Alle Benutzer, bei denen „Kennwort läuft nie ab“ gesetzt ist:

```
(&(objectCategory=person)(objectClass=user)
(userAccountControl:1.2.840.113556.1.4.803:=65536))
```

- Alle Benutzer, die nach dem 10.11.2004 angelegt wurden:

```
(&(objectCategory=person)(objectClass=user)
(whenCreated>=20041110000000.0Z))
```



Eine Abfrage, die nur aus der Bedingung `class=*` besteht, funktioniert nicht. Um alle Verzeichnisobjekte zurückzuliefern, muss der Sternoperator auf ein anderes Attribut angewendet werden.

Verwendung von Get-ADObject

Die PowerShell Community Extensions beinhalten das Commandlet `Get-ADObject`, mit dem man Einträge, die bestimmten Kriterien entsprechen, aus dem Active Directory filtern kann. Ausgabeobjekte sind vom Typ `System.DirectoryServices.DirectoryEntry`.

Tabelle 40.6 *Beispiel für den Einsatz von Get-ADObject*

`Get-ADObject -Class user` Liefert alle Benutzerkonten (Instanzen der Verzeichnisdienstklasse „user“)
`Get-ADObject -value "*domain*"` Liefert alle Verzeichnisdienstobjekte, deren Name das Wort „Domain“ enthält
`Get-ADObject -Filter "(&(objectCategory=person)(objectClass=user)(userAccountControl:1.2.840.113556.1.4.803:=2))"` Liefert alle deaktivierten Benutzerkonten
`Get-ADObject -Server E02 -SizeLimit 10` Liefert die ersten zehn Verzeichniseinträge von Domänencontroller E02
`Get-ADObject -Server E02 -Scope subtree -DistinguishedName "CN=Users,DC=IT-Visions,DC=local"` Liefert alle Einträge in dem Container „Users“ und seinen Untercontainern

40.7 Navigation im Active Directory mit den PowerShell Extensions

PowerShell Provider

Durch Installation der PowerShell Community Extensions (PSCX) [CODEPLEX01] steht das Active Directory als Navigationscontainer (alias PowerShell Provider) bereit. Beim Start der PowerShell erzeugen die PSCX automatisch ein neues Laufwerk für das Active Directory, zu dem der Computer gehört. Das Laufwerk wird benannt wie der NT 4.0-kompatible Domänenname (also z.B. „FBI:" für die Domäne mit dem DNS-Namen „fbi.net").

Der folgende Befehl selektiert aus dem „users"-Container des Active Directory alle Gruppen, die das Wort „Domain" im Namen tragen, und gibt diese Liste nach Namen sortiert aus.

```
dir FBI:/users | where { ($_.name -match "domain") -and ($_.Type -
match "group") } | sort name
```

Um eine neue Organisationseinheit mit Namen „Directors" anzulegen, braucht man nur einen Befehl:

```
New-Item -path FBI://Directors -type organizationalunit
```

40.8 Verwendung der Active-Directory-Erweiterungen von www.IT-Visions.de

Die Commandlet-Bibliothek von *www.IT-Visions.de* stellt einige Commandlets für die Verzeichnisdienstverwaltung bereit, welche die Arbeit wesentlich vereinfachen. Dort enthalten sind folgende Commandlets:

- `Get-DirectoryEntry`: Zugriff auf ein einzelnes Verzeichnisobjekt
- `Get-DirectoryChildren`: Zugriff auf den Inhalt eines Containerobjekts (listet die Unterelemente auf)
- `Add-User`: Anlegen eines Benutzerkontos mit Kennwort
- `Add-DirectoryObject`: Anlegen eines Verzeichnisobjekts, das kein Kennwort benötigt
- `Remove-DirectoryObject`: Löschen eines Verzeichnisobjekts
- `Get-DirectoryValue`: Auslesen eines Werts für ein Verzeichnisattribut
- `Set-DirectoryValue`: Festlegen eines Werts für ein Verzeichnisattribut



Die Commandlets unterstützen sowohl den Commandlet-basierten Programmierstil:

```
Add-User -RDN $Name -Container ("WinNT://" + $Computer) -
Password "geheim"
Set-DirectoryValue -Path ("WinNT://" + $Computer +"/" + $Name) -
Name "Fullname" -Value "Dr. Holger Schwichtenberg"
```

als auch den objektbasierten Stil, da die Commandlets die entsprechenden Objekte in die Pipeline legen:

```
$u = Add-User -Password "geheim" -RDN $Name -Container ("WinNT:/
/" + $Computer)
$u.Fullname
$u.PSBase.CommitChanges()
```

Beispiel Das folgende Beispiel zeigt die Anwendung der Commandlets – wahlweise für eine lokale Windows-Benutzerdatenbank (getestet auf einem Windows-Server-2003-Mitgliedsserver) oder ein Active Directory (getestet auf einem Windows-Server-2003-Domänencontroller).

Listing 40.21 *Beispiel für verschiedene Verzeichnisdienstoperationen mit dem Win-NT-Provider unter Verwendung der www.IT-Visions.de-Commandlets [Einsatzgebiete\Benutzer\Local-User_IT-Visions_Commandlets.PS1]*

```
##############################################
## Testskript für Verzeichnisdienstzugriff mit den
## www.IT-Visions.de PowerShell Commandlets
## Dr. Holger Schwichtenberg 2007 - 2010
##############################################

# Parameter

# WinNT
$Name = "FoxMulder"
$Computer = "F171"
$Container = "WinNT://$Computer"


# LDAP (ADS)
$Name = "cn=FoxMulder"
$Container = "LDAP://XFilesServer1/OU=Agents,DC=FBI,DC=net"

Write-Host "Zugriff auf Container" -ForegroundColor yellow
Get-DirectoryEntry $Container | select name
# Alternative: Get-DirectoryEntry "WinNT://$Computer" | select name

Write-Host "Benutzer anlegen" -ForegroundColor yellow
$u = Add-User -Name $Name -Container $Container -Password
"sdd%24343423" -verbose

Write-Host "Attribute setzen - Commandlet-Stil" -ForegroundColor
yellow
Set-DirectoryValue -Path $u.psbase.path -Name "Description" -Value
```

```
"Autor dieses Buchs"

Write-Host "Attribute setzen - Objektstil" -ForegroundColor yellow
$u.Description = "Autor dieses Buchs"
$u.PSBase.CommitChanges()
# Alternative: Set-DirectoryValue -Path ("WinNT://" + $Computer +"/
" + $Name) -Name "Fullname" -Value "Agent Fox Mulder"

Write-Host "Benutzer auslesen - Objektstil" -ForegroundColor yellow
$u = Get-DirectoryEntry $u.psbase.path
"Name: " + $u.Description

Write-Host "Benutzer auslesen - Commandlet-Stil" -ForegroundColor
yellow
Get-DirectoryValue -Path $u.psbase.path  -Name "Description"

Write-Host "Benutzer löschen" -ForegroundColor yellow
Remove-DirectoryEntry $u.psbase.path

Write-Host "Liste aller Containerelement" -ForegroundColor yellow
Get-DirectoryChildren $Container | select name
```

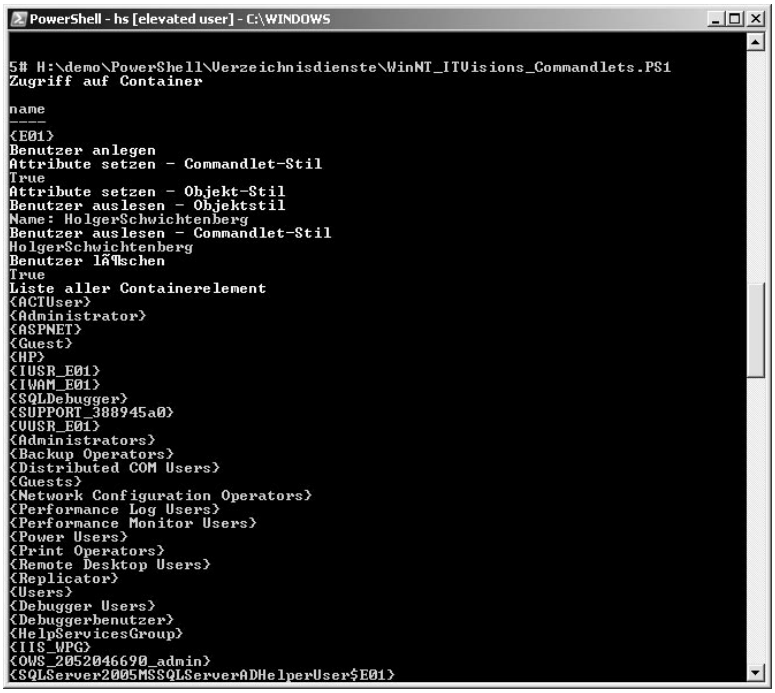


Abbildung 40.14
Ausschnitt aus der Ausgabe des obigen Skripts

40.9 PowerShell-Modul „Active Directory“ (ADPowerShell)

ADPowerShell Windows Server 2008 Release 2 enthält ein Active-Directory-PowerShell-Modul mit 76 Commandlets und einem Navigationsprovider. Das Modul wird bei Microsoft „Active Directory PowerShell“ (kurz: ADPowerShell) genannt. Das Modul kann man im Rahmen von RSAT auch auf Windows 7 installieren.



Man kann auch Instanzen von Active Directory Lightweight Directory Services (AD LDS) mit dem Modul verwalten.



Das neue GUI-Werkzeug für das Active Directory in Windows Server 2008 R2, das Active Directory Administrative Center (ADAC), basiert komplett auf dem ADPowerShell-Modul.



Zu dem Active-Directory-Modul gibt es ein eigenes Weblog: *http://blogs.msdn.com/adPowerShell/*.

Architektur und Installation

ADWS Das Active-Directory-Modul nutzt für den Zugriff auf einen Domänencontroller nicht das LDAP-Protokoll, sondern Webservices. Voraussetzung ist daher, dass auf dem Domänencontroller die Active Directory Web Services (ADWS) installiert sind. ADWS wird automatisch auf einem Windows-Server-2008-R2-Domänencontroller installiert. ADWS basieren auf der .NET-Kommunikationsinfrastruktur „Windows Communication Foundation (WCF)“ und diversen W3C-Standards wie WS-Transfer sowie Microsoft-eigenen Erweiterungen (z.B. WS-Enumeration).

ADWS verwendet TCP-Port 9389 und ist implementiert im Windows-Systemdienst „ADWS“ (Microsoft.ActiveDirectory.WebServices.exe). Das Installationsverzeichnis ist C:\Windows\ADWS. Die Konfiguration ist möglich über Microsoft.ActiveDirectory.WebServices.exe.config.

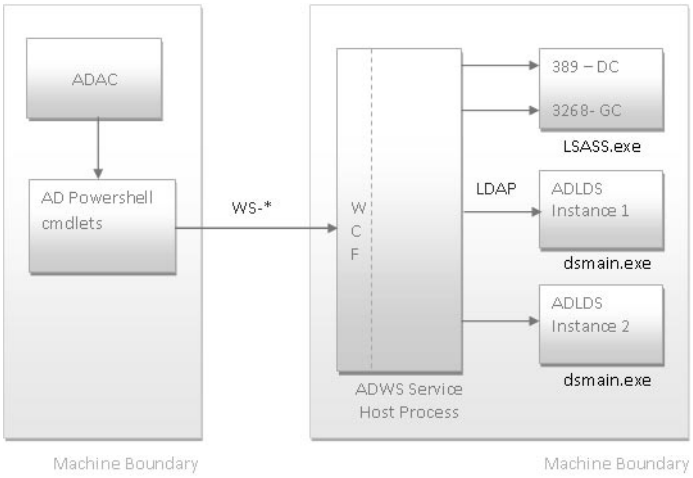


***Abbildung 40.15**: Das Schaubild zeigt den Weg vom Active Directory Administrative Center (ADAC) über die PowerShell und WCF bis zum Dienst lsass.exe (für Active Directory Domain Services) bzw. dsmain.exe (für Lightweight Directory Services). [Quelle: http://blogs.msdn.com/adPowerShell/archive/2009/04/06/active-directory-web-services-overview.aspx]*

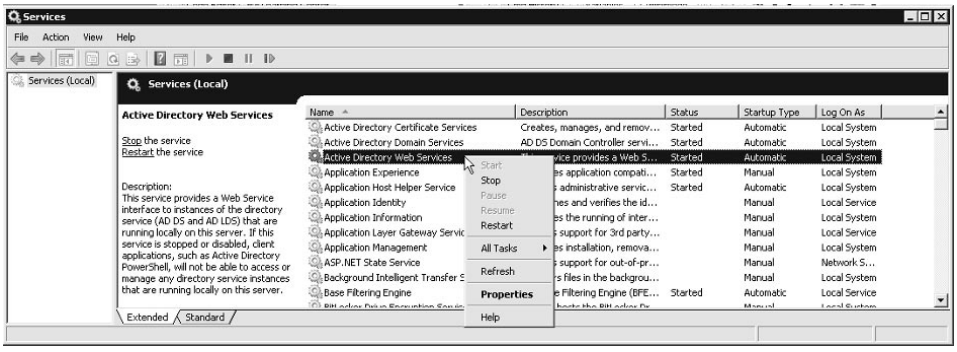


***Abbildung 40.16**: Der Systemdienst „Active Directory Web Services“ (Microsoft.ActiveDirectory.WebServices.exe), der ADWS realisiert*

Beim Auftreten des Fehlers „Unable to find a default server with Active Directory Web Services running.“ Prüfen Sie, ob der ADWS-Dienst läuft und erreichbar ist.

ADMGS

ADWS sind für Windows Server 2008 (mit oder ohne SP2) sowie Windows Server 2003 (auch R2, mit SP2) als Zusatzkomponenten verfügbar. Dazu muss man dort den „Active Directory Management Gateway Service“ (ADMGS) installieren. ADMGS basiert auf .NET Framework 3.5 mit SP1, das vorher installiert sein muss [MS04].

Hinweis: ADMGS entspricht hinsichtlich der Funktionalität ADWS unter Windows Server 2008 R2. Warum hier zwei Namen notwendig waren, bleibt schleierhaft.

Auf einem Windows Server 2008 R2, der nicht Domänencontroller ist, kann man das ADPowerShell-Modul einzeln (z.B. mit Hilfe des PowerShell-Moduls Servermanager) installieren (RSAT steht hier für „Remote Server Administration Tools“):

```
import-module servermanager
Add-WindowsFeature -Name "RSAT-AD-PowerShell" -IncludeAllSubFeature
```

RSAT Unter Windows 7 kann man das AD-Modul ebenfalls nutzen (zum Zugriff auf entsprechend ausgestattete Domänencontroller). Hier muss man die Microsoft Remote Server Administration Tools (RSAT) für Windows 7 installieren und danach unter den optionalen Features das „Active Directory Module for Windows PowerShell“ aktivieren (siehe *Abbildung 40.17*).

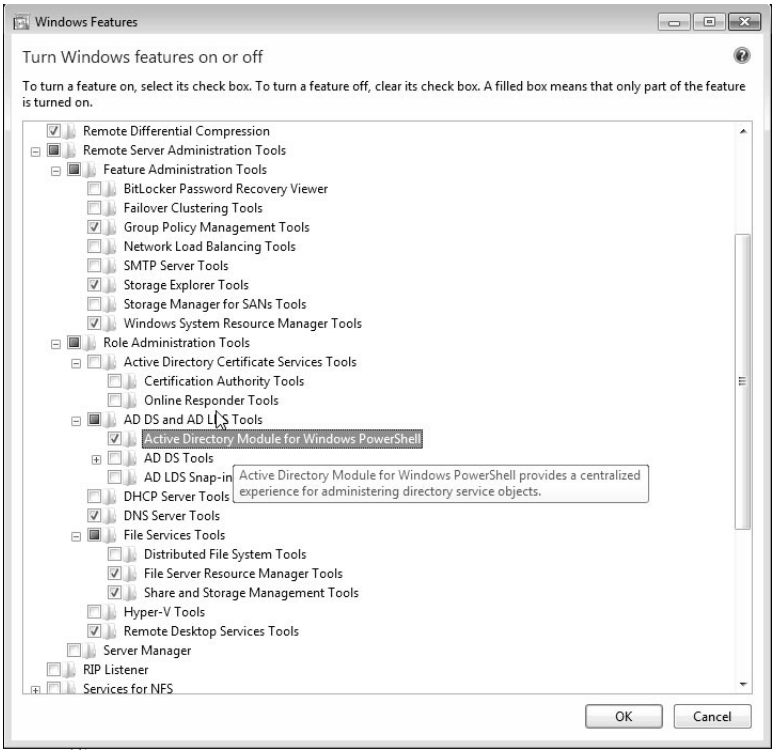


Abbildung 40.17 *Aktivieren von ADPowerShell in Windows 7*

Aktivieren des Active-Directory-Moduls

Das Active-Directory-Modul ist im Standard nicht aktiv, wenn man die Windows PowerShell 2.0 auf Windows Server 2008 R2 startet. Es gibt zwei Möglichkeiten:

- Start des Eintrags „Active Directory-Module für Windows PowerShell“ unter „Administrative Tools“ im Startmenü. Dies startet eine PowerShell-Konsole mit aktiviertem Active-Directory-Modul.

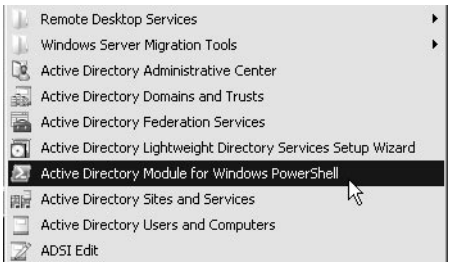


Abbildung 40.18 *Active-Directory-Werkzeuge in Windows Server 2008 R2*

- Eingabe von „Import-Module ActiveDirectory“ in einer normalen PowerShell-Konsole

Wie die folgende Bildschirmabbildung zeigt, erhöht sich die Anzahl der Commandlets/Funktionen um 76 und die Anzahl der Navigationsprovider um eins.

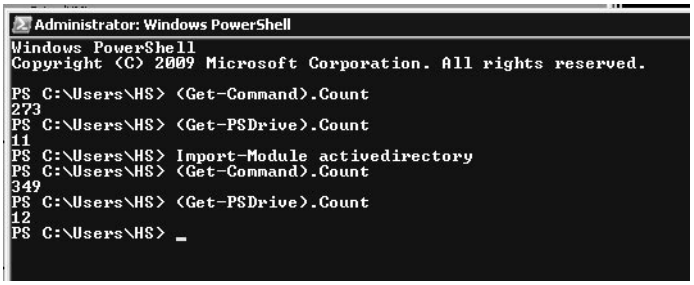


Abbildung 40.19 *Veränderungen nach dem Import des Moduls*

Active-Directory-Navigationsprovider

AD:

Der PowerShell-Navigationsprovider für das Active Directory trägt den Namen „ActiveDirectory“. Auf einem Windows-Server-2008-R2-Domänencontroller oder einem Windows 7-System mit aktiviertem ADPowerShell-Modul wird durch die Aktivierung des Active-Directory-Moduls automatisch ein Laufwerk „AD:“ eingerichtet, das zur Wurzel des Active Directory, dem Element „rootDSE“, führt. Unter diesem existieren der DefaultNamingContext, der ConfigurationNamingContext und der SchemaNamingContext.

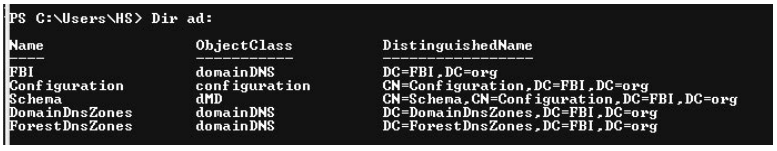


Abbildung 40.20 *Ausführung von Dir ad*



Zu beachten ist, dass zur Navigation im Active Directory der Provider nicht den Namen, sondern den DN (Distinguished Name) verwendet. Falsch ist also:

```
Dir ad:\FBI
```

(auch wenn die Anzeige von `DIR ad:` dies suggeriert), sondern richtig ist:

```
Dir ad:\"dc=FBI,dc=org"
```

(mit den Anführungszeichen!).

Abbildung 40.21 *Auflisten des DefaultNaming-Context im Active Directory „FBI.org“*

```
PS C:\Users\HS> Dir ad:\"DC=FBI,DC=org"

Name                ObjectClass           DistinguishedName
----                -----------           -----------------
Agents              organizationalUnit    OU=Agents,DC=FBI,DC=org
Aliens              organizationalUnit    OU=Aliens,DC=FBI,DC=org
Builtin             builtinDomain         CN=Builtin,DC=FBI,DC=org
Computers           container             CN=Computers,DC=FBI,DC=org
Directors           organizationalUnit    OU=Directors,DC=FBI,DC=org
Domain Controllers  organizationalUnit    OU=Domain Controllers,DC=FBI,DC=org
ForeignSecurityPr... container            CN=ForeignSecurityPrincipals,DC=FBI,DC=org
Infrastructure      infrastructureUpdate  CN=Infrastructure,DC=FBI,DC=org
Lone Gunmen         organizationalUnit    OU=Lone Gunmen,DC=FBI,DC=org
LostAndFound        lostAndFound          CN=LostAndFound,DC=FBI,DC=org
Managed Service A... container            CN=Managed Service Accounts,DC=FBI,DC=org
NTDS Quotas         msDS-QuotaContainer   CN=NTDS Quotas,DC=FBI,DC=org
Program Data        container             CN=Program Data,DC=FBI,DC=org
Syndicate           organizationalUnit    OU=Syndicate,DC=FBI,DC=org
System              container             CN=System,DC=FBI,DC=org
The Lone Gunmen     organizationalUnit    OU=The Lone Gunmen,DC=FBI,DC=org
Users               container             CN=Users,DC=FBI,DC=org
```

Abbildung 40.22 *Auflisten der Organisationseinheit „Agents“ im Active Directory „FBI.org“*

```
PS C:\Users\HS> Dir ad:\"OU=Agents,DC=FBI,DC=org"

Name                ObjectClass           DistinguishedName
----                -----------           -----------------
All Agents          group                 CN=All Agents,OU=Agents,DC=FBI,DC=org
Dana Scully         user                  CN=Dana Scully,OU=Agents,DC=FBI,DC=org
Fox Mulder          user                  CN=Fox Mulder,OU=Agents,DC=FBI,DC=org
John Doggett        user                  CN=John Doggett,OU=Agents,DC=FBI,DC=org
Monica Reyes        user                  CN=Monica Reyes,OU=Agents,DC=FBI,DC=org
```

Man kann mit `CD` den aktuellen Pfad ins Active Directory setzen

```
cd ad:"ou=Agents,dc=FBI,dc=org"
```

oder einen neuen Laufwerksnamen definieren:

```
New-PSDrive -Name Agents -PSProvider ActiveDirectory -Root
AD:"ou=Agents,dc=FBI,dc=org"
Dir Agents:
```

Mit `Get-Item` kann man gezielt auf einzelne Active-Directory-Einträge zugreifen. Allerdings stellt man schnell fest, dass man auf diesem Wege nur die Basisinformationen für einen Eintrag, nicht aber die spezifischen Attribute erreichen kann.

Abbildung 40.23 *Einsatz von Get-Item*

```
PS C:\Users\HS> $fm = Get-Item ad:\"CN=Fox Mulder,OU=Agents,DC=FBI,DC=org"
PS C:\Users\HS> $fm | fl


Name              : Fox Mulder
ObjectClass       : user
DistinguishedName : CN=Fox Mulder,OU=Agents,DC=FBI,DC=org
ObjectGuid        : b0719f9f-721b-418d-ae85-b039a1a93e6f


PS C:\Users\HS> _
```

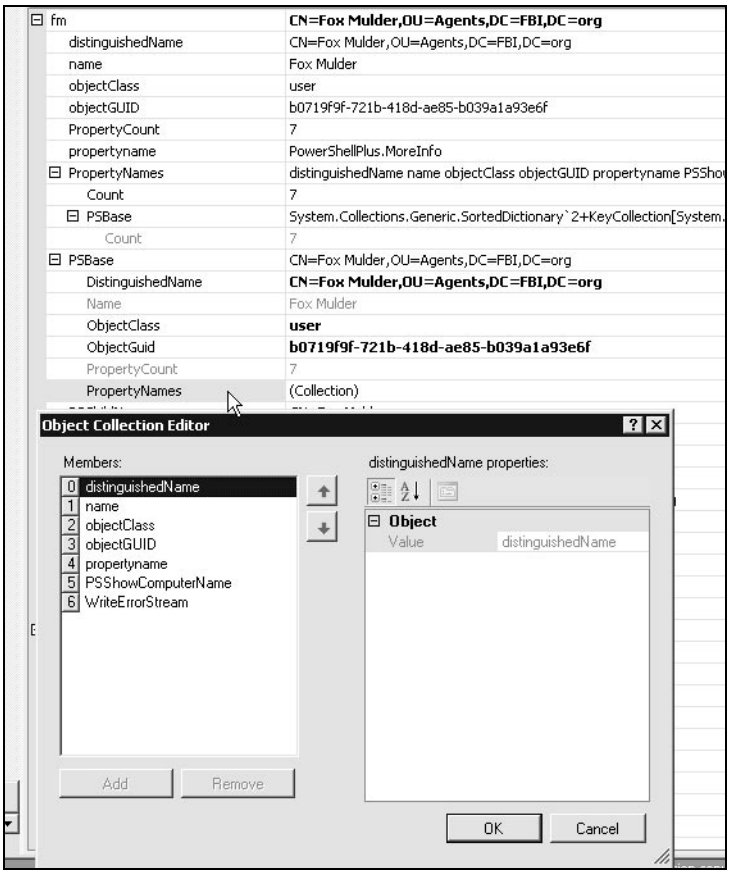


Abbildung 40.24 *PowerShellPlus offenbart, dass es keine spezifischen Objektinformationen über den Navigationsprovider gibt.*

Objektmodell

Die Commandlets des ADPowerShell-Moduls bieten mehr Möglichkeiten als der Provider. Die Commandlets verwenden ein eigenes Objektmodell zur Abbildung der ADS-Strukturen. Die Vererbungshierarchie der Datenklassen im Active-Directory-Modul entspricht nicht der Vererbungshierarchie im Active-Directory-Schema. Im AD-Schema ist z.B. „Computer“ eine Spezialisierung von „User“. Im ADPowerShell-Objektmodell sind `ADUser` und `ADComputer` auf gleicher Ebene Unterklassen von `ADAccount`.

```
ADEntity
  ADRootDSE
  ADObject
    ADFineGrainedPasswordPolicy
    ADOptionalFeature
    ADOrganizationalUnit
    ADPartition
      ADDomain
    ADPrincipal
      ADAccount
        ADComputer
        ADServiceAccount
        ADUser
      ADGroup
  ADDefaultDomainPasswordPolicy
  ADForest
  ADDirectoryServer
    ADDomainController
```

Abbildung 40.25 *Das Vererbungsmodell der Datenklassen im Active-Directory-Modul (Quelle: Hilfe zum Modul „ADPowerShell“)*

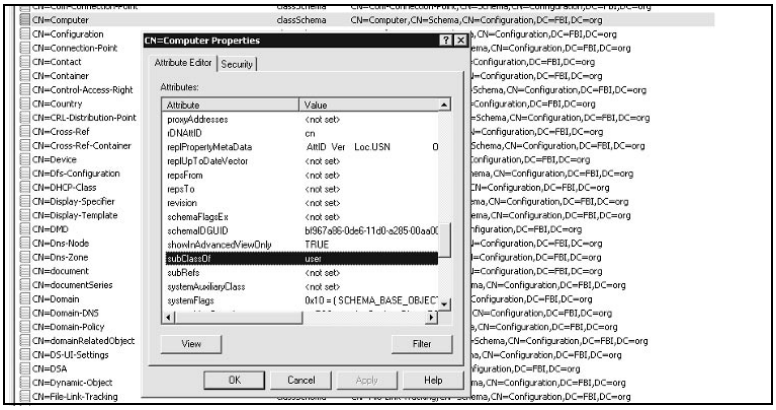


Abbildung 40.26 *Im ADS-Schema ist „Computer“ eine Unterklasse von „User“ (hier angezeigt im Werkzeug ADSI Edit).*

Die Datenklassen besitzen zwei Arten von Attributen:

- Direkt aus dem Active Directory stammende Attribute (z.B. `ObjectClass`, `City`, `GivenName`). Die Attribute haben zum Teil von den LDAP-Namen abweichende Bezeichnungen (z.B. „City“ statt „l“).
- Zusammengesetzte Attribute, die mehrere Daten aus dem Active Directory zusammenfassen. Ein Beispiel dafür ist `ProtectedFromAccidentalDeletion`. Dieser Boolean-Wert (Ja-/Nein-Wert) ergibt sich aus `nTSecurityDescriptor`, `sdRightsEffective`, `instanceType` und `isDeleted`.

Name des Attributs im AD-Modul	LDAP-Name
Name	name
ObjectClass	objectClass
ObjectGUID	objectGUID
CN	cn
DistinguishedName	distinguishedName
DisplayName	displayName

Tabelle 40.7*: Abbildung von LDAP-Namen auf Namen im AD-Modul*

Name des Attributs im AD-Modul	LDAP-Name
Description	description
Title	title
Surname	sn
GivenName	givenName
City	l
StreetAddress	street
Country	c
Office	physicalDeliveryOfficeName
Fax	facsimileTelephoneNumber
EmailAddress	mail
SamAccountName	sAMAccountName
HomeDrive	homeDrive
HomeDirectory	homeDirectory
ProfilePath	profilePath
ProtectedFromAccidental-Deletion	nTSecurityDescriptor, sdRights-Effective, instanceType, isDeleted

***Tabelle 40.7**: Abbildung von LDAP-Namen auf Namen im AD-Modul (Forts.)*

Überblick über die Commandlets

Arten von Commandlets im ADPowerShell -Modul

Die beim Active-Directory-PowerShell-Modul mitgelieferten Commandlets lassen sich in drei Gruppen unterteilen:

- Allgemeine Verwaltungs-Commandlets
- Kontenverwaltung (Organisationseinheiten, Benutzer, Gruppen, Computer) mit den Untergruppen:
  - Lebenszyklusverwaltung
  - Kontoeigenschaftenverwaltung
  - Dienstkontenverwaltung
  - Gruppenmitgliedschaftsverwaltung
  - Kennwortrichtlinienverwaltung
- Topologieverwaltung
  - Verwaltung der Domänen und Wälder
  - Verwaltung der Domänencontroller
  - Verwaltung der optionalen Features
  - Verwaltung der Replikationsrichtlinien

In dem folgenden Schaubild sind zusätzlich die allgemeinen PowerShell-Commandlets für die Providerverwaltung erwähnt.

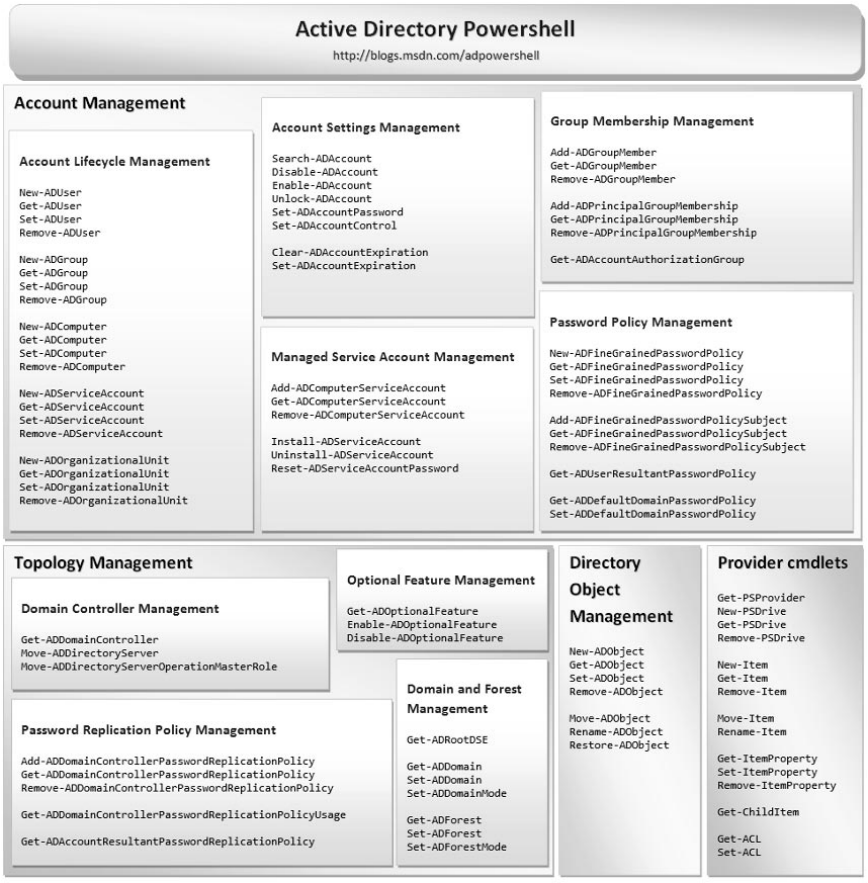


***Abbildung 40.27**: Post der Commandlets aus dem Active-Directory-PowerShell-Modul [Quelle: http://blogs.msdn.com/adPowerShell/archive/2009/03/05/active-directory-PowerShell-overview.aspx]*

Allgemeine Verwaltungs-Commandlets

Das Active-Directory-Modul stellt folgende allgemeine Commandlets bereit, mit denen sich Active-Directory-Objekte unabhängig von der Verzeichnisdienstklasse verwalten lassen:

- `Get-ADObject`: holt ein AD-Objekt
- `Set-ADObject`: setzt Werte in einem AD-Objekt
- `New-ADObject`: erzeugt ein neues AD-Objekt (unter Angabe des Klassennamens)
- `Remove-ADObject`: löscht ein AD-Objekt
- `Rename-ADObject`: Umbenennen eines AD-Objekts
- `Move-ADObject`: Verschieben eines AD-Objekts
- `Restore-ADObject`: Wiederherstellen eines gelöschten AD-Objekts

Das folgende Listing zeigt ein Skript, das eine Organisationseinheit löscht, an einem anderen Ort unter anderem Namen wieder anlegt, dann verschiebt und umbenennt.

Listing 40.22 *Beispielskript für den Einsatz der allgemeinen AD-Commandlets [WPS2_ADS_CommonCommandlets.ps1]*

```
"Delete and Recreate an OU....."

$ou = Get-ADObject "ou=Agents,dc=FBI,dc=org"
$ou | fl

Set-adobject "ou=Agents,dc=FBI,dc=org" -
protectedFromAccidentalDeletion $false
Remove-ADObject "ou=Agents,dc=FBI,dc=org" -confirm:$false -
recursive

New-ADObject -type "OrganizationalUnit" -
ProtectedFromAccidentalDeletion $false -name "Alien-Agents" -Path
"ou=Aliens,dc=FBI,dc=org"

"Move an OU..."

Move-ADObject -Identity "ou=Alien-Agents,ou=Aliens,dc=FBI,dc=org" -
targetpath "dc=FBI,dc=org"

Rename-ADObject "ou=Alien-Agents,dc=FBI,dc=org" -newname "Agents"

# Option #1
Set-ADObject "ou=Agents,dc=FBI,dc=org" -description "FBI Agents"
# Option #2
Set-ADObject "ou=Agents,dc=FBI,dc=org" -replace
@{ManagedBy="cn=Walter Skinner,ou=Directors,dc=fbi,dc=org"}
# Option #3
$newou = Get-ADObject "ou=Agents,dc=FBI,dc=org"
$newou.ManagedBy = "cn=Walter Skinner,ou=Directors,dc=fbi,dc=org"
Set-ADObject -instance $newou

"Ergebnis:"
$ou = Get-ADObject  "ou=Agents,dc=FBI,dc=org"
$ou | fl
```

Interessant im obigen Listing sind vor allem die drei Wege, auf denen `Set-ADObject` arbeitet:

- Option 1: Es gibt einige wenige AD-Attribute bzw. Attribute der AD-Modul-Klassen, die Parameter des Commandlets `Set-ADObject` sind (z.B. `Description` und `ProtectedFromAccidentalDeletion`). Diese können direkt gesetzt werden.
- Option 2: Andere Attribute können entweder unter Angabe im „Replace“-Parameter gesetzt werden (hier z.B. „Managedby“).
- Option 3: Oder diese Attribute können in objektorientierter Vorgehensweise in das Objekt geschrieben werden. `Set-ADObject` sorgt dann für die Übermittlung der Änderungen an das AD.



Das Commandlet `Remove-ADObject` bietet leider nicht den sonst üblichen Parameter `-Force`, mit dem man schreibgeschützte AD-Einträge („ProtectedFromAccidentalDeletion“) einfach löschen kann. Daher muss man vorher `ProtectedFromAccidentalDeletion = $false` setzen, da sonst `Remove-ADObject` mit einem Fehler abbrechen könnte!

Filtern und Suchen

LDAP-Filtersyntax

Mit `Get-ADObject` kann man Objekte suchen. Dabei bietet ADPowerShell wahlweise die LDAP-Filtersyntax (mit Präfix-Notation) oder eine vereinfachte Infix-Notation an, die an die PowerShell-Ausdruckssyntax angelehnt ist.

Mit `-searchbase` legt man den Ausgangspunkt der Suche fest, z.B.:

```
-searchbase "ou=agents,dc=fbi,dc=org"
```

Mit `-searchscope` legt man die Tiefe der Suche fest. Erlaubt sind: `Base`, `OneLevel` und `SubTree`, z.B.:

```
-SearchScope SubTree
```

Der folgende Befehl liefert aus dem aktuellen Active-Directory-Pfad die ersten fünf Einträge, deren Name mit „F“ beginnt.

```
Get-ADObject -Filter 'Name -like "f*"' -SearchScope SubTree -
resultSetsize 5
```

Der folgende Befehl liefert alle Einträge, was durch `$null` bei `resultSetsize` anzugeben ist (im Standard würden sonst nur die ersten 1000 ausgegeben!).

```
Get-ADObject -Filter 'Name -like "f*"' -SearchScope SubTree -
resultSetsize $null
```

Der folgende Befehl listet alle Einträge auf, in denen der Vorname mit „F“ und der Nachname mit „M“ beginnt.

```
Get-ADObject -Filter 'givenname -like "f*" -and sn -like "m*"' -
SearchScope SubTree -resultSetsize $null
```



Man muss die LDAP-Attributnamen (z.B. „sn“) verwenden, nicht die Attributnamen der PowerShell (wie „surname“).

Anstelle der moduleigenen Filtersyntax kann man auch die LDAP-Suchsprache verwenden. Das folgende Beispiel sucht korrekt in allen Benutzerkonten, die mit „F“ beginnen.

```
Get-ADObject -LDAPFilter
'(&(objectCategory=person)(objectClass=user)(name=f*))' -
SearchScope SubTree -resultSetsize $null
```

Die gleichzeitige Verwendung von `objectCategory` und `objectClass` in der Suchanfrage steigert die Leistung.



Die Tabelle zeigt die PowerShell-Filter und die äquivalenten LDAP-Filter.

PowerShell-Operator	LDAP-Operator	Beschreibung
-eq	=	Gleich
-ne	! x = y	Ungleich
-like	=	Mustergleichheit
-notlike	! x = y	Musterungleichheit
-le	<=	Kleiner gleich
-lt	! x >= y	Kleiner
-ge	>=	Größer gleich
-gt	! x <= y	Größer
-and	&	Und
-or	|	Oder
-not	!	Nicht
-bor	:1.2.840.113556.1.4.804:=	Bitweise Oder
-band	:1.2.840.113556.1.4.803:=	Bitweise Und

Tabelle 40.8 *Vergleich der Filtersprachen (Quelle: [MSBlog01])*

Die PowerShell-Suchsyntax kann viel einfacher als die LDAP-Syntax sein, wie ein abschließendes Beispiel zeigt. Aufgabe ist es, alle Benutzer zu finden, die sich in den letzten fünf Tagen angemeldet haben.

```
$date = (Get-date) - (New-Timespan -days 5)
Get-ADUser -Filter { lastLogon -gt $date }
```

Mit der LDAP-Syntax wäre dies (die Zeitangabe erfolgt in Einheiten zu 100 Nanosekunden seit dem 1.1.1601):

```
Get-ADUser -LDAPFilter "(&(lastLogon>=128812906535515110)
(objectClass=user)(!(objectClass=computer)))"
```

Verwaltung von Organisationseinheiten

OU

Zur Verwaltung von Organisationseinheiten stehen folgende spezielle Commandlets zur Verfügung:

- `Get-ADOrganizationalUnit`
- `New-ADOrganizationalUnit`
- `Remove-ADOrganizationalUnit`
- `Set-ADOrganizationalUnit`

Die Neufassung des Skripts aus dem Unterkapitel „Allgemeine Verwaltungs-Commandlets“ unter Einsatz dieser Commandlets zeigt das folgende Listing. Die größten Unterschiede sind:

- Bei `New-ADOrganizationalUnit` muss man im Gegensatz zu `New-ADObject` keinen Klassennamen („Type“) angeben.
- Mit `Set-ADOrganizationalUnit` kann man das Attribut `ManagedBy` direkt setzen.

Listing 40.23 *Beispielskript für den Einsatz der „OU“-Commandlets [WPS2_ADS_OUManagement.ps1]*

```
"Delete and Recreate an OU....."

$ou = Get-ADObject "ou=Agents,dc=FBI,dc=org"
$ou | fl

Set-adobject "ou=Agents,dc=FBI,dc=org" -
protectedFromAccidentalDeletion $false
Remove-ADOrganizationalUnit "ou=Agents,dc=FBI,dc=org" -
confirm:$false -recursive

New-ADOrganizationalUnit -ProtectedFromAccidentalDeletion $false -
name "Alien-Agents" -Path "ou=Aliens,dc=FBI,dc=org"

"Move an OU..."

Move-ADObject -Identity "ou=Alien-Agents,ou=Aliens,dc=FBI,dc=org" -
targetpath "dc=FBI,dc=org"

Rename-ADObject "ou=Alien-Agents,dc=FBI,dc=org" -newname "Agents"

Set-ADOrganizationalUnit "ou=Agents,dc=FBI,dc=org" -ManagedBy
"cn=Walter Skinner,ou=Directors,dc=fbi,dc=org"

"Ergebnis:"
$ou = Get-ADObject  "ou=Agents,dc=FBI,dc=org"
$ou | fl
```

Verwaltung von Benutzerkonten

Konten

Zur Verwaltung von Benutzerkonten stehen im PowerShell-AD-Modul die folgenden Commandlets zur Verfügung:

- `Get-ADUser`: Benutzerkontenliste oder Daten eines Benutzerkontos
- `New-ADUser`: Benutzerkonto anlegen
- `Remove-ADUser`: Benutzerkonto löschen
- `Set-ADUser`: Eigenschaften eines Benutzers festlegen

Zugriff auf Benutzer

Zugriff auf einzelne Benutzer oder Mengen von Benutzern ermöglicht Get-ADUser.

Der folgende Befehl holt den Benutzer „FoxMulder“:

```
Get-ADUser FoxMulder
```

Allerdings liefert Get-ADUser im Standard nur eine sehr kleine Teilmenge aller Attribute eines Benutzers (z.B. GivenName, Surname, SamAcccountname).

```
PS C:\Users\MS> Get-ADUser FoxMulder  | gm

   TypeName: Microsoft.ActiveDirectory.Management.ADUser

Name              MemberType            Definition
----              ----------            ----------
Contains          Method                bool Contains(string propertyName)
Equals            Method                bool Equals(System.Object obj)
GetEnumerator     Method                System.Collections.IDictionaryEnumerator GetEnumerator()
GetHashCode       Method                int GetHashCode()
GetType           Method                type GetType()
ToString          Method                string ToString()
Item              ParameterizedProperty Microsoft.ActiveDirectory.Management.ADPropertyValueCollection Item(string p...
DistinguishedName Property              System.String DistinguishedName {get;set;}
Enabled           Property              System.Boolean Enabled {get;set;}
GivenName         Property              System.String GivenName {get;set;}
Name              Property              System.String Name {get;}
ObjectClass       Property              System.String ObjectClass {get;set;}
ObjectGUID        Property              System.Nullable`1[System.Guid, mscorlib, Version=2.0.0.0, Culture=neutral, ...
SamAccountName    Property              System.String SamAccountName {get;set;}
SID               Property              System.Security.Principal.SecurityIdentifier SID {get;set;}
Surname           Property              System.String Surname {get;set;}
UserPrincipalName Property              System.String UserPrincipalName {get;set;}
```

Abbildung 40.28 *Standardattribute bei Get-ADUser*

Möchte man mehr Attribute nutzen, muss man diese explizit angeben:

```
Get-ADUser FoxMulder -properties City, Company, Office
```

```
PS C:\Users\MS> Get-ADUser FoxMulder -properties City, Company, Office | gm

   TypeName: Microsoft.ActiveDirectory.Management.ADUser

Name              MemberType            Definition
----              ----------            ----------
Contains          Method                bool Contains(string propertyName)
Equals            Method                bool Equals(System.Object obj)
GetEnumerator     Method                System.Collections.IDictionaryEnumerator GetEnumerator()
GetHashCode       Method                int GetHashCode()
GetType           Method                type GetType()
ToString          Method                string ToString()
Item              ParameterizedProperty Microsoft.ActiveDirectory.Management.ADPropertyValueCollection Item(string p...
City              Property              System.String City {get;set;}
Company           Property              System.String Company {get;set;}
DistinguishedName Property              System.String DistinguishedName {get;set;}
Enabled           Property              System.Boolean Enabled {get;set;}
GivenName         Property              System.String GivenName {get;set;}
Name              Property              System.String Name {get;}
ObjectClass       Property              System.String ObjectClass {get;set;}
ObjectGUID        Property              System.Nullable`1[System.Guid, mscorlib, Version=2.0.0.0, Culture=neutral, ...
Office            Property              System.String Office {get;set;}
SamAccountName    Property              System.String SamAccountName {get;set;}
SID               Property              System.Security.Principal.SecurityIdentifier SID {get;set;}
Surname           Property              System.String Surname {get;set;}
UserPrincipalName Property              System.String UserPrincipalName {get;set;}
```

Abbildung 40.29 *Zusätzliche Attribute bei Get-ADUser*

Tipp

Alle Attribute bekommt man mit:

```
Get-ADUser FoxMulder -properties *
```

Man kann mit Get-ADUser auch Filter anwenden (vgl. *Abschnitt 40.10.7 „Filtern und Suchen“*). Der folgende Befehl liefert alle Benutzer in der Organisationseinheit „Agents“, deren Anmeldename mit „F“ beginnt:

```
Get-ADUser -searchbase "ou=agents,dc=fbi,dc=org" -Filter
'samaccountname -like "F*"'
```

Der folgende Befehl liefert alle Benutzer in der Organisationseinheit „Agents“:

```
$oupath = "ou=Agents,dc=FBI,dc=org"
Get-ADUser -Searchbase $oupath -Filter "*"
```

Benutzerkonto anlegen

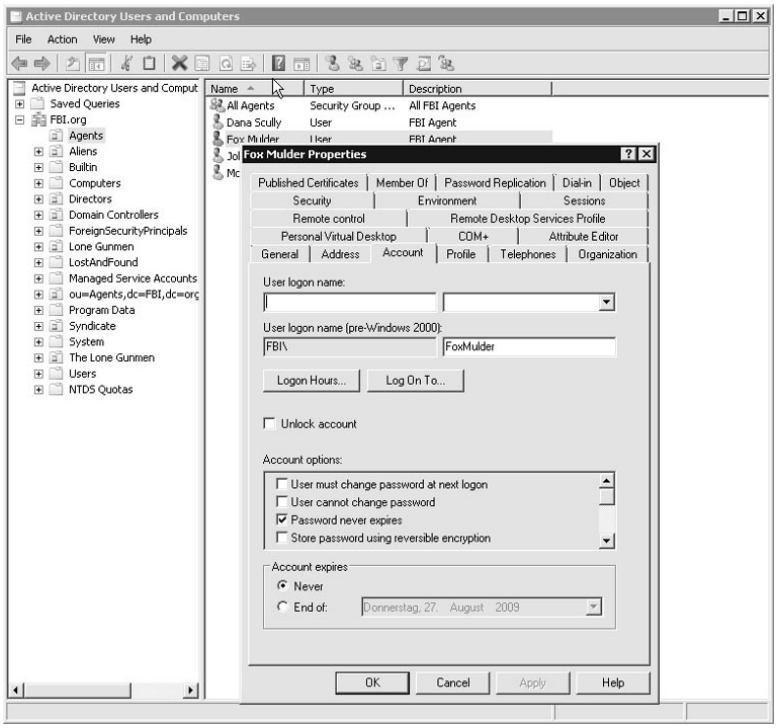
New-ADUser

Zum Anlegen von Benutzerkonten kommt `New-ADUser` zum Einsatz. Dabei kann man zahlreiche Kontoeigenschaften direkt setzen. Auf eine detaillierte Besprechung der Parameter wird hier zur Einsparung von Platz verzichtet, da diese größtenteils selbst erklärend sind.

Listing 40.24 *Anlegen des Agenten „Fox Mulder“ unter Angabe des Kennworts im Skripttext [WPS2_ADS_UserManagement.ps1]*

```
$fm = New-ADUser -path $oupath -Name "Fox Mulder" -SamAccountName
"FoxMulder" -DisplayName "Fox Mulder" -Title "Agent" -Enabled $true
-ChangePasswordAtLogon $false -AccountPassword (ConvertTo-
SecureString "I+love+Scully" -AsPlainText -force) -PassThru -
PasswordNeverExpires:$true -Description "FBI Agent" -HomePage
"www.xfiles.com" -Company "FBI"
```

Abbildung 40.30 *Ergebnis der Ausführung des oben genannten Befehls*



Verwaltung von Benutzergruppen

Zur Gruppenverwaltung gibt es acht Commandlets, davon beziehen sich vier auf die Gruppen und vier auf die Gruppenmitgliedschaften:

- `Get-ADGroup`: Benutzergruppen auflisten
- `New-ADGroup`: Anlegen einer Benutzergruppe
- `Set-ADGroup`: Eigenschaften einer Benutzergruppe setzen
- `Remove-ADGroup`: eine Benutzergruppe entfernen
- `Get-ADGroupMember`: Auflisten der Mitglieder einer Benutzergruppe
- `Get-ADPrincipalGroupMembership`: Auflisten der direkten Mitglieder einer Benutzergruppe
- `Add-ADGroupMember`: Hinzufügen eines Gruppenmitglieds
- `Remove-ADGroupMember`: Entfernen eines Gruppenmitglieds

Das folgende Listing erzeugt die Gruppe „All Agents“ mit vier Agenten. Danach wird eine Agentin „entlassen“ und daher aus der Gruppe entfernt.

Listing 40.25 Verwaltung von Benutzergruppen [WPS2_ADS_User-Management.ps1]

```
"--- Gruppe anlegen..."
New-ADGroup -path $oupath -Name "All Agents" -SamAccountName
"AllAgents" -GroupScope Global -GroupCategory Security -Description
"All FBI Agents" -PassThru
"--- Mitglieder in die Gruppe aufnehmen..."
Add-ADGroupMember -Identity AllAgents -Members FoxMulder
Add-ADGroupMember -Identity AllAgents -Members DanaScully
Add-ADGroupMember -Identity AllAgents -Members JohnDoggett
Add-ADGroupMember -Identity AllAgents -Members MonicaReyes
"--- Gruppenmitglieder:"
Get-ADGroupMember -Identity AllAgents
"--- Mitglied entfernen…"
Remove-ADGroupMember -Identity AllAgents -Members MonicaReyes -
Confirm:$false
"--- Gruppenmitglieder:"
Get-ADGroupMember -Identity AllAgents
```

Mit

```
Get-ADPrincipalGroupMembership -Identity FoxMulder
```

listet man alle Gruppen auf, in denen Fox Mulder direkt Mitglied ist.

40.10 Informationen über die Active-Directory-Struktur

Informationen über die Active-Directory-Struktur (z.B. Liste der Domänencontroller) gewinnt man über die .NET-Klassenbibliothek oder das ADPowerShell-Modul.

Informationen über die Domäne durch die .NET-Klassenbibliothek

System.DirectoryServices.ActiveDirectory

Zusätzlich zu dem Namensraum `System.DirectoryServices`, der allgemeine Klassen zur Verzeichnisdienstprogrammierung enthält, gibt es in .NET seit Version 2.0 den Unternamensraum `System.DirectoryServices.ActiveDirectory` (alias Active Directory Management Objects – ADMO). Dieser Namensraum implementiert einige Active-Directory-spezifische Funktionen, die nicht auf andere Verzeichnisdienste anwendbar sind.

Insbesondere bietet dieser Namensraum Klassen zur Verwaltung der Gesamtstruktur eines Active Directory, beispielsweise `Forest`, `Domain`, `ActiveDirectoryPartition`, `DomainController`, `GlobalCatalog` und `ActiveDirectorySubnet`. Auch einige spezielle Klassen für den Active Direc-

tory Application Mode (ADAM), eine funktionsreduzierte Version des Active Directorys zum Einsatz als Datenspeicher für eigene Anwendungen, werden mit Klassen wie `ADAMInstanceCollection` und `ADAMInstance` unterstützt.

Beispiel 1: Informationen über die Domäne und den Domänenwald

Das Beispiel liefert Informationen über die Domäne, zu welcher der aktuelle Computer gehört, und über die Gesamtstruktur (Forest), zu der diese Domäne gehört.

Listing 40.26 Informationen über die Domäne und den Forest [ADS_Domain_Info.ps1]

```
# Aktuelle Domain ermitteln
$d =
[System.DirectoryServices.ActiveDirectory.Domain]::GetCurrentDomain
();

# Informationen über aktuelle Domäne
"Name: " + $d.Name
"Domain Mode: " + $d.DomainMode
"Inhaber der InfrastructureRole: " +
$d.InfrastructureRoleOwner.Name
"Inhaber der PdcRole: " + $d.PdcRoleOwner.Name

# Informationen über Forest der aktuellen Domäne
$f = $d.Forest;
"Name der Gesamtstruktur: " + $f.Name
"Modus der Gesamtstruktur: " + $f.ForestMode
```

Beispiel 2: Liste der Domänencontroller und ihrer Rollen

Im zweiten Beispiel werden alle Domänencontroller (und deren Rollen) aus einer speziellen Domäne aufgelistet.

Listing 40.27 Informationen über die Domänencontroller und ihre Rollen [ADS_Domaincontroller_Info.ps1]

```
# Aktuelle Domain ermitteln
$d =
[System.DirectoryServices.ActiveDirectory.Domain]::GetCurrentDomain
()
$DCs = $d.DomainControllers
# Schleife über alle Domänencontroller
foreach ($DC in $DCs)
{
   "Name: " + $DC.Name
   "IP: " + $DC.IPAddress.ToString()
   "Zeit: " + $DC.CurrentTime.ToString()
    "Rollen:"
   # Schleife über alle Rollen des DC
    foreach ($R in $DC.Roles)
    {
     "- " + $R.ToString()
    }
}
```

Informationen über die Domäne durch das Modul ADPowerShell

Domänen-informationen

Folgende Commandlets im PowerShell-AD-Modul liefern Daten über die Domäne:

- `Get-ADDomain`: liefert Daten über die Domäne.
- `Get-ADDomainController`: liefert Daten über die Domänencontroller.
- `Get-ADForest`: liefert Informationen über die AD-Gesamtstruktur.
- `Get-ADOptionalFeature`: liefert eine Liste der optionalen Features des Active Directory.
- `Get-ADRootDSE`: liefert die Wurzel des Active Directory.



***Abbildung 40.31**: Ausgabe von Get-ADDomain und Get-ADDomainController in der Beispieldomäne FBI.org*

41 Gruppenrichtlinien

Das PowerShell-Modul „Group Policy“ (in Windows Server 2008 R2 und optional mit RSAT in Windows 7) bietet Commandlets, die den Funktionen der Gruppenrichtlinienverwaltungskonsole (GPMC) bzw. der zugehörigen COM-Komponente „GPM“ entsprechen.

GPMC

```
PS C:\Users\HS> (get-command) | where { $_.ModuleName -eq "GroupPolicy" } | ft name, modulename, pssnapin

Name                                ModuleName                          PSSnapIn
----                                ----------                          --------
Backup-GPO                          grouppolicy
Copy-GPO                            grouppolicy
Get-GPInheritance                   grouppolicy
Get-GPO                             grouppolicy
Get-GPOReport                       grouppolicy
Get-GPPermissions                   grouppolicy
Get-GPPrefRegistryValue             grouppolicy
Get-GPRegistryValue                 grouppolicy
Get-GPResultantSetOfPolicy          grouppolicy
Get-GPStarterGPO                    grouppolicy
Import-GPO                          grouppolicy
New-GPLink                          grouppolicy
New-GPO                             grouppolicy
New-GPStarterGPO                    grouppolicy
Remove-GPLink                       grouppolicy
Remove-GPO                          grouppolicy
Remove-GPPrefRegistryValue          grouppolicy
Remove-GPRegistryValue              grouppolicy
Rename-GPO                          grouppolicy
Restore-GPO                         grouppolicy
Set-GPInheritance                   grouppolicy
Set-GPLink                          grouppolicy
Set-GPPermissions                   grouppolicy
Set-GPPrefRegistryValue             grouppolicy
Set-GPRegistryValue                 grouppolicy
```

Abbildung 41.1 *Übersicht über die Commandlets im Group-Policy-Modul*



Anders als bei der „GPM“-COM-Komponente kann man mit den PowerShell-Commandlets auch Einstellungen innerhalb der Gruppenrichtlinien skriptbasiert erstellen.

41.1 Verwaltung der Gruppenrichtlinien

Commandlets

Folgende Commandlets dienen der Verwaltung der Gruppenrichtlinien:

- `Get-GPO`: listet alle Gruppenrichtlinien bzw. Zugriff auf eine Gruppenrichtlinie.
- `New-GPO`: erzeugt eine neue, leere Gruppenrichtlinie.
- `Remove-GPO`: entfernt eine Gruppenrichtlinie sowohl aus dem Active Directory als auch aus dem System Volume Folder (SysVol). Anders als bei der GPM-COM-Komponente werden dadurch auch alle Verknüpfungen entfernt.
- `Rename-GPO`: Umbenennen einer Gruppenrichtlinie.
- `Copy-GPO`: Kopieren einer Gruppenrichtlinie (auch zwischen Domänen in einer AD-Gesamtstruktur). Beim Kopieren zwischen Domänen kann man Einstellungen abändern lassen.

- `New-GPStarterGPO`: Erstellen einer neuen, auf andere Domänen transferierbare Gruppenrichtlinienvorlage („Starter GPO“, vgl. Gruppenrichtlinienmanagementkonsole ab Version 2.0).
- `Get-GPStarterGPO`: Auflisten der Gruppenrichtlinienvorlagen.

Die Datenobjekte sind vom Typ `Microsoft.GroupPolicy.GPO` (siehe Bildschirmabbildung).

Abbildung 41.2
Datenobjekte, die Get-GPO liefert

```
PS C:\Users\W8> Get-GPO -all | gm

   TypeName: Microsoft.GroupPolicy.Gpo

Name                MemberType Definition
----                ---------- ----------
Backup              Method     Microsoft.GroupPolicy.GpoBackup Backup(string directory, string comment), Micros...
CopyTo              Method     Microsoft.GroupPolicy.Gpo CopyTo(Microsoft.GroupPolicy.GPDomain targetDomain, st...
Delete              Method     System.Void Delete()
Equals              Method     bool Equals(Microsoft.GroupPolicy.Gpo other), bool Equals(System.Object obj)
GenerateReport      Method     string GenerateReport(Microsoft.GroupPolicy.ReportType reportType), string Gener...
GenerateReportToFile Method    System.Void GenerateReportToFile(Microsoft.GroupPolicy.ReportType reportType, st...
GetHashCode         Method     int GetHashCode()
GetSecurityInfo     Method     Microsoft.GroupPolicy.GPPermissionCollection GetSecurityInfo()
GetType             Method     type GetType()
Import              Method     Microsoft.GroupPolicy.Gpo Import(Microsoft.GroupPolicy.GpoBackup gpoBackup), Mic...
IsAclConsistent     Method     bool IsAclConsistent()
MakeAclConsistent   Method     System.Void MakeAclConsistent()
SetSecurityInfo     Method     System.Void SetSecurityInfo(Microsoft.GroupPolicy.GPPermissionCollection collect...
ToString            Method     string ToString()
Computer            Property   Microsoft.GroupPolicy.ComputerConfiguration Computer {get;}
CreationTime        Property   System.DateTime CreationTime {get;}
Description         Property   System.String Description {get;set;}
DisplayName         Property   System.String DisplayName {get;set;}
DomainName          Property   System.String DomainName {get;}
GpoStatus           Property   Microsoft.GroupPolicy.GpoStatus GpoStatus {get;set;}
Id                  Property   System.Guid Id {get;}
ModificationTime    Property   System.DateTime ModificationTime {get;}
Owner               Property   System.String Owner {get;}
Path                Property   System.String Path {get;}
User                Property   Microsoft.GroupPolicy.UserConfiguration User {get;}
WmiFilter           Property   Microsoft.GroupPolicy.WmiFilter WmiFilter {get;set;}
```

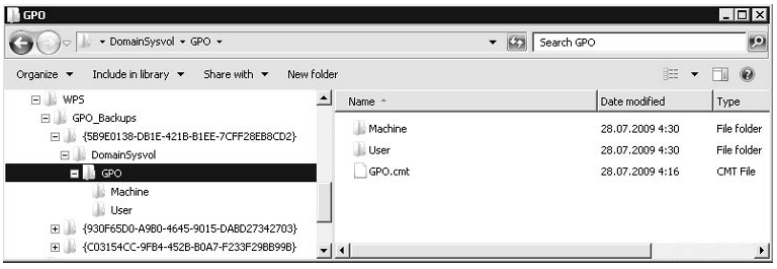
Datensicherung und Wiederherstellung

Zur Datensicherung und Wiederherstellung von Gruppenrichtlinien gibt es `Backup-GPO`, `Restore-GPO` und `Import-GPO`.

`Backup-GPO` erstellt eine Sicherung von einer oder mehreren Gruppenrichtlinien. Die Angabe einer Gruppenrichtlinie erfolgt durch `-Name` oder `-GUID`. Der Parameter `-All` liefert alle Gruppenrichtlinien. Man kann auch GPO-Objekte sichern, die in der Pipeline sind, z.B.:

```
# Datensicherung für alle bestehenden FBI-Gruppenrichtlinien
Get-GPO -all | where { $_.displayname -like "*FBI*" } | Backup-GPO
-Path "c:\wps\GPO_backups"
```

Abbildung 41.3
Ergebnis der Sicherung von drei Gruppenrichtlinien im Dateisystem



Die Wiederherstellung mit Restore-GPO erfolgt anhand des Namens, der GUID oder durch `-all` für alle gesicherten Gruppenrichtlinien.

```
Restore-GPO "GP FBI" -Path "c:\wps\GPO_backups"
Restore-GPO -all -Path "c:\wps\GPO_backups"
```

Import-GPO erlaubt durch `-TargetName` die Wiederherstellung unter einem anderen Namen.

41.2 Verknüpfung der Gruppenrichtlinien

GPLink

Zur Verknüpfung der Gruppenrichtlinien mit Containern im Active Directory stehen zur Verfügung:

- `New-GPLink`: verknüpft eine Gruppenrichtlinie mit einem Container (Site, Domäne oder Organisationseinheit).
- `Remove-GPLink`: entfernt eine Verknüpfung zwischen Gruppenrichtlinie und Container, behält aber die Gruppenrichtlinie selbst.
- `Set-GPLink`: setzt die Eigenschaften Enabled, Enforced und Order für eine Verknüpfung.

Beispiel

Das folgende Skript legt drei leere Gruppenrichtlinien an und verlinkt diese mit zwei Organisationseinheiten. (Vorher wurden die Gruppenrichtlinien gelöscht, falls sie schon existierten.)

Listing 41.1 Beispiel zum Anlegen und Verlinken von Gruppenrichtlinien [WPS2_GP_CreateAndLinkGPO.ps1]

```
Import-Module grouppolicy

Remove-GPO "GP for FBI Agents"
Remove-GPO "GP FBI"
Remove-GPO "GP for FBI Directors"

New-GPO -name "GP FBI" -Comment "Standard Policy for all FBI
Employees"
New-GPO -name "GP for FBI Directors" -Comment "Standard Policy for
all FBI Directors"
New-GPO -name "GP for FBI Agents" -Comment "Standard Policy for all
FBI Agents"

New-GPLink -name  "GP FBI" -target "dc=org" -Linkenabled Yes
New-GPLink -name  "GP for FBI Agents" -target
"ou=agents,dc=fbi,dc=org" -Linkenabled Yes
New-GPLink -name  "GP for FBI Directors" -target
"ou=directors,dc=fbi,dc=org" -Linkenabled Yes
```

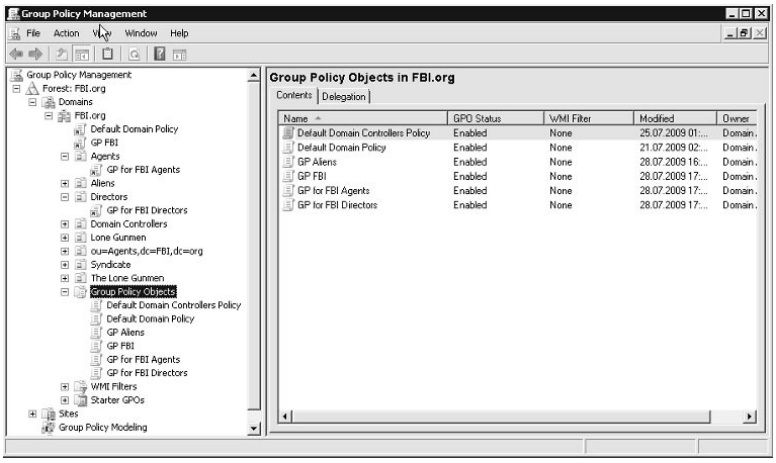


Abbildung 41.4 *Zustand der Gruppenrichtlinien nach Ausführung des obigen Skripts*

41.3 Berichte

`Get-GPOReport` erstellt einen Bericht über eine Gruppenrichtlinie inklusive ihrer Einstellungen und aller Verknüpfungen. Das Ausgabeformat ist wahlweise HTML oder XML.

```
Get-GPOReport "GP FBI"  -Reporttype html >c:\wps\go_report.htm
```

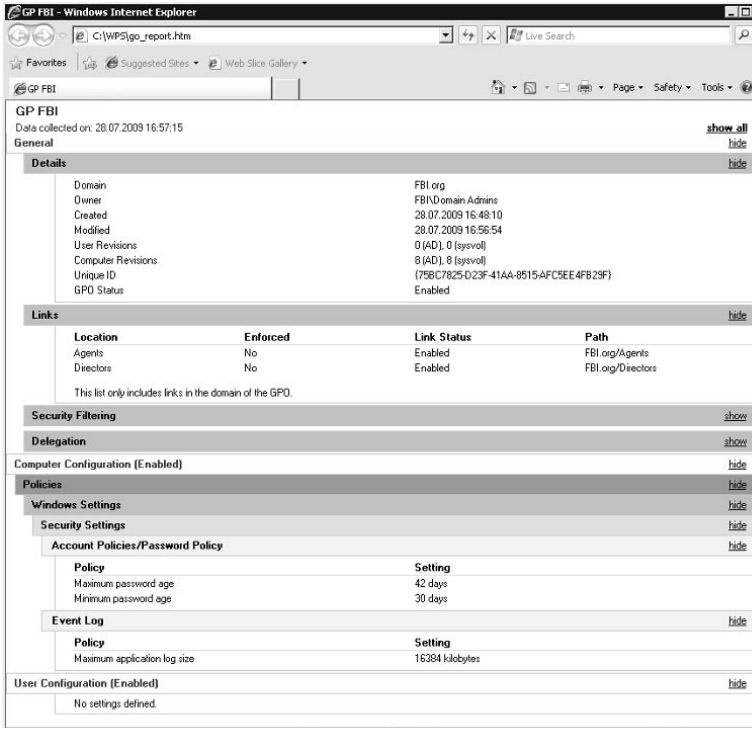


Abbildung 41.5 *Gruppenrichtlinienbericht in HTML-Form*

Richtlinien-ergebnisbericht

Ein Richtlinienergebnisbericht (engl. Resultant Set Of Policy) zeigt an, welche Gruppenrichtlinien auf einen Computer oder einen konkreten Benutzer eines Computers bei der Anmeldung an diesem Computer wirken.

Diesen Bericht erstellt das Commandlet `Get-GPResultantSetOfPolicy` in HTML- oder XML-Form.

Der folgende Befehl erstellt den Bericht für den Benutzer „Fox Mulder“ auf dem Computer „F171“:

```
Get-GPResultantSetOfPolicy -user "FoxMulder" -computer "F171" -
ReportType HTML -path c:\wps\rsop.htm
```

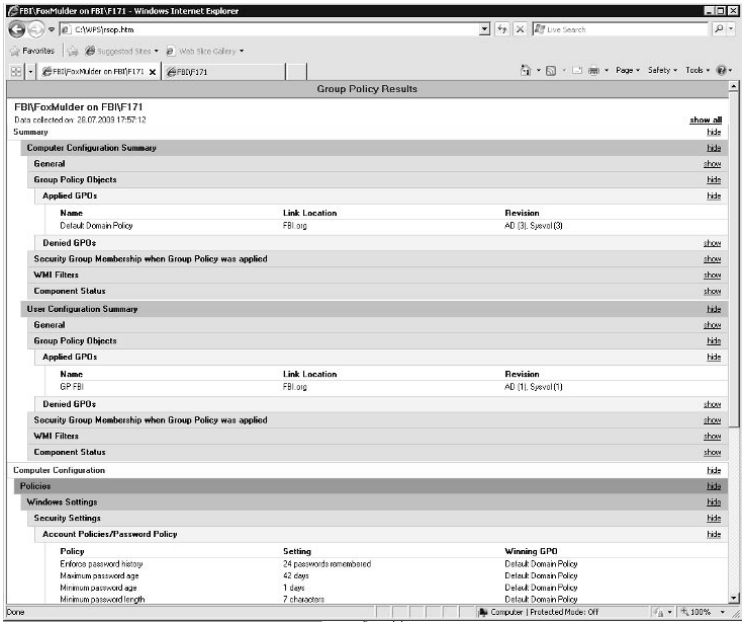


Abbildung 41.6 *Richtlinienergebnisbericht in HTML-Form*

41.4 Gruppenrichtlinienvererbung

Get-GPInheritance

Auf einen Container wirken Gruppenrichtlinien, die direkt verknüpft sind oder die geerbt werden. Mit `Get-GPInheritance` kann eine Liste aller wirkenden Gruppenrichtlinien für einen Container ermittelt werden.

```
Get-GPInheritance -target "ou=agents,dc=fbi,dc=org"
```

Die Bildschirmabbildung zeigt, dass „GP for FBI Agents“ direkt verknüpft ist, während „GP FBI“ und „Default Domain Policy“ geerbt werden.

```
PS C:\Users\MS> Get-GPInheritance -target "ou=agents,dc=fbi,dc=org"

Name                    : agents
ContainerType           : OU
Path                    : ou=agents,dc=fbi,dc=org
GpoInheritanceBlocked   : No
GpoLinks                : {GP for FBI Agents}
InheritedGpoLinks       : {GP for FBI Agents, Default Domain Policy, GP FBI}
```

Abbildung 41.7 *Ergebnis der Ausführung von Get-GPInheritance*



Mit `Set-GPInheritance` kann man steuern, ob ein Container Gruppenrichtlinien von übergeordneten Containern erben soll.

Der folgende Befehle verhindern, dass die „Aliens“ die Richtlinien „GP FBI“ und „Default Domain Policy“ erben:

```
Set-GPInheritance -target "ou=aliens,dc=fbi,dc=org" -isblocked yes
```

Abbildung 41.8
Veränderte Vererbung durch Einsatz von Set-GP-Inheritance

```
PS C:\Users\HS> Get-GPInheritance -target "ou=aliens,dc=fbi,dc=org"


Name                   : aliens
ContainerType          : OU
Path                   : ou=aliens,dc=fbi,dc=org
GpoInheritanceBlocked  : No
GpoLinks               : {GP for Aliens}
InheritedGpoLinks      : {GP for Aliens, Default Domain Policy, GP FBI}


PS C:\Users\HS> Set-GPInheritance -target "ou=aliens,dc=fbi,dc=org" -isblocked yes


Name                   : aliens
ContainerType          : OU
Path                   : ou=aliens,dc=fbi,dc=org
GpoInheritanceBlocked  : Yes
GpoLinks               : {GP for Aliens}
InheritedGpoLinks      : {GP for Aliens}


PS C:\Users\HS> Get-GPInheritance -target "ou=aliens,dc=fbi,dc=org"


Name                   : aliens
ContainerType          : OU
Path                   : ou=aliens,dc=fbi,dc=org
GpoInheritanceBlocked  : Yes
GpoLinks               : {GP for Aliens}
InheritedGpoLinks      : {GP for Aliens}


PS C:\Users\HS> _
```

41.5 Weitere Möglichkeiten

Mit den folgenden Commandlets kann man Einstellungen in den Gruppenrichtlinien vornehmen, lesen oder löschen:

- `Set-GPRegistryValue`
- `Set-GPPrefRegistryValue`
- `Get-GPPrefRegistryValue`
- `Get-GPRegistryValue`
- `Remove-GPPrefRegistryValue`
- `Remove-GPRegistryValue`

Außerdem gibt es noch Commandlets zur Verwaltung der Zugriffsrechte auf Gruppenrichtlinien:

- `Set-GPPermissions`
- `Get-GPPermissions`

Eine Besprechung ist hier aufgrund der verlegerischen Seitenrestriktionen leider nicht möglich!

42 Internet Information Server (IIS)

IIS-Verwaltung

Das Modul „WebAdministration“ (in Windows Server 2008 R2 und Windows 7 nach Installieren von RSAT) dient der Verwaltung des in Windows Client und Windows Server integrierten Webservers „Internet Information Services“ (IIS). Mit dem Modul lässt sich aber nur der IIS 7 verwalten, der ab Windows Vista bzw. Windows Server 2008 in Windows enthalten ist. Auch der in Windows Server 2008 R2 enthaltene IIS 7.5 lässt sich damit verwalten.

42.1 Überblick

Das Modul „WebAdministration“ enthält einen gleichnamigen PowerShell-Navigationsprovider sowie zahlreiche Commandlets:

- `Add-WebConfiguration`
- `Add-WebConfigurationLock`
- `Add-WebConfigurationProperty`
- `Backup-WebConfiguration`
- `Clear-WebConfiguration`
- `Clear-WebRequestTracingSettings`
- `ConvertTo-WebApplication`
- `Disable-WebGlobalModule`
- `Disable-WebRequestTracing`
- `Enable-WebGlobalModule`
- `Enable-WebRequestTracing`
- `Get-WebAppDomain`
- `Get-WebApplication`
- `Get-WebAppPoolState`
- `Get-WebBinding`
- `Get-WebConfigFile`
- `Get-WebConfiguration`
- `Get-WebConfigurationBackup`
- `Get-WebConfigurationLocation`
- `Get-WebConfigurationLock`
- `Get-WebConfigurationProperty`

- Get-WebFilePath
- Get-WebGlobalModule
- Get-WebHandler
- Get-WebItemState
- Get-WebManagedModule
- Get-WebRequest
- Get-Website
- Get-WebsiteState
- Get-WebURL
- Get-WebVirtualDirectory
- New-WebApplication
- New-WebAppPool
- New-WebBinding
- New-WebFtpSite
- New-WebGlobalModule
- New-WebHandler
- New-WebManagedModule
- New-Website
- New-WebVirtualDirectory
- Remove-WebApplication
- Remove-WebAppPool
- Remove-WebBinding
- Remove-WebConfigurationBackup
- Remove-WebConfigurationLocation
- Remove-WebConfigurationLock
- Remove-WebConfigurationProperty
- Remove-WebGlobalModule
- Remove-WebHandler
- Remove-WebManagedModule
- Remove-Website
- Remove-WebVirtualDirectory
- Rename-WebConfigurationLocation
- Restart-WebAppPool
- Restart-WebItem
- Restore-WebConfiguration

- Select-WebConfiguration
- Set-WebBinding
- Set-WebConfiguration
- Set-WebConfigurationProperty
- Set-WebGlobalModule
- Set-WebHandler
- Set-WebManagedModule
- Start-WebAppPool
- Start-WebCommitDelay
- Start-WebItem
- Start-Website
- Stop-WebAppPool
- Stop-WebCommitDelay
- Stop-WebItem
- Stop-Website

42.2 Navigationsprovider

IIS

Das Modul „WebAdministration“ legt beim Import für den Navigationsprovider „WebAdministration“ ein „Laufwerk“ „IIS:“ an. Unterhalb des Wurzelordners gibt es die Unterordner: „AppPools“, „Sites“ und „SslBinding“. Sites enthält die HTTP-Websites (siehe *Abbildung 42.1*).

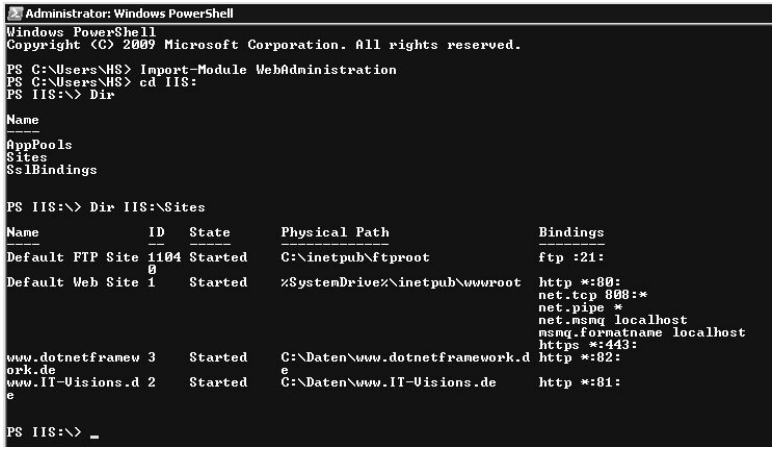
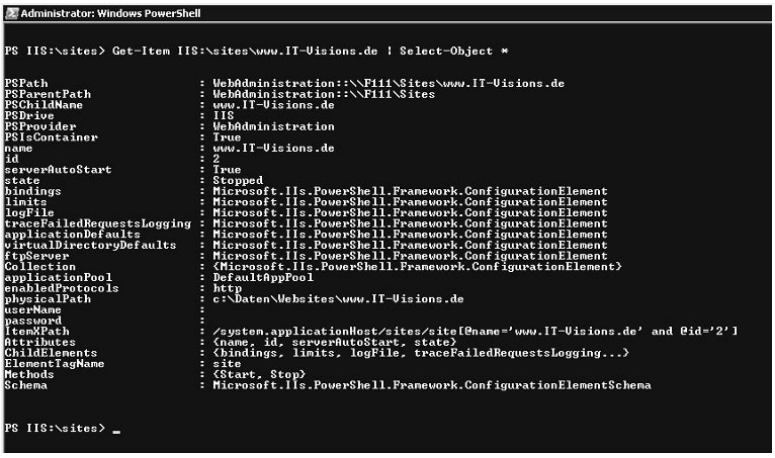


Abbildung 42.1 *Auflisten der vorhandenen Websites*

Abbildung 42.2
*Mit Get-Item | Select-Object * sieht man Details über eine Website.*

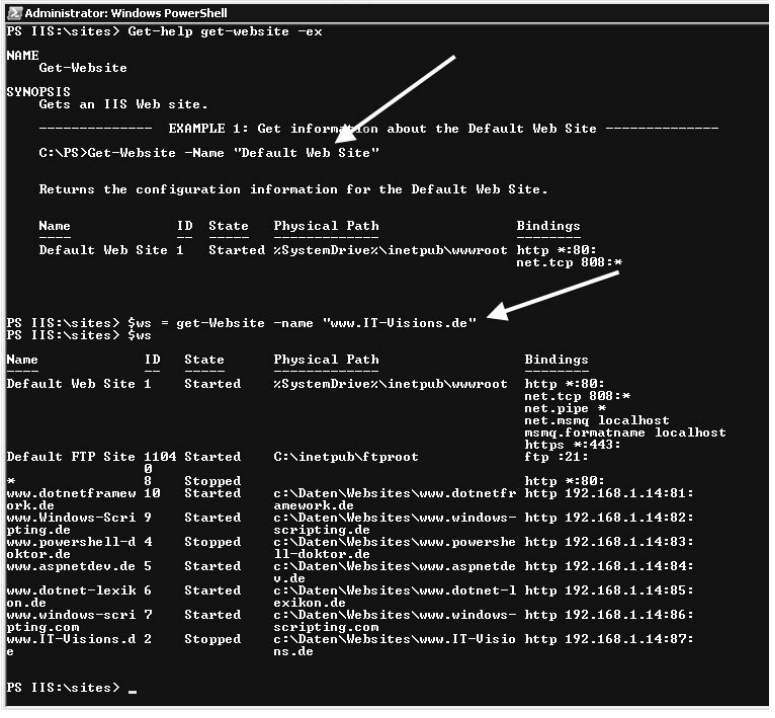


Get-Website

Alternativ kann man zum Auflisten der Inhalte die Commandlets `Get-Website`, `Get-WebvirtualDirectory` und `Get-WebApplication` verwenden.

Achtung: Bei `Get-Website` ist zumindest in der zum Redaktionsschluss dieses Buchs vorliegenden RC-Version ein Fehler. `Get-Website` ignoriert den Parameter `-Name` und liefert immer alle Websites, obwohl die Hilfedatei genau ein Beispiel zeigt, wie man mit `-Name` filtern könnte.

Abbildung 42.3
Get-Websites ignoriert den Parameter „-name“.



Um dies zu umgehen, muss man `Where-Object` benutzen:

```
$ws = Get-Website  | where { $_.name -eq "www.IT-Visions.de" }
```

Grundsätzlich kann man auch `Get-Item` verwenden:

```
$ws2 = Get-Item "iiS:\sites\ www.IT-Visions.de".
```

Der Unterschied ist nur, dass im ersten Fall die PowerShell den Pipeline-Inhalt als Instanzen „Microsoft.IIs.PowerShell.Framework.ConfigurationElement#site“ identifiziert und im zweiten Fall die PowerShell dort lediglich „System.Object“ erkennt, obwohl die Attribute, die Get-Member anzeigt, in beiden Fällen gleich sind.

42.3 Anlegen von Websites

Zum Anlegen eines virtuellen Webservers („Website“) verwendet man `New-Website`, z.B.:

New-Website

```
New-Website -Name "www.PowerShell-Doktor.de" -PhysicalPath "c:\
Daten\www.PowerShell-doktor.de" -Port 83
```

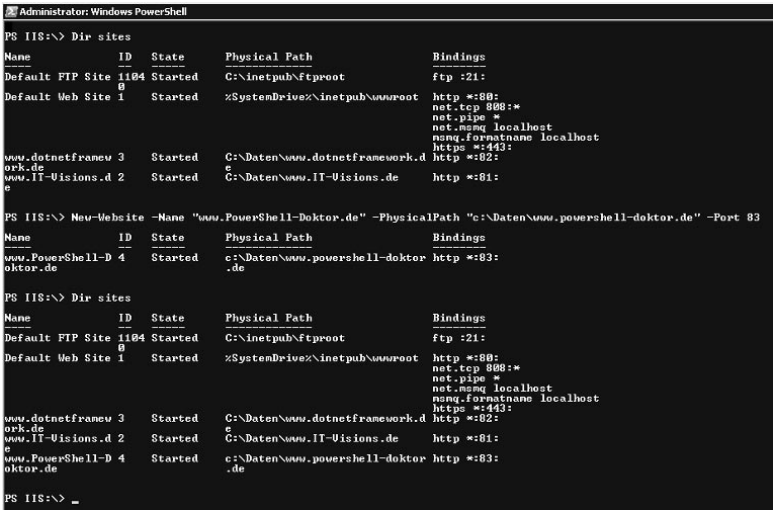


Abbildung 42.4
Vor und nach dem Anlegen einer Website

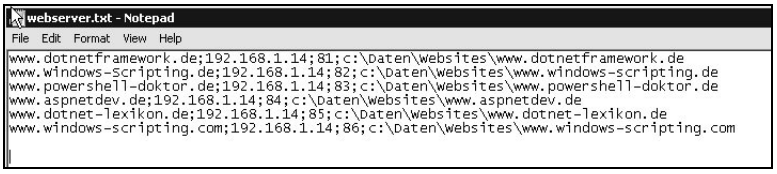
Das angegebene Verzeichnis im Dateisystem muss vorher existieren!



42.4 Massenanlegen von Websites

Es sei folgende Textdatei gegeben, aus der virtuelle Webserver („Websites“) erzeugt werden sollen.

Abbildung 42.5
Datei webserver.txt



Ohne das WebAdministration-Modul müsste man das folgende Skript starten, das WMI verwendet.

Listing 42.1
Massenanlegen von Websites mit PowerShell ohne das WebAdministration-Modul [Einsatzgebiete\IIS\IIS_CreateSites.ps1]

```
# ======================
# IIS Script: Websites anlegen ohne das WebAdministration-Modul
# (C) Dr. Holger Schwichtenberg
# ======================

# === Get WMI Object with DCOM encryption
Function Get-WMIObjectEx($Namespace, $Path)
{
#Write-Host $Namespace $Path
$connection = New-Object System.Management.ConnectionOptions
$connection.Authentication =
[System.Management.AuthenticationLevel]::PacketPrivacy
$scope = New-Object System.Management.ManagementScope($Namespace,
$connection)
$path = New-Object System.Management.ManagementPath($Path)
$GetOptions = New-Object System.Management.ObjectGetOptions
$WMI = New-Object
System.Management.ManagementObject($scope,$path,$GetOptions)
return $WMI
}

# === Get WMI class with DCOM encryption
Function Get-WMIClassEx($Namespace, $Path)
{
Write-Host $Namespace $Path
$connection = New-Object System.Management.ConnectionOptions
$connection.Authentication =
[System.Management.AuthenticationLevel]::PacketPrivacy
$scope = New-Object System.Management.ManagementScope($Namespace,
$connection)
$path = New-Object System.Management.ManagementPath($Path)
```

```
$GetOptions = New-Object System.Management.ObjectGetOptions
return New-Object
System.Management.ManagementClass($scope,$path,$GetOptions)
}

# === Create Site
function New-IISVirtWeb ([string]$Computer, [string]$Name,
[string]$IP, [string]$Port, [string]$Hostname, [string]$RootDir)
{
$Namespace = "\\"  + $Computer + "\root\MicrosoftIISv2"
$Path1 = $Namespace + ":ServerBinding"
$Path2 = $Namespace + ":IIsWebService='W3SVC'"


# Create Binding
$class = Get-WMIClassEx $Namespace $Path1
$binding = $class.CreateInstance()
$binding.IP = $IP
$binding.Port = $Port
$binding.Hostname = $Hostname
[array] $bindings = $binding

# Create Site
$Webservice = Get-WMIObjectEx $Namespace  $Path2
$Website = $Webservice.CreateNewSite($Name, $bindings, $RootDir)

Write-Host "Webserver" $Name "angelegt auf Computer" $Computer "!"
}


# --- Parameters
$InputFile = "H:\demo\WPS\B_IIS\webserver.txt"
$Computer = "F171"

# Read textfile and create a new webserver for each line
Get-Content $InputFile | Foreach-Object {
$a = $_.Split(";")
# Create directory if it does not exist!
mkdir  $a[3] -erroraction silentlycontinue
# Create Websitegm
New-IISVirtWeb $Computer  $a[0]  $a[1] $a[2] "" $a[3]
}
```

Mit dem WebAdministration-Modul ist diese Aufgabe wesentlich kürzer zu erledigen.

Listing 42.2 Massenanlegen von Websites mit PowerShell mit dem WebAdministration-Modul [WPS2_IIS-CreateWebsites_From_CSV.ps1]

```
# --- Parameters
$InputFile = "c:\wps\webserver.txt"
$Computer = "F111"

# --- Read textfile and create a new webserver for each line
Get-Content $InputFile | Foreach-Object {
$a = $_.Split(";")
# Create directory if it does not exist!
mkdir  $a[3] -erroraction silentlycontinue
# Create Website
New-Website -Name $a[0] -IPAddress  $a[1] -port $a[2] -PhysicalPath
$a[3] -force
}
```

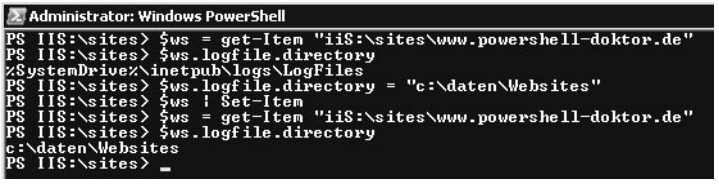
42.5 Ändern von Eigenschaften von Websites

Get-Website

Zum Ändern von Eigenschaften einer Website greift man auf die Website mit `Get-Item` (oder `Get-Website`) zu. Dann kann man auf die Attribute schreibend zugreifen. Nach dem Beschreiben sorgt man mit `Set-Item` für die Speicherung der Änderungen.

Achtung: Anders als der Dateisystemprovider der PowerShell speichert der WebAdministration-Provider erst nach einem expliziten Speichervorgang!

Abbildung 42.6 Verändern des Standorts für die Protokolldaten





Ohne den Neuabruf der Daten mit `Get-Item` nach dem `Set-Item` hätte man in dem obigen Beispiel keinen Beweis, dass die Änderung wirklich gespeichert wurde.

42.6 Anwendungspool anlegen

New-WebAppPool

Sofern man keinen Anwendungspool angibt, landet die neue Website im Standardpool. Alternativ kann man vorher mit `New-WebAppPool` einen Pool anlegen und bei New-Website den Namen des Pools im Parameter `-ApplicationPool` angeben:

```
$pool = New-WebAppPool -name "PowerShell-Doktor Pool"
$site = New-Website -Name "www.PowerShell-Doktor.de" -PhysicalPath
"c:\Daten\www.PowerShell-doktor.de" -Port 83 -ApplicationPool
$pool.name
```

Außerdem kann man die Identität des Anwendungspools setzen:

```
Set-ItemProperty "iis:\apppools\PowerShell-Doktor Pool" -name
processModel -value @{userName="PSDoktor Pool
User";password="Very!Secret!09";identitytype=3}
```

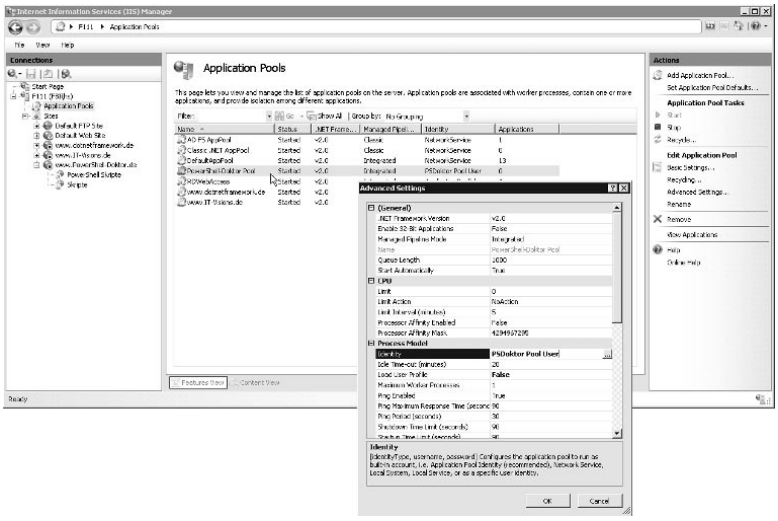


Abbildung 42.7
Nach dem Anlegen des Anwendungspools und dem Zuweisen der Identität

Die PowerShell-Commandlets prüfen nicht, ob die angegebene Identität existiert.



42.7 Virtuelle Verzeichnisse und IIS-Anwendungen

Das Anlegen eines virtuellen Verzeichnisses ist ebenfalls sehr einfach:

New-WebVirtual-Directory

```
New-WebVirtualDirectory -Site "www.PowerShell-doktor.de" -
PhysicalPath "c:\Daten\WPSSkripte" -Name "Skripte"
```

Aus einem virtuellen Verzeichnis kann man eine IIS-Anwendung erzeugen:

```
ConvertTo-WebApplication -PSPath "IIS:\sites\www.IT-Visions.de\
Skripte"
```

New-Web-Application

Oder man kann direkt eine IIS-Anwendung anlegen:

```
New-WebApplication -Site "www.PowerShell-doktor.de" -Name
"PowerShell Skripte" -PhysicalPath "C:\Daten\Skripte"
```

42.8 Website-Zustand ändern

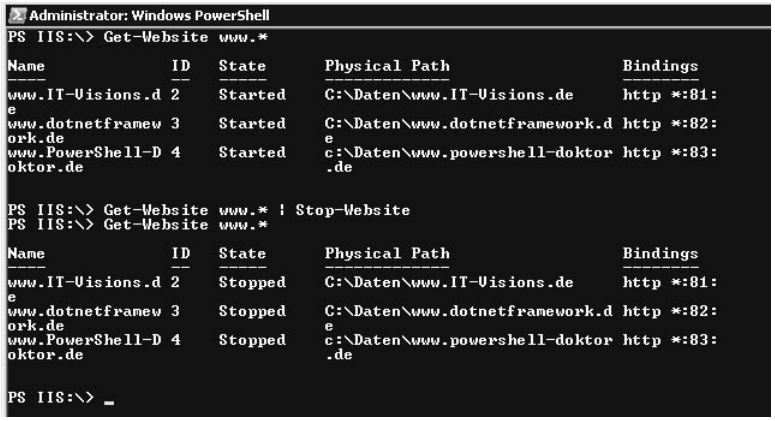
Starten und Stoppen

Mit `Get-WebitemState` kann man abfragen, ob eine Website läuft oder gestoppt ist:

```
Get-WebitemState "IIS:\sites\www.PowerShell-doktor.de"
```

Mit `Start-Website` und `Stop-Website` kann man den Status einer IIS-Website ändern (siehe Beispiel in der Bildschirmabbildung).

Abbildung 42.8 Anhalten einer Website, deren Name mit „www.“ beginnt



42.9 Anwendungspools starten und stoppen

Get-Webitem-State

Mit `Get-WebitemState` kann man auch den Status von Anwendungspools abfragen:

```
Get-WebitemState "IIS:\Apppools\PowerShell-Doktor Pool"
```

Hier erfolgt die Statusänderung mit `Start-Webitem` und `Stop-Webitem`, z.B.:

```
Stop-Webitem "IIS:\Apppools\PowerShell-Doktor Pool"
Start-Webitem "IIS:\Apppools\PowerShell-Doktor Pool"
```

Will man einen laufenden Pool einfach neu starten, kann man auch `Restart-Webitem` verwenden:

```
Restart-Webitem "IIS:\Apppools\PowerShell-Doktor Pool"
```

Die Commandlets mit „Webitem“ im Namen lassen sich auch auf Websites anwenden.



42.10 Löschen von Websites

Remove

Zum Löschen von Einträgen im IIS gibt es diverse Commandlets, z.B. `Remove-Website`, `Remove-WebApplication`, `Remove-WebVirtualDirectory` und `Remove-WebAppPool`.

Achtung: Die Commandlets arbeiten im Standard ohne Nachfrage. Der Befehl „Remove-WebSite www*“ löscht alle Websites, deren Name mit „www“ beginnt.

43 Microsoft Exchange Server

Microsoft Exchange Server 2007 war das erste Microsoft-Server-Produkt mit einer kompletten PowerShell-Unterstützung.

Nach dem Start der auf PowerShell basierenden Exchange Management Shell erhält man mit dem Befehl

```
Get-ExCommand
```

eine Liste der Exchange-Server-spezifischen Commandlets.

43.1 Daten abrufen

Datenabruf

Eine Liste aller Postfächer erhält man durch:

```
Get-Mailbox
```

Die Liste der Datenbanken liefert:

```
Get-Mailboxdatabase
```

Und die Speichergruppen bekommt man mit:

```
Get-Storagegroup
```

Die Funktionsfähigkeit eines Exchange Servers kann man testen mit:

```
Test-ServiceHealth
```

43.2 Postfächer verwalten

Postfach-verwaltung

Eine Speichergruppe legt man an mit:

```
New-Storagegroup "Autorenspeichergruppe" -server "E12"
```

Eine Datenbank für Postfächer erstellt man mit:

```
New-Mailboxdatabase "Autorenpostfachdatenbank" -storagegroup
"Autorenspeichergruppe"
```

Zum Erstellen eines Postfachs kann man folgenden Befehl verwenden:

```
New-Mailbox -alias "HSchwichtenberg" -name HolgerSchwichtenberg -
userprincipalname HS@IT-Visions.de -database "E12\First Storage
Group\Mailbox Database" -org users
```

Wenn der Benutzer im Active Directory schon existiert, ist der Befehl kürzer:

```
Enable-Mailbox hs@IT-Visions.de -database "E12\First Storage Group\
Mailbox Database"
```

Nach dem Anlegen kann man mit `Get-Mailbox` bzw. `Set-Mailbox` auf die Eigenschaften des Postfachs zugreifen. Das nachträgliche Ergänzen einer E-Mail-Adresse funktioniert durch Neusetzen der Eigenschaft `EMailAddresses` unter Berücksichtigung der bisherigen Adressen:

```
Set-Mailbox HS@IT-Visions.de -EmailAddresses ((Get-Mailbox hs@IT-
Visions.de).EmailAddresses + "HSchwichtenberg@IT-Visions.de ")
```

Das Postfach kann man zu einer Verteilerliste hinzufügen:

```
Add-DistributionGroupMember Autoren -Member "hs@IT-Visions.de"
```

Das Postfach kann man in eine andere Datenbank verlagern:

```
Move-Mailbox hs@IT-Visions.de -targetdatabase
"Autorenpostfachdatenbank"
```

Oder man begrenzt den Speicherplatz:

```
Get-Mailbox hs@IT-Visions.de | Set-Mailbox -
UseDatabaseQuotaDefaults:$false -ProhibitSendReceiveQuota 100MB -
ProhibitSendQuota 90MB -IssueWarningQuota 80MB
```

Begrenzen kann man auch die Größe für eingehende E-Mails für eine Verteilerliste:

```
Set-DistributionGroup Autoren -MaxReceiveSize 5000KB
```

Auch zum Deaktivieren eines Postfachs gibt es ein Commandlet:

```
Disable-Mailbox hs@IT-Visions.de
```

43.3 Öffentliche Ordner verwalten

Öffentliche Ordner

Eine Datenbank für öffentliche Ordner erstellt man mit:

```
New-PublicFolderDatabase "Autorenordnerdatenbank" -storagegroup
"Autorenspeichergruppe "
```

Einen öffentlichen Ordner legt man an mit:

```
New-PublicFolder "\Dokumente" -Path \pubfolders -Server "E12"
```

Rechte auf einen Ordner vergibt man mit:

```
Add-PublicFolderPermission "\ManuScripte" -User hs -AccessRights
"CreateItems"
```

Die Speicherplatzgrenzen für einen öffentlichen Ordner setzt man mit:

```
Set-PublicFolder "\Dokumente" -PostStorageQuota 20MB -MaxItemSize 2MB
```

Weitere Informationen

Weitere PowerShell-Skripte zur Exchange-Verwaltung finden Sie unter [TNET02].

44 Optimierungen und Problemlösungen

Windows 7 und Windows Server 2008 R2 bieten zwei Module, die dabei helfen, das Betriebssystem optimal zu konfigurieren und Probleme zu lösen.

44.1 PowerShell-Modul „TroubleShootingPack“

Windows Troubleshooting Platform

Ein „Troubleshooting Pack“ (Problemlösungspaket) ist ab Windows 7 und Windows Server 2008 R2 eine Möglichkeit für den Windows-Benutzer, Problemunterstützung zu erhalten. Man findet die Problemlöser als Benutzer in der Systemsteuerung. Microsoft spricht auch von der Windows Troubleshooting Platform (WTP).

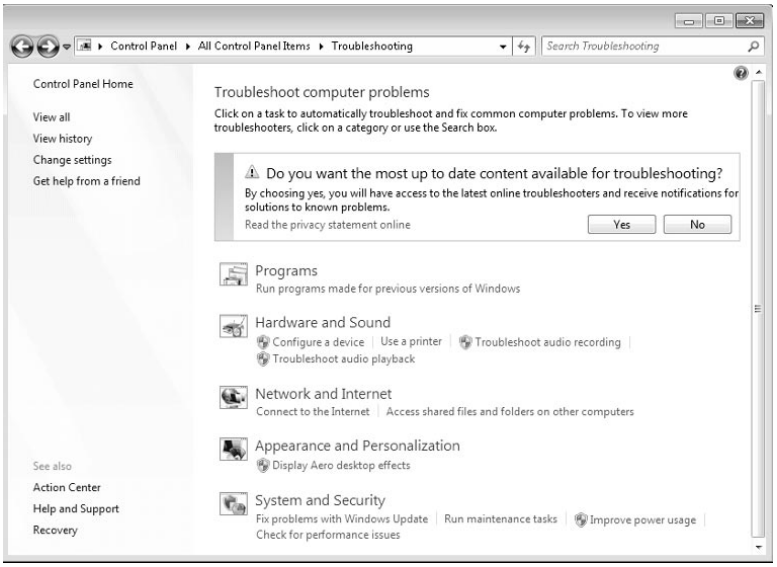


Abbildung 44.1 *Problemlösungsangebote in der Windows 7-Systemsteuerung*

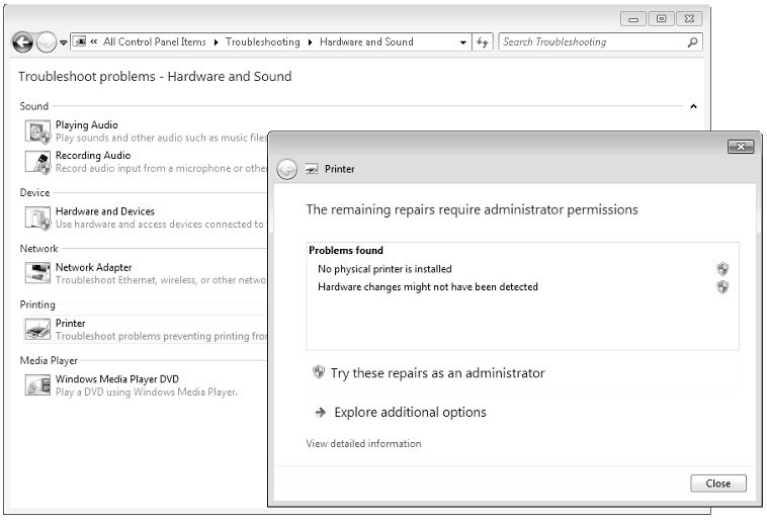


Abbildung 44.2 *Der Problemlöser für den Drucker, der nicht gefunden wurde*

Implementiert sind die Problemlöser in Form von DLLs und PowerShell-Skripten (die zum Teil wiederum aus C#-Programmcode bestehen, der ad hoc kompiliert wird). Die Problemlöser liegen unter *C:\Windows\Diagnostics*.

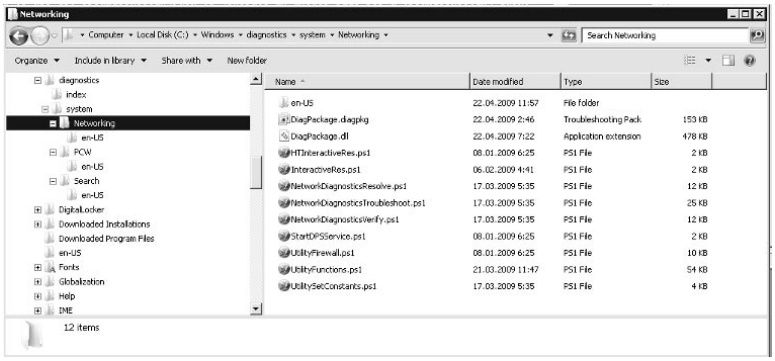


Abbildung 44.3 *Troubleshooting Packs in Windows Server 2008 R2*

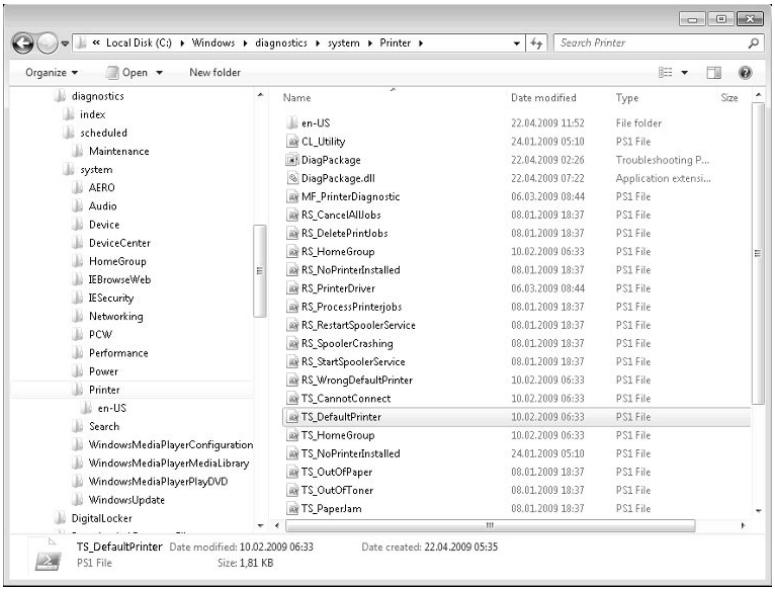


Abbildung 44.4
Troubleshooting Packs in Windows 7

Das Modul „TroubleShootingPack“ in Windows 7 und Windows Server 2008 R2 enthält nur zwei Commandlets:

- `Get-TroubleshootingPack`
- `Invoke-TroubleshootingPack`

Mit dem Commandlet `Get-TroubleshootingPack` kann man Informationen über ein durch den Standort im Dateisystem spezifiziertes Problemlösungspaket abrufen, z.B.:

```
Get-TroubleshootingPack C:\Windows\diagnostics\system\printer
```

Rückgabeobjekt ist eine Instanz der Klasse `Microsoft.Windows.Diagnosis.DiagPack`.

```
PS C:\Windows\system32> Get-TroubleshootingPack C:\Windows\diagnostics\system\printer

Id                    Name                                 Publisher                            Version
--                    ----                                 ---------                            -------
PrinterDiagnostic     Printer                              Microsoft Windows                    1.0

PS C:\Windows\system32> Get-TroubleshootingPack C:\Windows\diagnostics\system\printer | gm


   TypeName: Microsoft.Windows.Diagnosis.DiagPack

Name                MemberType Definition
----                ---------- ----------
Cancel              Method     System.Void Cancel()
CopyResults         Method     System.Void CopyResults(string destinationDirectoryPath)
Diagnose            Method     System.Void Diagnose()
Equals              Method     bool Equals(Microsoft.Windows.Diagnosis.DiagPack other), bool Equals(System.Object obj)
GetHashCode         Method     int GetHashCode()
GetType             Method     type GetType()
Resolve             Method     System.Void Resolve(Microsoft.Windows.Diagnosis.DiagResolution resolution)
ToString            Method     string ToString()
Verify              Method     System.Void Verify()
Description         Property   System.String Description {get;}
ExtensionPoint      Property   System.Xml.XmlDocument ExtensionPoint {get;}
FoundRootCauses     Property   Microsoft.Windows.Diagnosis.DiagRootCause[] FoundRootCauses {get;}
Id                  Property   System.String Id {get;}
Interactions        Property   Microsoft.Windows.Diagnosis.DiagInteraction[] Interactions {get;}
Interactive         Property   System.Boolean Interactive {get;}
MinimumVersion      Property   System.String MinimumVersion {get;}
Name                Property   System.String Name {get;}
Path                Property   System.String Path {get;}
PrivacyLink         Property   System.String PrivacyLink {get;}
Publisher           Property   System.String Publisher {get;}
RequiresElevation   Property   System.Boolean RequiresElevation {get;}
RootCauses          Property   Microsoft.Windows.Diagnosis.DiagRootCause[] RootCauses {get;}
SupportsAmd64       Property   System.Boolean SupportsAmd64 {get;}
SupportsClient      Property   System.Boolean SupportsClient {get;}
SupportsIA64        Property   System.Boolean SupportsIA64 {get;}
SupportsServer      Property   System.Boolean SupportsServer {get;}
SupportsX86         Property   System.Boolean SupportsX86 {get;}
Version             Property   System.String Version {get;}


PS C:\Windows\system32>
```

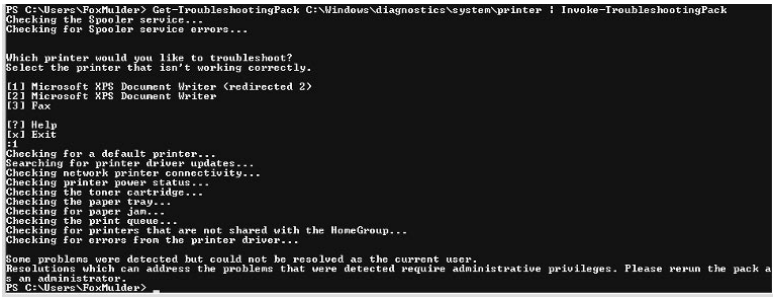
Abbildung 44.5
Aufruf von Get-TroubleshootingPack

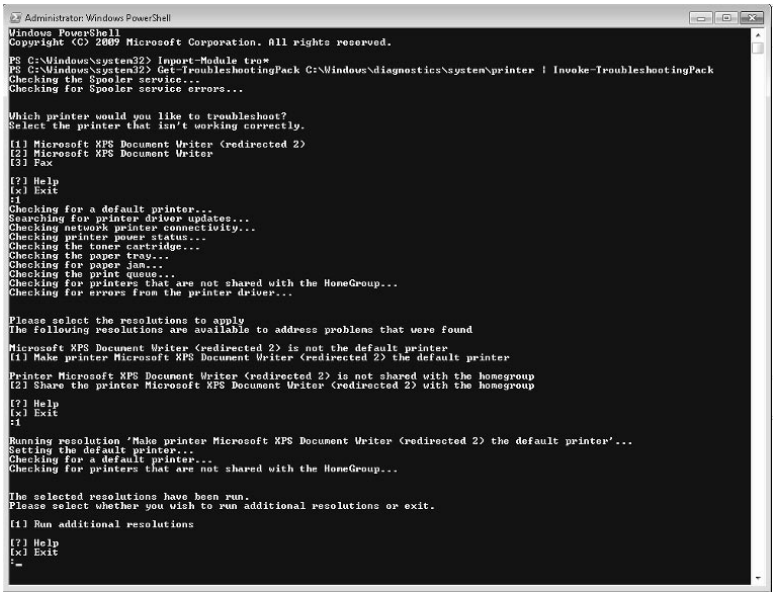
Mit `Invoke-TroubleshootingPack` kann man ein zuvor mit `Get-TroubleshootingPack` geladenes Paket aufrufen, z.B.

```
Get-TroubleshootingPack C:\Windows\diagnostics\system\printer |
Invoke-TroubleshootingPack
```

Die Interaktion ist dann kommandozeilenbasiert (siehe *Abbildung 44.6*).

Abbildung 44.6 *Aufruf des Problemlösers für den Drucker mit Administratorrechten*





44.2 PowerShell-Modul „Best Practices“

BPA-Modelle Ein „Best Practices Analyzer-Modell“ (kurz: „BPA-Modell“) ist in Windows Server 2008 R2 eine Prüfroutine, ob Windows-Funktionen gemäß den Richtlinien von Microsoft eingesetzt werden.

Das Modul umfasst vier Commandlets:

- `Get-BPAModel`: Liste der installierten Best-Practices-Modelle
- `Invoke-BPAModel`: Aufruf eines BPA-Modells
- `Get-BPAResult`: Anzeige der Prüfergebnisse eines BPA-Modells (es wird immer nur der letzte Prüfvorgang für das angegebene Modell gespeichert)
- `Set-BPAResult`: dient dem Ausschluss von Ergebnissen aus der Liste der Prüfergebnisse, die `Get-BPAResult` liefert

`Get-BPAModel` zeigt auf einem Windows Server 2008 R2 mit nahezu allen installierten Rollen und Funktionen (Ausnahme: HyperV und Active Directory Rights Management Services) die in nachstehender Bildschirmabbildung dokumentierten BPA-Modelle.

```
PS C:\Users\HS> Get-BpaModel

Id                                                          LastScanTime
--                                                          ------------
Microsoft/Windows/CertificateServices                       Never
Microsoft/Windows/DirectoryServices                         Never
Microsoft/Windows/DNSServer                                 Never
Microsoft/Windows/TerminalServices                          Never
Microsoft/Windows/WebServer                                 Never
```

Abbildung 44.7
Anzeige der installierten BPA-Modelle

```
PS C:\Users\HS> Invoke-BpaModel Microsoft/Windows/Webserver

ModelId                     Success Detail
-------                     ------- ------
Microsoft/Windows/WebServer True    {InvokeBpaModelOutputDetail}
```

Abbildung 44.8
Aufruf eines BPA-Modells

```
Administrator: Windows PowerShell
PS C:\Users\HS> Get-Bparesult Microsoft/Windows/Webserver


ResultNumber : 1
ModelId      : Microsoft/Windows/WebServer
RuleId       : 1
ResultId     : 1122680488
Severity     : Error
Category     : Security
Title        : Grant a handler execute/script or write permissions, but not both
Problem      : The attribute 'accessPolicy' in the handlers section under path 'MACHINE/WEBROOT/APPHOST' is set to allo
               w both Execute/Script and Write permissions.
Impact       : By allowing both Execute/Script and Write permissions, a handler can run malicious code on the target se
               rver.
Resolution   : Determine if the handler requires both Execute/Script and Write permissions, and revoke the one that is
               not needed.
Compliance   :
Help         : http://go.microsoft.com/fwlink/?LinkId=130708
Excluded     : False

ResultNumber : 2
ModelId      : Microsoft/Windows/WebServer
RuleId       : 1
ResultId     : 1122680488
Severity     : Error
Category     : Security
Title        : Grant a handler execute/script or write permissions, but not both
Problem      : The attribute 'accessPolicy' in the handlers section under path 'MACHINE/WEBROOT/APPHOST' is set to allo
               w both Execute/Script and Write permissions.
Impact       : By allowing both Execute/Script and Write permissions, a handler can run malicious code on the target se
               rver.
Resolution   : Determine if the handler requires both Execute/Script and Write permissions, and revoke the one that is
               not needed.
Compliance   :
Help         : http://go.microsoft.com/fwlink/?LinkId=130708
Excluded     : False

ResultNumber : 3
ModelId      : Microsoft/Windows/WebServer
RuleId       : 1
ResultId     : 1122680488
Severity     : Error
Category     : Security
Title        : Grant a handler execute/script or write permissions, but not both
Problem      : The attribute 'accessPolicy' in the handlers section under path 'MACHINE/WEBROOT/APPHOST' is set to allo
               w both Execute/Script and Write permissions.
Impact       : By allowing both Execute/Script and Write permissions, a handler can run malicious code on the target se
               rver.
Resolution   : Determine if the handler requires both Execute/Script and Write permissions, and revoke the one that is
               not needed.
Compliance   :
Help         : http://go.microsoft.com/fwlink/?LinkId=130708
Excluded     : False

ResultNumber : 4
ModelId      : Microsoft/Windows/WebServer
RuleId       : 2
ResultId     : 4106384391
Severity     : Information
Category     : Security
Title        : The configuration attribute 'notListedCgisAllowed' should be false
Problem      :
Impact       :
Resolution   :
Compliance   : The IIS Best Practices Analyzer scan has determined that you are in compliance with this best practice.
Help         :
Excluded     : False

ResultNumber : 5
ModelId      : Microsoft/Windows/WebServer
RuleId       : 1
ResultId     : 3312542137
Severity     : Information
Category     : Security
Title        : The configuration attribute 'notListedIsapisAllowed' should be false
Problem      :
```

Abbildung 44.9
Aufruf der Prüfergebnisse mit Get-BPAResult

Tipp: Mit `Set-BPAResult` kann man einzelne Teilergebnisse („Results“) ausschließen. Der folgende Befehl schließt alle Ergebnisse mit einem bestimmten Titel aus:

```
Get-Bparesult Microsoft/Windows/Webserver | where { $_.Title -eq
"Grant a handler execute/script or write permissions, but not both"
} | Set-BPAResult Microsoft/Windows/Webserver -exclude $true
```

Die „Result“-Einträge werden dadurch nicht gelöscht, aber mit „Excluded: True“ markiert.

Anschließend kann man dann mit

```
Get-Bparesult Microsoft/Windows/Webserver | where { $_.excluded -eq
$false }
```

auf die verbliebenen Ergebnisse zugreifen.

`Set-BPAResult` ist sinnvoll, um bei einer großen Menge von Ergebnissen den Überblick zu behalten, was man schon bearbeitet/erledigt hat bzw. was man ignorieren möchte.

45 Grafische Benutzeroberflächen

System.Windows.Forms

Die Microsoft Shell besitzt keine eingebauten Commandlets zur Anzeige grafischer Benutzerschnittstellen. Es spricht aber nichts dagegen, die `System.Windows.Forms`-Bibliothek (kurz: Windows Forms oder WinForms) von .NET direkt zu nutzen. Auch die Windows Presentation Foundation (WPF) kann man verwenden. In dem PowerShellPack aus dem Windows 7 Resource Kit gibt es ein PowerShell-Modul, das in rund 700 Commandlets die Funktionen von WPF kapselt. Lesen Sie dazu das Unterkapitel „WPF PowerShell Kit (WPK)".



Für die ausführliche Erläuterung der Windows-Forms-Bibliothek (einige Hundert Klassen!) ist in diesem Buch kein Raum. Zwei Beispiele sollen den Ansatz erläutern.

45.1 Eingabemasken

Das folgende Skript erzeugt eine Eingabemaske für drei Werte. Zur Vereinfachung gibt es folgende Einschränkungen:

- Die Eingabefelder werden automatisch angeordnet und nicht absolut positioniert („Flussgestaltung", vgl. HTML).
- Das Formular kann nur über das Kreuz in der Fensterzeile geschlossen werden. Es gibt keine zusätzliche Schaltfläche (weil es kompliziert ist, in einem Formular mit der PowerShell Programmcode zu hinterlegen).



Abbildung 45.1 *Ein mit Windows Forms in der PowerShell erzeugtes Eingabefenster*

Form

Das folgende PowerShell-Skript zeigt das Beispiel, in dem ein Formular (`Form`), ein Flussgestaltungsbereich (`FlowLayoutPanel`), drei Beschriftungsfelder (`Label`) und drei Eingabefelder (`Textbox`) zum Einsatz kommen. Wichtig ist, dass der Bereich das Formular ausfüllt (`[System.Windows.Forms.DockStyle]::Fill`) und man die Steuerelemente korrekt hintereinander verschachtelt (`Controls.Add()`).

Listing 45.1 *Eingabefenster anzeigen und auswerten [Einsatzgebiete\GUI\GUI_Form.ps1]*

```
# Windows Forms laden
[System.Reflection.Assembly]::LoadWithPartialName("System.windows.
forms")

# Fenster erzeugen
$form = New-Object "System.Windows.Forms.Form"
$form.Size = New-Object System.Drawing.Size @(200,200)
$form.topmost = $true
$form.text = "Eingabeformular"

# Bereich erzeugen
$Bereich = New-Object "System.Windows.Forms.FlowGestaltungBereich "
$Bereich.Dock = [System.Windows.Forms.DockStyle]::Fill
$form.Steuerelements.Add($Bereich)

# Textfelder erzeugen
$L1 = New-Object "System.Windows.Forms.Label"
$L2 = New-Object "System.Windows.Forms.Label"
$L3 = New-Object "System.Windows.Forms.Label"
$T1 = New-Object "System.Windows.Forms.Texteingabefeld"
$T2 = New-Object "System.Windows.Forms.Texteingabefeld"
$T3 = New-Object "System.Windows.Forms.Texteingabefeld"
$B1 = New-Object "System.Windows.Forms.Schaltfläche"

# Texte setzen
$L1.Text = "Name:"
$L2.Text = "E-Mail:"
$L3.Text = "Website:"

# Größen setzen
$T1.Width = 180
$T2.Width = 180
$T3.Width = 180

# Elemente zum Bereich hinzufügen
$Bereich.Steuerelements.Add($L1)
$Bereich.Steuerelements.Add($T1)
$Bereich.Steuerelements.Add($L2)
$Bereich.Steuerelements.Add($T2)
$Bereich.Steuerelements.Add($L3)
$Bereich.Steuerelements.Add($T3)

# Fenster anzeigen
$form.showdialog()

# Werte ausgeben
"Eingegeben wurden: " + $T1.Text + ";" + $T2.Text + ";" + $T3.Text
```

45.2 Universelle Objektdarstellung

PropertyGrid

Wenn man ein Objekt mit vielen Eigenschaften darstellen möchte, ist die obige Vorgehensweise der einzelnen Erzeugung von Windows-Forms-Elementen sehr aufwändig. Einfacher geht es mit dem in Windows Forms definierten Steuerelement `PropertyGrid`, an das man jedes beliebige .NET-Objekt binden kann und das erfolgte Änderungen auch an das Objekt weitergibt.

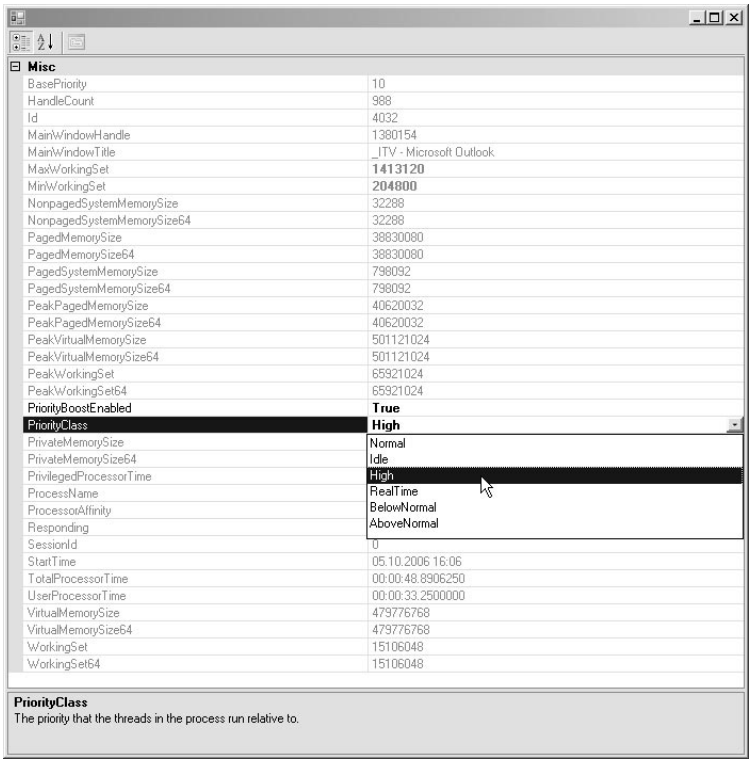


Abbildung 45.2
Anzeige und Änderung eines Process-Objekts mit einem Windows Forms PropertyGrid

```
# Windows Forms laden
[System.Reflection.Assembly]::LoadWithPartialName("System.windows.
forms")

# Fenster erzeugen
$form = New-Object "System.Windows.Forms.Form"
$form.Size = New-Object System.Drawing.Size @(700,800)
$form.topmost = $true

# PropertyGrid erzeugen
$PG = New-Object "System.Windows.Forms.PropertyGrid"
$PG.Dock = [System.Windows.Forms.DockStyle]::Fill
$form.Steuerelements.Add($PG)
```

Listing 45.2
Anzeige und Änderung eines Process-Objekts mit einem Windows Forms PropertyGrid [GUI_Propertygrid.ps1]

```
# Inhalt an PropertyGrid zuweisen
$i = Get-Process  outlook
$PG.selectedobject = $i

# Fenster anzeigen
$form.showdialog()
```

45.3 Zwischenablage

Zwischenablage

Zum Befüllen und Auslesen der Zwischenablage gibt es in den PSCX die Commandlets:

- `Write-Clipboard`
- `Set-Clipboard`
- `Get-Clipboard`

Abbildung 45.3
Einsatz von Set-Clipboard

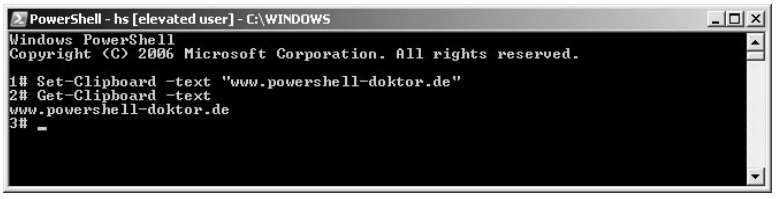
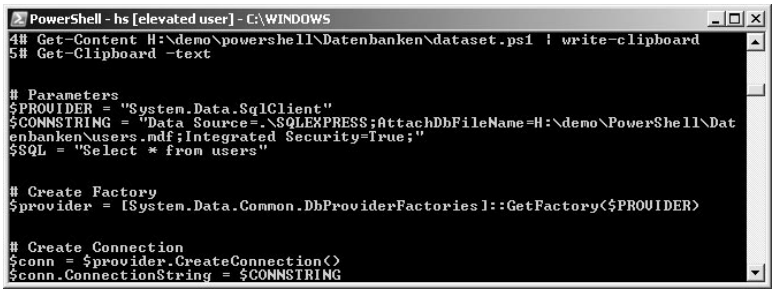


Abbildung 45.4
Einsatz von Write-Clipboard



45.4 WPF PowerShell Kit (WPK)

Thread-Modell

Die Verwendung der Windows Presentation Foundation (WPF) war in PowerShell 1.0 nicht möglich, weil das Thread-Modell der PowerShell (MTA) und von WPF (STA) nicht zueinander passten. In PowerShell 2.0 ist weiterhin MTA der Standard, aber die PowerShell lässt sich mit einem Schalter in den STA-Modus versetzen:

```
powershell.exe -sta
```

Die ISE hat das Problem nicht, denn da die ISE selbst in WPF geschrieben ist, verwendet sie im Standard „STA“.

Man kann die WPF-Bibliothek (`System.Windows`) direkt verwenden. Sehr viel eleganter ist die Verwendung aber mit dem WPF PowerShell Kit (WPK) aus dem Windows 7 Resource Kit PowerShellPack. Dieses Modul enthält 716 Commandlets.

Das WPK-Modul aktiviert man mit `Import-Module WPK`. Dieser Import kann aufgrund seiner Größe einige Sekunden dauern.

Hello World mit WPF

Einführende Beispiele geben immer den Text „Hello World“ auf dem Bildschirm aus. So soll es auch hier sein. Die Lösung ist mit einem einzigen Commandlet realisierbar.

New-Label "Hello World" -Show

New-Label



Abbildung 45.5
Hello-World-Bildschirm mit WPK

Es gibt zwar einen Parameter `-AsJob`, aber auch hierbei ist die PowerShell am Weiterarbeiten gehindert. (Es erscheint keine neue Befehlsaufforderung.)

Für das Bezeichnungsfeld stehen zahlreiche Formatierungen zur Verfügung, z.B.

```
New-Label "Hello World" -Show -FontSize 30 -Foreground red -Back-
ground yellow -FontStyle Italic -FontWeight Bold
```

Unzureichende Dokumentation

Dokumentation

Eine gravierende Schwäche des WPK ist, dass keine hinreichende Dokumentation mitgeliefert wird. Die nachfolgende Bildschirmabbildung zeigt die Dokumentation zu New-Label, die leider repräsentativ für die ganze Bibliothek ist.

Letztlich kann man WPK nur dann nutzen, wenn man auf die Entwicklerdokumentation im Microsoft Developer Network (MSDN) zum Namensraum System.Windows (*http://msdn.microsoft.com/en-us/library/system.windows.aspx*) zurückgreift. Aber auch hier ist das natürlich mühsam, denn `New-Label` gibt es hier nicht.

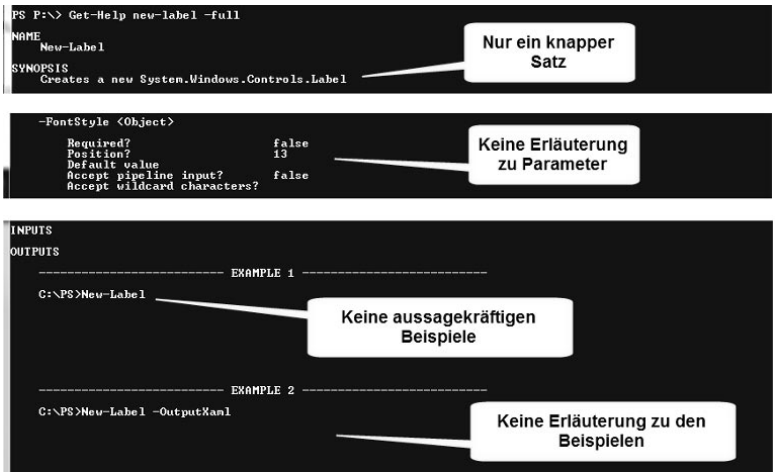


Abbildung 45.6 *Ausschnitte aus der unzureichenden Dokumentation des WPK am Beispiel von New-Label*

Hintergründe Um die MSDN-Dokumentation nutzen zu können, muss man insbesondere Folgendes verstehen:

- Die Commandlets der Form „New-XY“ beziehen sich auf eine Steuerelementklasse, die man im Namensraum `System.Windows.Controls` findet. `New-Label` erzeugt also die Klasse `System.Windows.Controls.Label`. Den Namen der Klasse inklusive Namensraum erhält man auch mit `New-XY | Get-Member`.
- Die Attribute (Properties) der WPF-Klasse sind Parameter des Commandlets. In der Detailansicht der einzelnen Properties erhält man auch eine Liste der möglichen Werte.
- Ebenso sind die Ereignisse als Parameter verfügbar, aber mit vorangestelltem „On_“, also „On_Click“ statt „Click“.

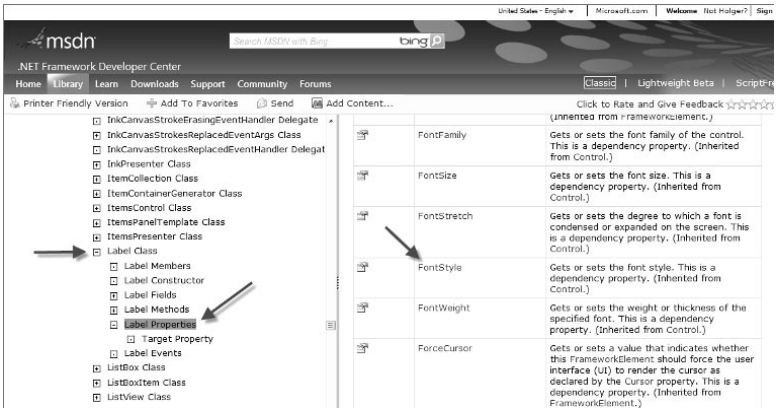


Abbildung 45.7 *MSDN-Dokumentation zu den möglichen Werten für FontStyle*

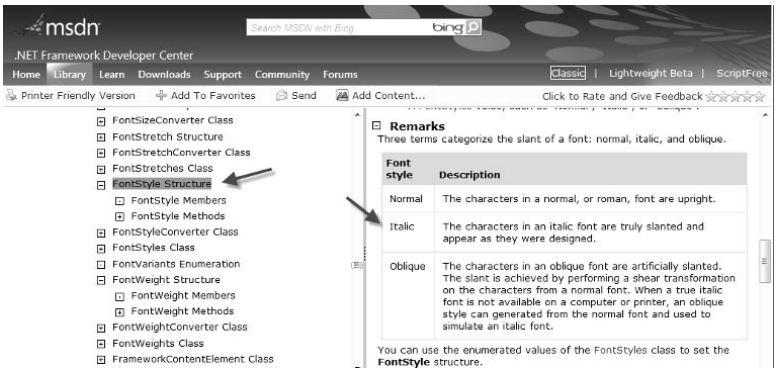


Abbildung 45.8 *MSDN-Dokumentation zu den möglichen Werten für FontStyle*



Alternativ kann man sich auch durch Ausprobieren durcharbeiten. Nutzen Sie die Tabulator-Vervollständigungsfunktion der PowerShell-Konsole oder die IntelliSense-Unterstützung eines Editors wie PowerShellPlus, um die vorhandenen Attribute zu erforschen. Geben Sie dann einfach irgendeinen Wert an, z.B. `-FontStyle xy`. Die PowerShell wird Ihnen dann eine Fehlermeldung zeigen, die etwas weiterhilft: *Cannot convert value "xy" to type "System.Windows.FontStyle"*. Durch diese Fehlermeldung wissen Sie genau, wonach Sie suchen müssen. Wenn Sie „`System.Windows.FontStyle`“ bei Google eingeben, steht der richtige Treffer meist direkt oben. Kurioserweise hat Microsofts Suchmaschine Bing die MSDN-Entwicklerdokumentation nicht so gut indiziert (siehe *Abbildung 45.9*).

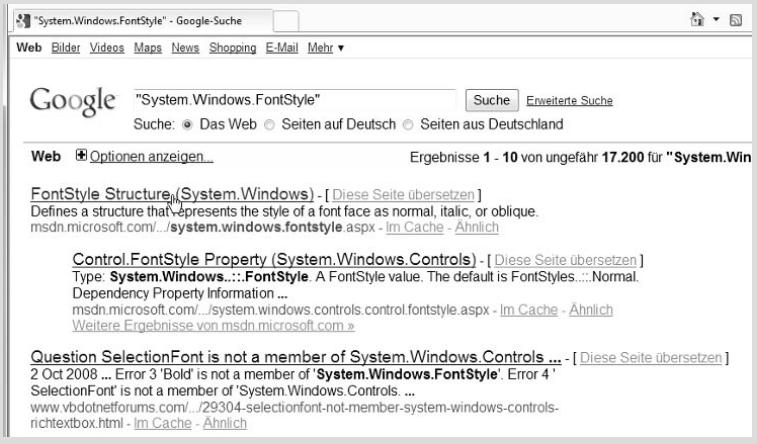


Abbildung 45.9*: Die ersten beiden Google-Treffer führen zur Dokumentation in Englisch und Deutsch.*

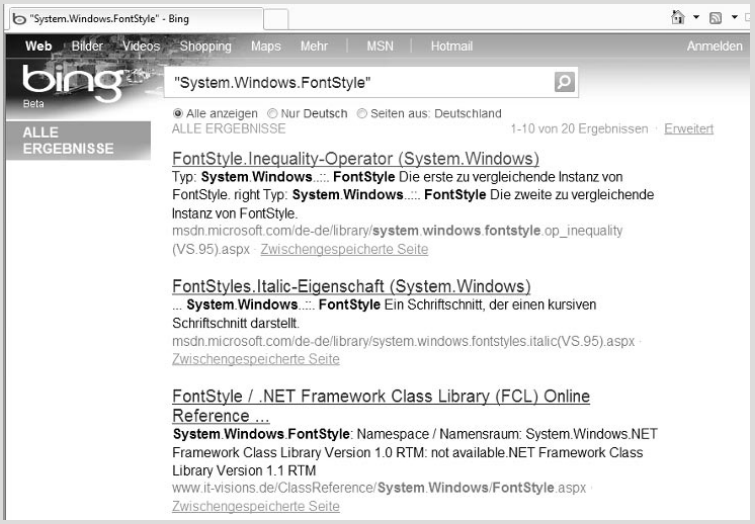


***Abbildung 45.10**: Bing kommt zwar in die Nähe der Lösung, aber findet nicht genau die richtige Seite.*

Schaltflächen und Ereignisse

New-Button

Eine Schaltfläche erstellt man mit `New-Button`. Wirklich Sinn macht eine Schaltfläche aber nur, wenn man auf das Klicken auch reagieren kann. Alle WPF-Steuerelemente stellen zahlreiche Ereignisse bereit. Zu einem Ereignis kann man eine Ereignisbehandlung hinterlegen, indem man den zugehörigen Parameter des WPK-Commandlets belegt. Diese Parameter beginnen alle mit „On_“.

Mit `Get-Help new-button -parameter "on_*" | ft name` erhält man eine Liste der Commandlets-Ereignisse des Button-Steuerelements. Erschrecken Sie sich nicht: Es sind 95 Ereignisse! Vom Namen her ist „On_Click“ das richtige Ereignis. In dem Parameter gibt man dann einen Skriptblock an.



Wichtig ist, dass der Skriptblock in der Zeile beginnt, in der auch der Parameter steht. Erst nach der geschweiften Klammer darf ein Zeilenumbruch erfolgen.

Falsch	`New-Button "Klick mich"  -Show -On_Click`
	`{ Aktion`
	`}`

Richtig	`New-Button "Klick mich"  -Show -On_Click {`
	`Aktion`
	`}`

Es folgt ein Beispiel.

Listing 45.3
Beispiel für die Ereignisbehandlung im WPK

```
Set-Variable -name count -Value 10 -Scope global

New-Button "Klick mich"  -Show -On_Click {
$this.Content = "Countdown: $Count" ;
$Count--;
if ($Count -lt 0) {
 [System.Windows.MessageBox]::Show("Ende!", "WPK-
Beispiel:CountDown");
 $Window.Close();
 };
}
```

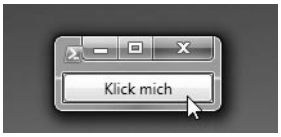


Abbildung 45.11
Nach dem Start des Beispiels



Abbildung 45.12
Nach dem ersten Klick

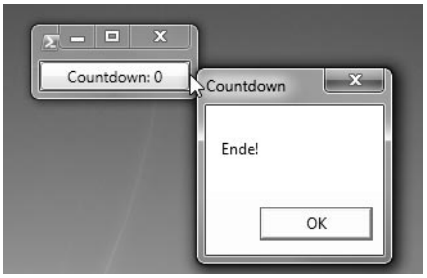


Abbildung 45.13
Nach dem elften Klick

Hinweise

Zu beachten sind folgende Punkte:

- Die Ereignis-Skriptblöcke sehen nur Variablen aus dem umgebenden Skript, die mit `Set-Variable` den Gültigkeitsbereich „Global" bekommen haben. Eine alternative Syntax wäre: `$global:Variable = Wert`.
- Ebenso sieht das aufrufende Skript die in den Ereignisbehandlungsroutinen definierten Variablen nur, wenn diese global definiert sind.
- In der Ereignisbehandlung adressiert `$this` das das Ereignis auslösende Steuerelement, also hier die Schaltfläche.
- In der Ereignisbehandlung adressiert `$window` das aktuelle Fenster. Über die Methode `Close()` kann man es schließen. Alternative wäre: `Close-Control $Window`.

Panel-Elemente zum Anordnen von Steuerelementen

Panel

Bisher bestand das Fenster immer nur aus einem Steuerelement. Das führt natürlich nicht sehr weit. Um mehrere Steuerelemente zu verwenden, muss man diese in einem sogenannten Panel verwenden. WPF kennt mehrere verschiedene Panels, die eine unterschiedliche Anordnung der Elemente durchführen.

Tabelle 45.1 *Panel-Elemente in WPF*

Name	Erläuterung
Canvas	Positionierung mit relativen Koordinaten. Das Panel-Element legt keine Gestaltung fest; die enthaltenen Elemente müssen selbst die Position bestimmen (vgl. absolute Positionierung in CSS).
Grid	Gitternetz-Layout (Tabelle mit Zeilen und Spalten). Die Steuerelemente bestimmen die Position im Gitter durch die Attribute `Row` und `Column`. In einer Gitterzelle ist die absolute Positionierung möglich, so dass im Fall der Reduktion auf eine Zelle (Standardeinstellung) das Grid dem Canvas sehr ähnlich ist.
UniformGrid	Gitternetz-Layout mit gleich großen Zellen
WrapPanel	Aneinanderreihung von links nach rechts bzw. unten nach oben mit automatischem Umbruch (vgl. HTML-Standardlayout)
DockPanel	Horizontale oder vertikale Aneinanderreihung mit automatischem Umbruch nach jedem Element
StackPanel	Horizontale oder vertikale Aneinanderreihung
VirtualizingStack-Panel	Ein Stackpanel, bei dem nur die wirklich sichtbaren Elemente auch verarbeitet werden (nur relevant bei Datenbindung)
TabPanel	Registerkarten in einem TabControl-Steuerelement

Zielsetzung soll es sein, das nachstehend abgebildete Anmeldefenster zu entwickeln. Das Fenster besteht aus zwei Label-Steuerelementen (`New-Label`), einem Texteingabe-Steuerelement (`New-TextBox`), einem Password-Steuerelement (`New-PasswordBox`) und einer Schaltfläche (`New-Button`). Als Panel kommt ein Grid (`New-Grid`) zum Einsatz. Das Ganze ist eingebettet in ein Fenster (`New-Window`). Der Farbübergang zwischen Hellgrün und Grün (möglicherweise in dem schwarzweiß gedruckten Buch schlecht zu erkennen) ist ein `LinearGradientBrush`.



Fenster wurden in den bisherigen Beispielen implizit erzeugt. Die explizite Erzeugung ermöglicht Einfluss auf Größe und Überschrift.

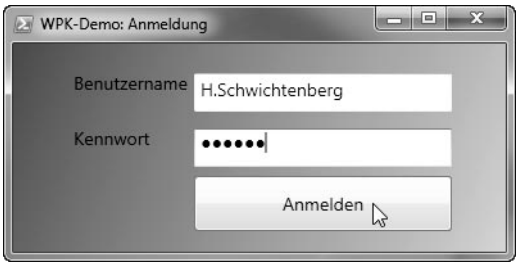


Abbildung 45.14
Ein Anmeldefenster mit Hilfe von WPK

Verschachtelung der Elemente

Die Verschachtelung der Elemente (Window enthält Grid, Grid enthält andere Steuerelemente) erfolgt durch geschweifte Klammern. Zu beachten ist dabei, dass eine Verschachtelung nur dann gebildet wird, wenn die geschweifte Klammer in der gleichen Zeile steht wie das Commandlet, das das übergeordnete Steuerelement erzeugt. Erst innerhalb der Klammer darf ein Zeilenumbruch erfolgen.

Falsch:	`New-Windows -Title "xy"`
	`{ New-Grid …`
	`}`

Richtig:	`New-Windows -Title "xy" {`
	`New-Grid …`
	`}`

Gleiches gilt für die Ereignisbehandlungsroutine, die für das `Click`-Ereignis um Button-Steuerelemente (`On_Click`) definiert wird.

Es folgt der PowerShell-Skriptcode für das Beispiel.

Listing 45.4
Implementierung des Anmeldefensters (im Schachtelungsstil)

```
Import-Module WPK

# Farbübergang für Hintergrund definieren
$global:b = New-LinearGradientBrush -GradientStops  {
New-GradientStop -color Green -Offset 0
New-GradientStop -color lightGreen -Offset 1
}

# Fenster mit Unterelementen definieren
New-Window -Title "WPK-Demo: Anmeldung" -background $b -Width 400 -
Height 200 -WindowStartupLocation centerscreen -show {
 New-Grid -Rows 3  -Width 300 -Height 130 -Columns 'Auto','1*'  {
    New-Label "Benutzername" -FontSize 14
    New-TextBox -Name UserName -Column 1 -FontSize 14 -Height 30 -
    On_Loaded { Set-Variable C_UserName $this -Scope Global }
    New-Label "Kennwort" -Row 1 -FontSize 14
    New-PasswordBox -Name Password -Column 1 -FontSize 14 -Height
    30 -Row 1 -On_Loaded { $global:C_Password = $this }
    New-Button -Name Login -FontSize 14 -Content "Anmelden" -
    Column 1 -Row 2 -On_Click{
```

```
    $Window.Close(); } # Ende Event
 } # Ende Grid
} # Ende Window

"Eingegeben wurden folgende Anmeldedaten: {0} / {1}" -f  $C_
UserName.Text, $C_Password.Password
```

Die Schachtelung von Elementen kann unübersichtlich werden. Beim `LinearGradientBrush` ist das Schachtelungsprinzip schon zu Gunsten des Verweisprinzips durchbrochen: Der Pinsel ist erst definiert und dann später über eine Variable ($b) referenziert.

Man könnte das Skript auch so schreiben:

Listing 45.5 *Implementierung des Anmeldefensters (im Verweisstil)*

```
Import-Module WPK

# Farbübergang für Hintergrund definieren
$global:b = New-LinearGradientBrush -GradientStops  {
New-GradientStop -color Green -Offset 0
New-GradientStop -color lightGreen -Offset 1
}

$global:l1 = New-Label "Benutzername" -FontSize 14
$global:tb = New-TextBox -Name UserName -Column 1 -FontSize 14 -
Height 30 -On_Loaded { $global:C_UserName = $this }
$global:l2 = New-Label "Kennwort" -Row 1 -FontSize 14
$global:pw = New-PasswordBox -Name Password -Column 1 -FontSize 14
-Height 30 -Row 1 -On_Loaded { $global:C_Password = $this }
$global:bt = New-Button -Name Login -FontSize 14 -Content
"Anmelden" -Column 1 -Row 2 -On_Click { $Window.Close() }

$global:g = New-Grid -Rows 3  -Width 300 -Height 130 -Columns
'Auto','1*'  { $l1, $tb, $l2, $pw, $bt }
$global:w = New-Window -Title "WPK-Demo: Anmeldung" -background $b
-Width 400 -Height 200 -WindowStartupLocation centerscreen -show {
$g }

"Eingegeben wurden folgende Anmeldedaten: {0} / {1}" -f  $C_
UserName.Text, $C_Password.Password
```

Weitere Möglichkeiten

Ausblick WPF ist eine derart komplexe Bibliothek, dass dieses Buch hier nur einen kleinen Ausschnitt wiedergeben kann. Zu erwähnen sind noch folgende Möglichkeiten:

- Weitere Eingabesteuerelemente wie `New-CheckBox`, `New-ComboBox`, `New-ListBox`, `New-Menu`, `NewRichTextBox`, `New-ScrollBar`, `New-TreeView`, `New-Slider`, `New-RadioButton`
- Weitere Textanzeigesteuerelemente wie `New-TextBlock`, `New-StatusBar`, `New-ProgressBar`

- Anzeige von Videos (`New-MediaElement`)
- Zeichnen mit Elementen wie `New-Line`, `New-Ellipse`, `New-Rectangle` und `New-Image`
- Transformationen von Elementen wie Kippen (z.B. Parameter `-RenderTransform`)
- Animationen von Elementen (z.B. `New-Int64Animation` und `New-Storyboard`)
- Komposition von Elementen (z.B. ein Kontrollkästchen in einer Schaltfläche)
- Nahtlose Vergrößerung von Elementen (`New-ViewBox`)
- Datenbindung (z.B. `-ItemsSource`, `Get-PowerShellDataSource`, `ConvertTo-DataTemplate`)
- Definition von Oberflächen in XML-Form durch die XML Application Markup Language (XAML)

Listing 45.6
Das Anmeldefenster in XAML-Form

```
<?xml version="1.0" encoding="utf-16"?>
<Window Title="WPK-Demo: Anmeldung"
WindowStartupLocation="CenterScreen" Width="400" Height="200"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:sma="clr-
namespace:System.Management.Automation;assembly=System.Management.A
utomation" xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:sc="clr-namespace:System.Collections;assembly=mscorlib">
  <Window.Background>
    <LinearGradientBrush>
      <LinearGradientBrush.GradientStops>
        <GradientStop Color="#FF008000" Offset="0" />
        <GradientStop Color="#FF90EE90" Offset="1" />
      </LinearGradientBrush.GradientStops>
    </LinearGradientBrush>
  </Window.Background>
  <Window.Resources>
    <sma:PSObject x:Key="Scripts" />
    <sc:Hashtable x:Key="TemporaryControls" />
    <sma:PSObject x:Key="Timers" />
  </Window.Resources>
  <Grid Width="300" Height="130">
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="Auto" />
      <ColumnDefinition Width="*" />
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
      <RowDefinition Height="*" />
      <RowDefinition Height="*" />
      <RowDefinition Height="*" />
    </Grid.RowDefinitions>
    <Grid.Resources>
      <sma:PSObject x:Key="Scripts" />
```

```
      <sc:Hashtable x:Key="TemporaryControls" />
      <sma:PSObject x:Key="Timers" />
    </Grid.Resources>
    <Label FontSize="14">
      <Label.Resources>
        <sma:PSObject x:Key="Scripts" />
        <sc:Hashtable x:Key="TemporaryControls" />
        <sma:PSObject x:Key="Timers" />
      </Label.Resources>Benutzername</Label>
    <TextBox FontSize="14" Name="UserName" Height="30"
Grid.Column="1" xml:space="preserve">
      <TextBox.Resources>
        <sc:Hashtable x:Key="EventHandlers">
          <RoutedEventHandler x:Key="On_Loaded" />
        </sc:Hashtable>
        <sma:PSObject x:Key="Scripts" />
        <sc:Hashtable x:Key="TemporaryControls" />
        <sma:PSObject x:Key="Timers" />
      </TextBox.Resources>
    </TextBox>
    <Label FontSize="14" Grid.Row="1">
      <Label.Resources>
        <sma:PSObject x:Key="Scripts" />
        <sc:Hashtable x:Key="TemporaryControls" />
        <sma:PSObject x:Key="Timers" />
      </Label.Resources>Kennwort</Label>
    <PasswordBox FontSize="14" Name="Password" Height="30"
Grid.Column="1" Grid.Row="1">
      <PasswordBox.Resources>
        <sc:Hashtable x:Key="EventHandlers">
          <RoutedEventHandler x:Key="On_Loaded" />
        </sc:Hashtable>
        <sma:PSObject x:Key="Scripts" />
        <sc:Hashtable x:Key="TemporaryControls" />
        <sma:PSObject x:Key="Timers" />
      </PasswordBox.Resources>
    </PasswordBox>
    <Button FontSize="14" Name="Login" Grid.Column="1"
Grid.Row="2">
      <Button.Resources>
        <sc:Hashtable x:Key="EventHandlers">
          <RoutedEventHandler x:Key="On_Click" />
        </sc:Hashtable>
        <sma:PSObject x:Key="Scripts" />
        <sc:Hashtable x:Key="TemporaryControls" />
        <sma:PSObject x:Key="Timers" />
      </Button.Resources>Anmelden</Button>
  </Grid>
</Window>
```



Teil 4

Profiwissen – Erweitern der PowerShell

In diesem Buchteil werden die Möglichkeiten zur Erweiterung der PowerShell behandelt.

Erweiterungsoptionen

Die PowerShell lässt sich wie folgt erweitern:

1. Entwicklung eigener Commandlets per PowerShell-Skript (siehe funktionsbasierte Commandlets)
2. Entwicklung eigener Commandlets per C# (alternativ auch Visual Basic oder eine andere .NET-Sprache)
3. Entwicklung eigener PowerShell-Provider per C# (alternativ auch Visual Basic oder eine andere .NET-Sprache)
4. Außerdem ist es möglich, die PowerShell in Anwendungen als Scripting-Umgebung zu integrieren (PowerShell-Hosting). Dies geht nur mit einer „richtigen" .NET-Sprache.



Der dritte Punkt in der Liste ist derart aufwändig, dass er nicht in diesem Buch behandelt werden kann.

Erforderliches Vorwissen

Es ist besonders darauf hinzuweisen,

- dass hier Punkte behandelt werden, die über das hinausgehen, was die meisten Systemadministratoren mit der PowerShell unternehmen werden.
- dass einige dieser Möglichkeiten Kenntnisse in der Programmiersprache C# und der Entwicklungsumgebung Visual Studio erfordern. Diese Kenntnisse können in diesem Buch allein schon aus Platzgründen nicht vermittelt werden. Hier sei zum Beispiel auf [SCH02] verwiesen.

46 Entwicklung von Commandlets in der PowerShell-Sprache

Funktionsbasierte Commandlets

Schon im Laufe des Buchs wurden an verschiedenen Stellen PowerShell-Funktionen erstellt (z.B. `Get-DirSize`), die wie Commandlets heißen und sich genau wie Commandlets aufrufen lassen. Diese Funktionen waren aber keine echten Commandlets. Echte Commandlets bieten noch einige weitere Möglichkeiten:

- Sie können Bedingungen für die übergebenen Parameter festlegen.
- Sie können festlegen, wie Inhalte der Pipeline auf die Parameter abgebildet werden, wenn sie in einer Pipeline verwendet werden.
- Sie können asynchron arbeiten, d.h., sie beginnen mit der Verarbeitung schon, wenn das erste Objekt angeliefert wird.
- Man kann mit `Get-Help` Hilfetexte abrufen.

PowerShell-Funktionen können aber auch echte Commandlets werden. Der Schritt zwischen einer PowerShell-Funktion und einem skriptbasierten Commandlet (auch „funktionsbasierte Commandlets" oder „Advanced Function" genannt) ist nicht sehr groß.

Für dieses Kapitel dieses Buchteils werden keine C#-Kenntnisse benötigt.



46.1 Aufbau eines skriptbasierten Commandlets

Ein skriptbasiertes Commandlet besteht aus folgenden Teilen:

- Mindestens eine in der PowerShell-Sprache geschriebene Funktion, deren Name wie ein Commandlet aus Verb und Substantiv aufgebaut ist
- Eine Liste von Parametern
- Ein Unterblock mit dem Namen `begin`, der einmalig zu Beginn der Verarbeitung des Commandlets ausgeführt wird
- Ein Unterblock `process`, der für jedes Objekt in der Pipeline ausgeführt wird. ACHTUNG: `Process` wird auch bei leerer Pipeline genau einmal aufgerufen.
- Ein Unterblock `end`, der einmalig zum Ende der Verarbeitung ausgeführt wird

Beispiel: Test-Pipeline

Test-Pipeline

Das folgende skriptbasierte Commandlet ist zunächst nur ein Test, um die Wirkungsweise eines solchen Commandlets zu demonstrieren. In jeder der drei Phasen macht das Commandlet eine Ausgabe. In `process` wird für jedes Objekt in der Pipeline der Typ ausgegeben. Das Commandlet besitzt einen Parameter. Der Skriptcode besteht neben der Definition der Methode mit den drei Verarbeitungsschritten aus zwei „freien" Befehlen, die bei der Installation ausgeführt werden: einer Alias-Definition und einer Meldung für die erfolgreiche Installation.

Der folgende Programmcode muss in einer PowerShell-Skriptdatei abgespeichert werden. Dann muss diese PowerShell-Datei mit einer speziellen Operation in die Shell eingebunden werden. Die Operation besteht aus dem Operator „." und dem vollständigen Pfad zur Skriptdatei (das Verfahren heißt im Englischen „Dot Sourcing", übersetzt „Punktquelle"), z.B.:

```
. H:\demo\powershell\Test-pipeline.ps1
```



Den obigen Befehl legt man üblicherweise in der *profile.ps1*-Datei ab, damit der Befehl immer beim Start der PowerShell schon ausgeführt wird.

Danach kann das skriptbasierte Commandlet wie ein normales Commandlet mit oder ohne Parameter und mit oder ohne Pipeline-Inhalt aufgerufen werden:

```
Test-Pipeline  "Hello Welt!"
Test
Get-Process a* | Test-Pipeline
```

Die dem Listing folgende Bildschirmabbildung zeigt die Reaktion der Shell auf die obigen drei Anweisungen.



Wichtig ist, dass das Skript nach jeder Änderung an seinem Quelltext neu in die PowerShell eingebunden werden muss.

Listing 46.1 *Definition eines skriptbasierten Commandlets [Test-Pipeline.ps1]*

```
# ------------------------------------------------------------------
# Author:       Dr. Holger Schwichtenberg
# Description:  Demo for a function based Commandlet
# Usage:        This file contains a function-based Commandlet.
#               In order to use
#               it, you must include the file into your shell:
#               PS> . c:\bin\Test-Pipeline.ps1
# Date:         09/25/2006
# Website:      http://www.PowerShell-Doktor.de
# ------------------------------------------------------------------
```

```
# Function
function Test-Pipeline {
    param([string[]]$Parameter)

    begin {

if ($Parameter)
      { "Commandlet starts with Parameter " + $Parameter + "..." }
    else
  { "Commandlet starts without Parameter..." }
}
    process {
if ($_)
 {
 "Commandlet receives a pipeline objekt of type " + $_.GetType()
     }
else
 {"The Pipeline is empty!"
 }
}
    end {
   "Commandlet ends..."
    }
}

# Define alias for function
Set-Alias test test-pipeline

# Confirm installation
"Function-based Commandlet Test-Pipeline (alias: Test) successfully
installed!"
```

```
PS H:\demo\PS>
PS H:\demo\PS>
PS H:\demo\PS> . H:\demo\PS\powershellide\Test-pipeline.ps1
Function-based Commandlet Test-Pipeline (alias: Test) successfully installed!
PS H:\demo\PS>
PS H:\demo\PS> Test-Pipeline "Hallo Welt!"
Commandlet starts with Parameter Hallo Welt!...
Die Pipeline ist leer!
Commandlet ends...
PS H:\demo\PS>
PS H:\demo\PS> Test "Hallo Welt!"
Commandlet starts with Parameter Hallo Welt!...
Die Pipeline ist leer!
Commandlet ends...
PS H:\demo\PS>
PS H:\demo\PS> Test
Commandlet starts without Parameter...
Die Pipeline ist leer!
Commandlet ends...
PS H:\demo\PS>
PS H:\demo\PS> get-process a* ! test
Commandlet starts without Parameter...
Commandlet receives a pipeline objekt of type System.Diagnostics.Process
Commandlet receives a pipeline objekt of type System.Diagnostics.Process
Commandlet receives a pipeline objekt of type System.Diagnostics.Process
Commandlet receives a pipeline objekt of type System.Diagnostics.Process
Commandlet ends...
PS H:\demo\PS>
```

Abbildung 46.1 *Beispiel zur Installation und Ausführung des funktionsbasierten Commandlets Test-Pipeline*



Das obige Commandlet `Test-Pipeline` dient der Veranschaulichung des Grundprinzips skriptbasierter Commandlets, bietet aber in der Praxis nur einen geringen Wert. Im Folgenden finden Sie skriptbasierte Commandlets zur Verzeichnisdienststeuerung.

46.2 Parameterfestlegung

Bei funktionsbasierten Commandlets kann man Parameter auf die gleiche Weise definieren und übergeben wie bei normalen Funktionen, also wahlweise im Funktionskopf, z.B.

```
function Add-LDAPObject([string]$Container, [string]$Class,
[string]$RDN) {
    …
}
```

oder mit `param(…)` im Funktionsrumpf:

```
function Add-LDAPObject {
    param([string]$Container, [string]$Class, [string]$RDN)
…
}
```



Die zweite Form bietet mehr Möglichkeiten, wie im Folgenden noch aufzeigt werden wird.

Beispiel: Commandlets für den Verzeichnisdienstzugriff

Das Beispiel umfasst Commandlets zur Automatisierung von LDAP-basierten Verzeichnisdiensten wie dem Active Directory. Diese Commandlets kapseln die Möglichkeiten des FCL-Namensraums `System.DirectoryServices`.



Dieses Beispiel wurde für die PowerShell 1.0 erstellt. Inzwischen gibt es für die PowerShell 2.0 derartige Commandlets in Windows 7 und Windows Server 2008 R2. Für ältere Betriebssysteme sind die Commandlets aber weiterhin eine wichtige Ergänzung.

Get-LDAPObject

Für den Administrator ist die Active-Directory-Automatisierung mit eigenen Commandlets einfacher als mit den FCL-Klassen. Sinnvoll ist z.B.:

```
Get-LDAPObject "LDAP://XFilesServer1/CN=Fox
Mulder,OU=Agents,DC=FBI,DC=net" | Select-Object sn, l, mail,
telephonenumber
```

genauso wie:

```
"LDAP://XFilesServer1/CN=Fox Mulder,OU=Agents,DC=FBI,DC=net" | Get-
LDAPObject | Select-Object sn, l, mail, telephonenumber
```

zum Zugriff auf ein Einzelobjekt und

Get-LDAP-Children

```
Get-LDAPChildren "LDAP://XFilesServer1/OU=Agents,DC=FBI,DC=net"
```

zum Auflisten eines Containers.

Am besten sollte man die beiden Commandlets auch kombinieren können:

```
Get-LDAPObject "LDAP://XFilesServer1/OU=Agents,DC=FBI,DC=net" |
Get-LDAPChildren
```



Abbildung 46.2 *Die Commandlets Get-LDAPObject und Get-LDAP-Children im Einsatz*

Add-Directory-Object, Remove-DirectoryObject

Auch das Anlegen und Löschen von Objekten wäre so viel schöner:

```
"OU anlegen..."
Add-LDAPObject "LDAP://XFilesServer1/OU=Agents,DC=FBI,DC=net" -
class "organizationalUnit" -rdn "ou=formerAgents"
"Benutzer anlegen..."
Add-LDAPObject "LDAP://XFilesServer1/
ou=formerAgents,OU=Agents,DC=FBI,DC=net" "user" "cn=AlexKrycek"
"Auflisten..."
Get-LDAPChildren "LDAP://XFilesServer1/
ou=formerAgents,OU=Agents,DC=FBI,DC=net"
"Alle Objekte im Container löschen..."
Get-LDAPChildren "LDAP://XFilesServer1/
ou=formerAgents,OU=Agents,DC=FBI,DC=net" | Remove-LDAPObject |
select name
"Container selbst löschen..."
Remove-LDAPObject "LDAP://XFilesServer1/
ou=formerAgents,OU=Agents,DC=FBI,DC=net" | select name
```

Skriptbasiertes Commandlet

Das folgende Skript zeigt die Realisierung von vier skriptbasierten Commandlets zur Steuerung des Active Directory oder anderer LDAP-basierter Verzeichnisdienste aus der PowerShell heraus:

- `Get-LDAPObject`: Zugriff auf ein einzelnes Verzeichnisobjekt
- `Get-LDAPChildren`: Zugriff auf den Inhalt eines Containerobjekts (listet die Unterelemente auf)
- `Add-LDAPObject`: Anlegen eines Verzeichnisobjekts
- `Remove-LDAPObject`: Löschen eines Verzeichnisobjekts

Das Commandlet `Get-LDAPObject` akzeptiert als Eingabe ein Array mit LDAP-Pfaden als Zeichenketten als Parameter oder in der Pipeline.

Beide folgenden Befehle sind also gültige Anweisungen:

- `Get-LDAPObject "LDAP://XFilesServer1/OU=Agents,DC=FBI,DC=net"`
- `"LDAP://XFilesServer1/OU=Agents,DC=FBI,DC=net"| Get-LDAPObject`

`Get-LDAPChildren` und `Remove-LDAPObject` akzeptieren als Eingabe:

- ein Array mit LDAP-Pfaden als Zeichenketten als Parameter oder in der Pipeline und/oder

eine Menge von `DirectoryEntry`-Objekten in der Pipeline.

Alle vier folgenden Befehle sind also gültige und gleichbedeutende Anweisungen:

- `Get-LDAPChildren "LDAP://XFilesServer1/OU=Agents,DC=FBI,DC=net"`
- `"LDAP://XFilesServer1/OU=Agents,DC=FBI,DC=net" | Get-LDAP-Children`
- `Get-LDAPObject "LDAP://XFilesServer1/OU=Agents,DC=FBI,DC=net" | Get-LDAPChildren`
- `"LDAP://XFilesServer1/OU=Agents,DC=FBI,DC=net"| Get-LDAPObject | Get-LDAPChildren`

Bei `Add-LDAPObject` können als Eingabe nur drei Zeichenkettenparameter verwendet werden.

Bitte beachten Sie, dass die skriptbasierten Commandlets erst mit dem .-Befehl („Dot Sourcing“) unter Angabe des kompletten Dateisystempfads eingebunden werden müssen:

```
. H:\demo\PS\LDAP_Commandlets.ps1
```



Listing 46.2 Commandlets für den Zugriff auf das Active Directory und andere LDAP-Server [LDAP_Commandlets.ps1]

```
# ------------------------------------------------------------------
----
# Author:    Dr. Holger Schwichtenberg
# Desc:      PowerShell Commandlets for handling LDAP-Objects
# Usage:     This file contains a function-based Commandlet.  In
#            order to use it, you must dot source the file into
#            your shell e.g.:
#            PH> . c:\PSExtensions\LDAP_Commandlets.ps1
# Date:      10/05/2006
# Version: 2.0
# Host:      PowerShell
# ------------------------------------------------------------------
----

# Get single LDAP object
function Get-LDAPObject {
    param([string[]]$LDAPPath)

    begin {
    }
    process {
 if ($_)
 {
   if ($_ -is [string])
   {
    new-object system.directoryservices.directoryEntry($_)
   }
   else
   { throw "Pipeline input must be [string]."
   }
 }
    }
    end {
        if ($LDAPPath) {
          foreach ($Path in $LDAPPath) {
            new-object
system.directoryservices.directoryEntry($Path)
          }
        }
    }
```

```
}
# Hilfsroutine
function getContainer([string] $path)
{
$con = new-object system.directoryservices.directoryEntry($path)
$con.PSBase.Children
}

# Get content of an LDAP container
function Get-LDAPChildren {
    param([string[]]$LDAPPath)

    begin {      }

    process {
    if ($_)
 {
   if ($_ -is [string])
   {
        getContainer($_)
   }
       elseif ($_ -is [System.DirectoryServices.DirectoryEntry])
   {
     getContainer($_.PSBase.Path)
   }
   else
   { throw "Pipeline input must be [string] or
[System.DirectoryServices.DirectoryEntry]."
   }
 }
    }
    end {
        if ($LDAPPath) {
          foreach ($Path in $LDAPPath) {
           getContainer($Path)
          }
        }
    }
}
# Remove an object from an LDAP container
function Remove-LDAPObject {
    param([string[]]$LDAPPath)

    begin {

function remove([string] $path)
{
if ([system.directoryservices.directoryEntry]::Exists($path))
     {
```

```
  $obj = new-object system.directoryservices.directoryEntry($path)
      $obj.PSBase.DeleteTree()
  $obj
     }
else
 {
 throw "Object does not exists!"
 }
}
    }
    process {
    if ($_)
 {
   if ($_ -is [string])
   {
        remove($_)
   }
       elseif ($_ -is [System.DirectoryServices.DirectoryEntry])
   {
     remove($_.PSBase.Path)
   }
   else
   { throw "Pipeline input must be [string] or
[System.DirectoryServices.DirectoryEntry]."
   }
 }
    }
    end {
        if ($LDAPPath) {
          foreach ($Path in $LDAPPath) {
           remove($Path)
          }
        }
    }
}

# Add a new object to a LDAP container
function Add-LDAPObject {
    param([string]$Container, [string]$Class, [string]$RDN)

    begin {  }

    process { }


    end {
    if ($Container -and $Class -and $RDN) {

 Write-Warning "Adding Object $RDN of type $Class to $Container"
```

```
if ([system.directoryservices.directoryEntry]::Exists($Container))
     {
  $obj = new-object
system.directoryservices.directoryEntry($Container)
  $newobj = $obj.PSBase.Children.Add([string]$RDN,[string]$Class)
      $newobj.PSBase.CommitChanges()
     }
else
 {
 throw "Container does not exists!"
 }
}
    }
}

# Define aliases for commandlet functions
Set-Alias LDP Get-LDAPObject
Set-Alias LDC Get-LDAPObject
Set-Alias RLDP RemoveLDAPObject
Set-Alias ALDP Add-LDAPObject

# Confirm installation
"Function-based commandlets for LDAP successfully installed!"
```

46.3 Auszeichnung der Parameterdefinitionen

In PowerShell 2.0 hat Microsoft die Möglichkeiten zum Erstellen von skriptbasierten Commandlets stark verbessert, so dass diese nun fast alle Möglichkeiten von .NET-basierten Commandlets haben. Insbesondere wurden zahlreiche Möglichkeiten ergänzt, die Parameter näher festzulegen. Solche Commandlets nennt Microsoft „Fortgeschrittene Funktionen“ (Advanced Functions).

Eine Advanced Function muss durch den Zusatz `[CmdletBinding()]` nach der geschweiften Klammer und vor der Parameterfestlegung deklariert werden.

Auszeichnungen zur Parameterdefinition

Die Möglichkeiten umfassen insbesondere die Festlegung:

- ob ein Parameter eine Pflichtangabe ist: `[Parameter(Mandatory=$true)]`
- dass ein Parameter seinen Wert aus der Pipeline erhalten darf, wobei das ganze Objekt gebunden wird `[Parameter(ValueFromPipeline=$true)]`
- dass ein Parameter seinen Wert aus der Pipeline erhalten darf, wobei ein einzelnes Attribut des Objekts gebunden wird `[Parameter(ValueFromPipelineByPropertyName=$true)]`

- einer Position für einen Parameter: `[Parameter(Position=0)]`
- eines Hilfetextes: `[Parameter(HelpMessage="Eine Hexadezimalzahl mit 1 bis 5 Stellen")]`
- der Anzahl der erlaubten Parameter: `[ValidateCount(2,5)]`
- ob leere Werte erlaubt sind: `[AllowNull()],[AllowEmptyString()], [AllowEmptyCollection()],[ValidateNotNull()]`
- der Länge einer Zeichenkette: `[ValidateLength(1,5)]`
- des Aufbaus einer Zeichenkette, beschrieben durch einen regulären Ausdruck: `[ValidatePattern("[0-9A-F]*")]`
- eines Ausdrucks, der wahr sein muss:`[ValidateScript({$_.StartsWith("A")})]`
- dass ein Parameter ein Wert aus einer Menge sein muss: `[ValidateRange("x", "y", "z")]`

Beispiel: Validate-CustomerID

In dem folgenden Beispiel wird ein Commandlet `Validate-CustomerID` in Form einer erweiterten Funktion definiert. Das Command selbst macht nur Testausgaben. Alle Prüfungen erfolgen durch die Beschreibungen der Parameter. Erlaubte Eingaben sind zwei Buchstaben gefolgt von ein bis drei Zahlen.

Listing 46.3
Beispiel „Validate-CustomerID" als erweiterte Funktion

```
# ------------------------------------------------------
# Author:      Dr. Holger Schwichtenberg
# Description: Demo for a function based Commandlet
# Usage:       This file contains an advanced function. In order
#              to use it, you must include the file into your shell:
#              PS> . c:\bin\Validate-Customer.ps1
# Date:        09/25/2006
# Website:     http://www.Windows-Scripting.com/PowerShell
# ------------------------------------------------------

# Function

function Validate-CustomerID {
[CmdletBinding()]
param(
[Parameter(Position=0,Mandatory=$true,
ValueFromPipelineByPropertyName=$true,
ValueFromPipeline=$true,HelpMessage="A Customer ID has two letters
and one to three digits.")]
[AllowNull()]
[ValidateLength(3,5)]
[ValidatePattern("[A-Z][A-Z][0-9]{1,3}")]
[ValidateScript({$_.StartsWith("A")})]
[string[]]
$CustomerID)
```

```
begin {

if ($CustomerID -eq $null)
{Write-Verbose "Commandlet starts without Parameter..." }
else
{Write-Verbose "Commandlet starts with Parameter $CustomerID ..." }
}

process {
Write-Verbose "Customer ID = $CustomerID"
if ($_)
{
Write-Verbose ("Commandlet receives a pipeline object of type " +
$_.GetType())
}
else
{
Write-Verbose "The Pipeline is empty!"
}
# Add Logic here (e.g. check database)
return $true
}

end {
"Commandlet ends..."

}
}
```

Listing 46.4 *Beispiele zum Aufruf „Validate-CustomerID“*

```
# Define alias for function
Set-Alias CID Validate-CustomerID

# Confirm installation
"Function-based Commandlet Test-Pipeline (alias: Test) successfully
installed!"

# Run Tests
"--- Test 1: (correct)"
CID "AB1"
"--- Test 2: (correct)"
CID "AB123"
"--- Test 3: (correct)"
"AB123" | CID
"--- Test 4: (wrong)"
"CD124" | CID
"--- Test 5: (wrong)"
"AB12456" | CID
```

Wie die nachstehende Bildschirmabbildung zeigt, schlagen die Tests 4 und 5 fehl, weil die PowerShell Verletzungen der für den Parameter `CustomerID` hinterlegten Regeln erkennt.

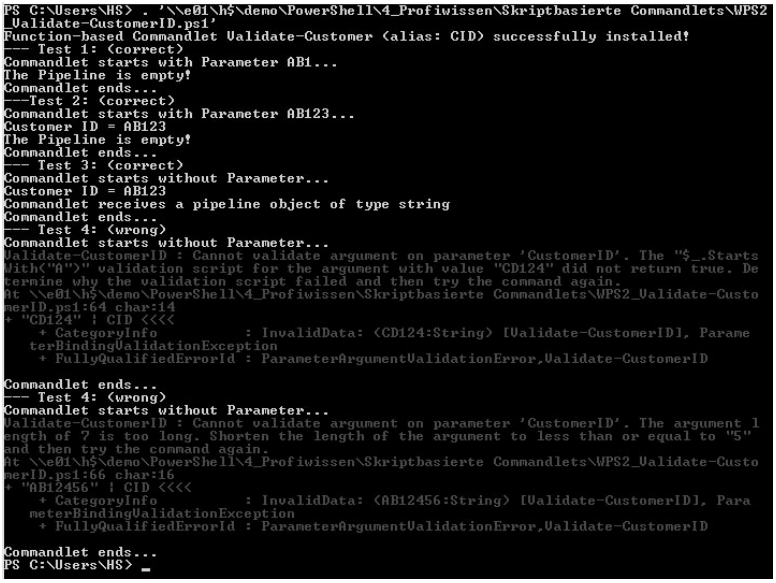


Abbildung 46.3
Ausgabe des obigen Skripts

46.4 Dokumentation

Advanced Functions erlauben eine eigene Dokumentationssyntax. Die Dokumentation steht in einem <# #>-Block zu Beginn des Funktionsrumpfs. Die Dokumentation kennt verschiedene Sektionen (`.Synopsis`, `.Description`, `.Parameter`, `.Notes`, `.Example`). `Get-Help` wertet diese Sektionen aus, wenn ein Nutzer Hilfe zu dem funktionsbasierten Commandlet aufruft.

Beispiel: Validate-CustomerID mit Dokumentation

Das folgende Listing zeigt `Validate-CustomerID` mit zusätzlicher Dokumentation.

```
# -------------------------------------------------------------------
# Author:      Dr. Holger Schwichtenberg
# Description: Demo for a function based Commandlet
# Usage:       This file contains an advanced function. In order
#              to use it, you must include the file into your shell:
#              PS> . c:\bin\Validate-Customer.ps1
# Date:        09/25/2006
# Website:     http://www.Windows-Scripting.com/PowerShell
# -------------------------------------------------------------------
```

Listing 46.5
Validate-CustomerID mit Dokumentation

```
# Function

function Validate-CustomerID {
<#
    .Synopsis
    Validates if a given string is a valid customer ID

    .Description
The commandlets implements logic for checking if the customer ID
exists.

    Return values are: $true or $false

.Parameter CustomerID
A Customer ID has two letters and one to three digits.

    .Notes
    This commandlet is a demo only.

    .Example
    C:PS> Get-CustomerID AB123

#>
[CmdletBinding()]
param(
[Parameter(Position=0,Mandatory=$true,
ValueFromPipeline=$true,HelpMessage="A Customer ID das two letters
and one to three digits.")]
[AllowNull()]
[ValidateLength(3,5)]
[ValidatePattern("[A-Z][A-Z][0-9]{1,3}")]
[ValidateScript({$_.StartsWith("A")})]
[string[]]
$CustomerID)

begin {

if ($CustomerID -eq $null)
{Write-Verbose "Commandlet starts without Parameter..." }
else
{Write-Verbose "Commandlet starts with Parameter $CustomerID ..." }
}

process {
Write-Verbose "Customer ID = $CustomerID"
if ($_)
{
Write-Verbose ("Commandlet receives a pipeline object of type " +
$_.GetType())
}
else
{
```

```
Write-Verbose "The Pipeline is empty!"
}
# Add Logic here (e.g. check database)
return $true
}

end {
"Commandlet ends..."

}
}
```

Den Vorteil der Commandlet-Dokumentation belegen die beiden nachfolgenden Bildschirmabbildungen.



Abbildung 46.4
Anzeige von Get-Help für Validate-Customer ohne Dokumentation

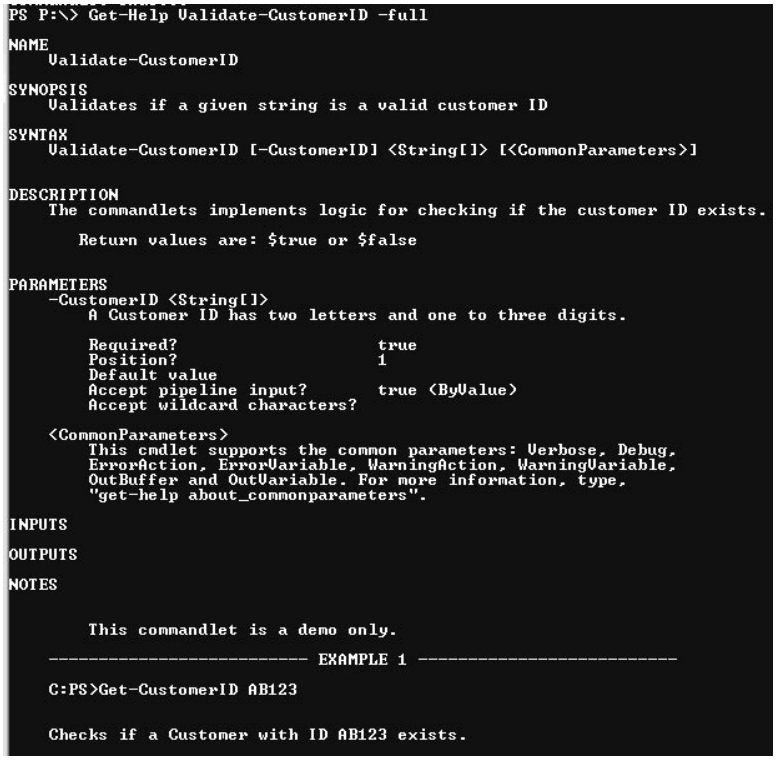


Abbildung 46.5
Anzeige von Get-Help für Validate-Customer mit Dokumentation

47 Entwicklung eigener Commandlets mit C#

Statt der auf PowerShell-Skriptfunktionen basierten Commandlets kann man Commandlets auch in einer beliebigen .NET-Programmiersprache erstellen und in kompilierter Form als eine DLL (alias „Snap-In“) verbreiten.



.NET-basierte Commandlets sind aufwändiger als die auf PowerShell-Skripten basierenden Commandlets, bieten aber mehr Optionen und den Vorteil, dass sie mit den komfortablen Möglichkeiten der Entwicklungsumgebung Visual Studio erstellt und in kompilierter Form verbreitet werden können.



Dieses Unterkapitel verlangt Grundkenntnisse in den Programmiersprachen C# oder Visual Basic .NET. Leider kann dieses Buch diese Grundkenntnisse nicht vermitteln, denn Grundlagenbücher zu Visual Basic .NET oder C# sind zwischen 500 und 1000 Seiten dick. Unmöglich kann dieser Stoff mit in dieses Buch aufgenommen werden. Dieses Kapitel dürfen Sie also nur als Ergänzung zu den dicken Grundlagenbüchern sehen, die üblicherweise nicht das Erstellen von PowerShell-Commandlets behandeln.



Grundsätzlich können Commandlets auch mit Visual Basic .NET, C++/CLI oder anderen .NET-Sprachen erstellt werden. Leider ist in diesem Buch kein Raum, die Beispiele in mehr als einer .NET-Programmiersprache abzudrucken. Der Autor hat sich für C# entschieden.

47.1 Technische Voraussetzungen

Entwicklungsumgebung

Zum Erstellen eines Commandlets sollten Sie eine Variante von Visual Studio 2005 oder Visual Studio 2008 oder Visual Studio 2010 verwenden. Es reicht dabei, wenn Sie eine der kostenfreien Express-Varianten (also Visual C# Express, Visual C++ Express oder Visual Basic Express) nutzen. Grundsätzlich ist eine Erstellung von Commandlets auch mit einem einfachen Texteditor und den im .NET Framework mitgelieferten Kommandozeilencompilern möglich. Dies ist jedoch deutlich mühsamer als die Verwendung von Visual Studio.



Mit Visual Web Developer Express (VWD) können Sie keine Commandlets, sondern nur Webanwendungen erstellen.



Aufgrund der vielen Übersetzungsfehler in der deutschen Version von Visual Studio verwendet der Autor nur die englische Version der Entwicklungsumgebung und alle Angaben zu Menüpunkten beziehen sich auf die englische Version.

Vorlagen Auf der CD-ROM zu diesem Buch sind Projektvorlagen für Visual Studio zur Erstellung von Commandlets enthalten. Diese ersparen ebenfalls Arbeit und sollten nach der Installation von Visual Studio zusätzlich aktiviert werden. Es gibt zwei Vorlagen:

- *Windows PowerShell (CS).vsi* enthält die Vorlage für die Erstellung von Commandlets in C#.
- *Windows PowerShell (VB).vsi* enthält die Vorlage für die Erstellung von Commandlets in Visual Basic .NET.

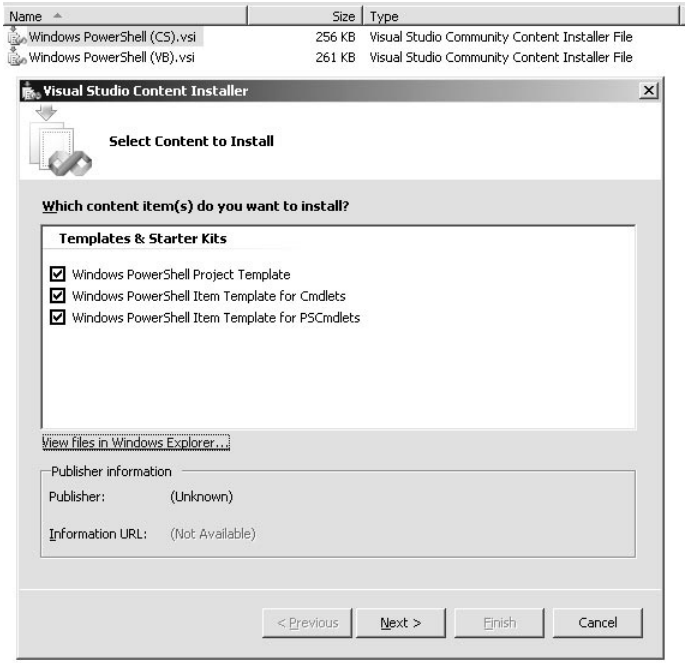


Abbildung 47.1 *Installation der Visual-Studio-Projektvorlagen zur Erstellung von Commandlets*

47.2 Grundkonzept der .NET-basierten Commandlets

.NET-basierte Commandlets werden in Form von Commandlet-Klassen im Rahmen eines sogenannten PowerShell-Snap-In erstellt.

Commandlet-Klassen

Cmdlet oder PSCmdlet

Jedes .NET-basierte Commandlet ist eine öffentliche .NET-Klasse (deklariert als `public`), die von einer bestimmten Basisklasse (`Cmdlet` oder `PSCmdlet`, beide im Namensraum `System.Management.Automation`) erbt und in einer DLL kompiliert wird. Die Commandlet-Klasse muss neben der Basisklasse auch noch eine Annotation `[Cmdlet]` besitzen, in welcher der Verbname und der Substantivname des Commandlets definiert werden. Der Name der .NET-Klasse ist für die Sichtbarkeit und die Anzeige in der PowerShell nicht maßgeblich.



Die Basisklasse `PSCmdlet` bietet für das Commandlet mehr Möglichkeiten als `Cmdlet`, z.B. den Zugriff auf die in der PowerShell deklarierten Variablen. Commandlets, die von `Cmdlet` abgeleitet sind, haben aber den Vorteil, dass sie auch außerhalb der Windows PowerShell genutzt werden können. Commandlet-Klassen, die von `PSCmdlet` erben, funktionieren hingegen nur in der PowerShell.

Methoden

Überschreiben der Methoden der Basisklasse

Die Commandlet-Klasse muss mindestens eine der Methoden durch Überschreiben der gleichnamigen Methode der Basisklasse implementieren und mit Code ausstatten:

- `BeginProcessing()`: wird genau einmal zu Beginn der Verarbeitung aufgerufen (vgl. „begin“ bei funktionsbasierten Commandlets)
- `ProcessRecord()`: wird einmal für jedes Objekt in der Pipeline aufgerufen. Bei leerer Pipeline wird `ProcessRecord` jedoch auch genau einmal aufgerufen. (vgl. „process“ bei funktionsbasierten Commandlets)
- `EndProcessing()`: wird genau einmal am Ende der Verarbeitung aufgerufen (vgl. „end“ bei funktionsbasierten Commandlets)
- `StopProcessing()`: wird nur dann einmal aufgerufen, wenn der Anwender den Abbruch der Verarbeitung anfordert



Mindestens eine der drei ersten Methoden muss implementiert sein.

Zusammenfassend gibt es also folgende Anforderungen an die Commandlet-Klasse:

- Die Klasse erbt von `System.Management.Automation.Cmdlet`.
- Die Klasse ist öffentlich.
- Das Verb-Substantiv-Paar, auf welches das Commandlet reagiert, gibt der PowerShell die Klasse über eine Annotation `[Cmdlet ("Verb","Substantiv")]` bekannt.
- Die Klasse implementiert mindestens eine der Methoden `BeginProcessing()`, `ProcessRecord()` oder `EndProcessing()`.

Ausgabe des Commandlets

WriteObject()

Ein Commandlet kann in jeder der Methoden die Ausgabepipeline mit einzelnen Objekten befüllen. Hierzu kommt die aus der Basisklasse geerbte Methode `WriteObject()` zum Einsatz.

Commandlets erzeugen die Ausgabe in `ProcessRecord()`, wenn die einzelnen Eingabeobjekte unabhängig voneinander sind. Ein Beispiel dafür ist ein Filter-Commandlet wie `Where-Object`, in dem jedes einzelne Objekt für sich geprüft wird und die Reihenfolge der Ausgabeobjekte der Reihenfolge der Eingabeobjekte entspricht. Dagegen erzeugen Commandlets die Ausgabe in `EndProcessing()`, wenn die Ausgabe von allen (!) Eingabeobjekten abhängig ist, z.B. `Sort-Object`, wo das letzte Eingabeobjekt das erste Ausgabeobjekt sein kann.



Die Ausgabeobjekte des Commandlets dürfen nicht mit dem C#-Sprachkonstrukt `return` an den PowerShell Pipeline Processor übergeben werden.

Snap-In-Klasse

PSSnapIn oder CustomPSSnapIn

Mehrere Commandlets können zu einer DLL zusammengefasst werden. Neben den Commandlet-Klassen muss es in der DLL auch noch eine Snap-In-Klasse (Basisklasse: `PSSnapIn` oder `CustomPSSnapIn`) geben.

Während die meisten .NET-basierten Anwendungen und .NET-Softwarekomponenten nicht mehr mit Einträgen in der Registrierungsdatenbank arbeiten und daher durch einfaches Kopieren verbreitet werden können (das sogenannte „XCopy-Deployment“), erfordert ein PowerShell-Snap-In Einträge in der Registrierungsdatenbank. Nach dem Kompilieren muss die DLL also erst in der PowerShell installiert werden. Dazu kann man ein MSI-Paket oder das Kommandozeilenwerkzeug *installutil.exe* verwenden. Nach der Installation ist ein weiterer Schritt notwendig, nämlich die Aktivierung des Snap-In in der PowerShell-Konsole mit `Add-PsSnapin`.

47.3 Schrittweise Erstellung eines minimalen Commandlets

Die Erstellung des ersten einfachen Commandlets wird relativ ausführlich beschrieben. Aus Platzgründen wird dann bei den weiteren Commandlets auf die schrittweisen Erläuterungen verzichtet.

Anlegen eines Visual-Studio-Projekts

Neues Projekt

Wenn Sie eine Variante von Visual Studio 2005 oder Visual Studio 2008 installiert und zusätzlich die PowerShell-Vorlagen eingefügt haben, erscheint beim Anlegen eines neuen Projekts die Option „Windows PowerShell“ (siehe *Abbildung 47.2*).

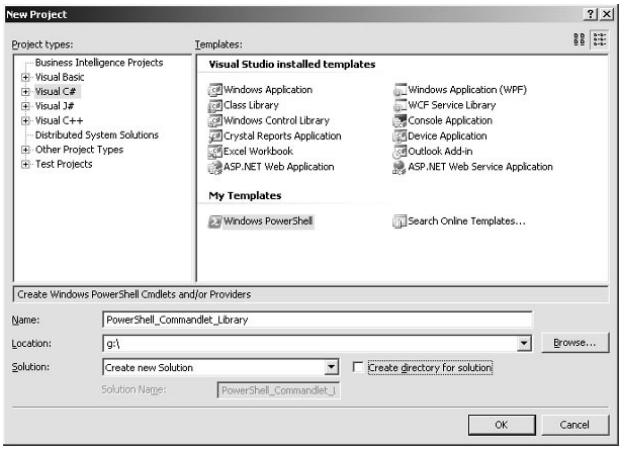


Abbildung 47.2
Anlegen eines neuen PowerShell-Projekts

Standardinhalte

Danach erscheint ein Visual-Studio-DLL-Projekt, das bereits folgende wichtige Vorstellungen enthält:

- Eine Referenz zu der Assembly *System.Management.Automation.dll*
- Eine Datei *PSSnapin.cs*, welche die Snap-In-Klasse realisiert

Das folgende Listing zeigt einen Ausschnitt aus dem Standardinhalt der Datei *PSSnapin.cs*. Sie können die Namen anpassen.

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Management.Automation;
using System.ComponentModel;

namespace PowerShell_Commandlet_Library
{
 [RunInstaller(true)]
 public class PowerShell_Commandlet_LibrarySnapIn : PSSnapIn
Automatisch generierte Snap-In-Klasse
```

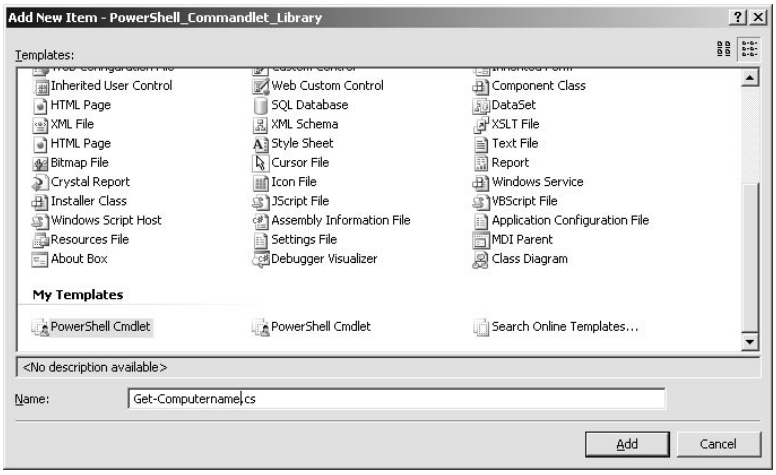


Wenn Sie die Projektvorlagen nicht verwenden wollen, können Sie, um zum gleichen Ziel zu kommen, auch selbst eine einfache Klassenbibliothek („Class Library"-Projekt) in Visual Studio anlegen, die Referenz auf die oben genannte Assembly einfügen und den Programmcode für die Snap-In-Klasse in Ihr Projekt aufnehmen.

Anlegen des Commandlets

Noch nicht in dem Projekt enthalten ist eine Vorlage für ein Commandlet. Wählen Sie dazu *Add/New Item* und dann *PowerShell Cmdlet*. Geben Sie „Get-Computername" als Namen an.

Abbildung 47.3 *Anlegen eines Commandlets*



Visual Studio erzeugt dann eine Datei *Get-Computername.cs* mit dem nachstehend dargestellten Inhalt.

Listing 47.1 *Standardinhalt der Elementvorlage für Commandlet-Klassen*

```
using System;
using System.Text;
using System.Management.Automation;
using System.Management.Automation.Provider;
namespace PowerShell_Commandlet_Library
{
 #region Get_ComputernameCommand

 /// <summary>
 /// This class implements the Get_Computername cmdlet
 /// </summary>
 [Cmdlet(VerbsCommon.Get, "",
    DefaultParameterSetName = "")]
 public class Get_Computername : PSCmdlet
 {
  #region Parameters
```

```
  //    [Parameter(
  //       Position = 0,
  //       ParameterSetName = "ID",
  //       ValueFromPipeline = true,
  //       ValueFromPipelineByPropertyName = true)]
  //    [ValidateNotNullOrEmpty]
  //    public string[] ID
  //    {
  //        get { return _ID; }
  //        set { _ID = value; }
  //    }

  #endregion Parameters

  #region Cmdlet Overrides
  protected override void BeginProcessing()
  {

  }

  protected override void ProcessRecord()
  {

  }

  #endregion Overrides
  #region Private Data


        #endregion Private Data

 } //Get_ComputernameComand

 #endregion
}
```

Implementierung des Commandlets

Get-Computername

Das folgende Listing zeigt eine einfache Implementierung von `Get-Computername` in der Programmiersprache C#. Aufgabe des Commandlets ist es, den Namen des Computers zu ermitteln und diesen als eine Zeichenkette an die PowerShell zu übergeben. Diese Information findet man in der .NET-Klasse `System.Environment.MachineName`. Das Commandlet liefert als Rückgabe eine Instanz der Klasse `System.String` (also eine Zeichenkette).

Neben den notwendigen Pflichtangaben für eine Commandlet-Klasse enthält das Listing nur eine einzige Zeile, in welcher der Computername mit `WriteObject()` in die Ausgabepipeline gelegt wird.



In diesem Beispiel macht es keinen Unterschied, ob die Ausgabe in `ProcessRecord()` oder `EndProcessing()` erzeugt wird. Sogar `BeginProcessing()` wäre möglich.

Listing 47.2 *Implementierung des Commandlets Get-Computername*

```
using System;
using System.Text;
using System.Management.Automation;

namespace de.ITVisions.PowerShell
{

 /// <summary>
 /// Commandlet-Klasse für das Commandlet Get-Computername
 /// </summary>
 [Cmdlet(VerbsCommon.Get, "Computername")]
 public class GetComputernameCommand : PSCmdlet
 {

  protected override void BeginProcessing()
  {
  }

  protected override void ProcessRecord()
  {
  }

  protected override void EndProcessing()
  {
   this.WriteObject(System.Environment.MachineName);
  }

  protected override void StopProcessing()
  {
  }
 }
}
```



Sie können den Inhalt der Datei *Get-Computername.cs* entweder komplett löschen und an dieser Stelle das nachstehende Listing eintragen oder aber die bestehende Datei anpassen. Im Prinzip sind nur die fett markierten Zeilen zu übertragen. (Sollten Sie übrigens keine fett markierten Zeilen im Buch sehen, hat der Setzer wieder einmal eine Formatierung nicht beachtet.) Die leeren Methoden könnten Sie weglassen.

Kompilieren des Commandlets

Nach dem Kompilieren der DLL mit der Funktion *Build Solution* sollte Visual Studio „1 succeeded“ melden (siehe *Abbildung 47.4*).

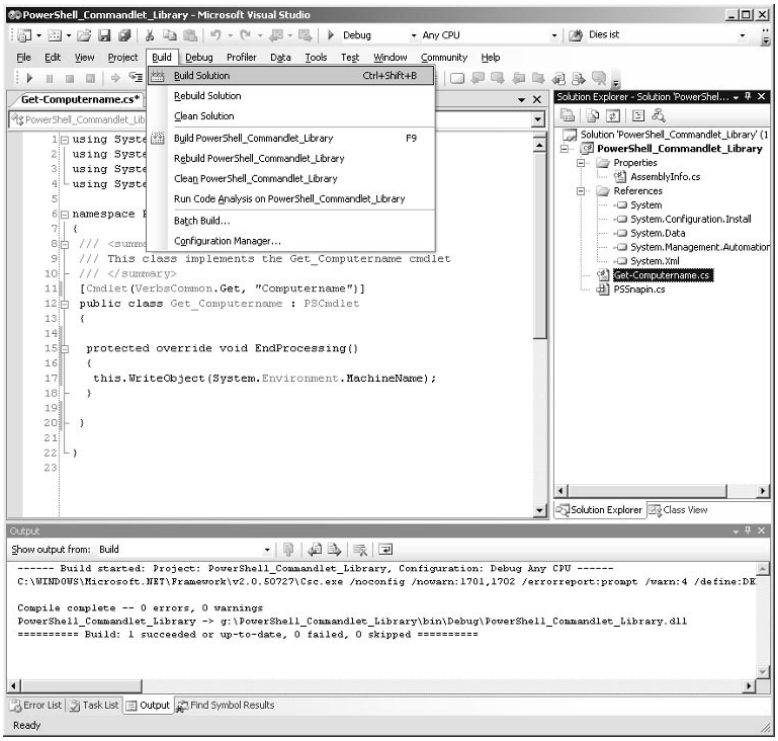


Abbildung 47.4 *Kompilieren des Commandlet-Projekts in Visual Studio*

Installation des Commandlets

installutil.exe

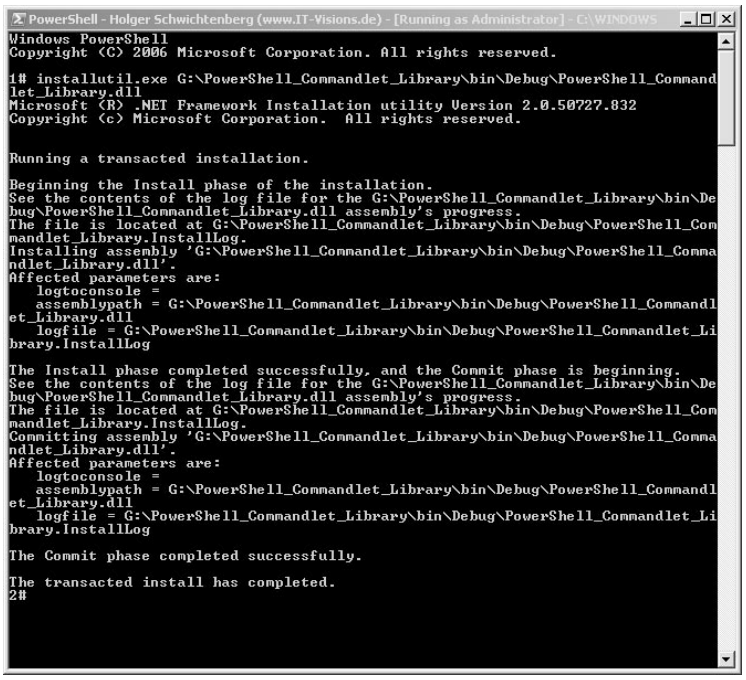
Bereits in *Teil 1 „PowerShell-Basiswissen“* wurde die Installation von PowerShell-Erweiterungen angesprochen. Um Platz in diesem Buch für andere Themen zu sparen, soll das Thema hier nur kurz wiederholt werden.

Folgende Schritte sind sowohl auf dem Entwicklungssystem als auch später auf allen Zielsystemen notwendig:

- Registrieren der DLL (alias Assembly), welche die Commandlets enthält

```
installutil.exe G:\PowerShell_Commandlet_Library\bin\release\
PowerShell_Commandlet_Library.dll
```

Abbildung 47.5
Ausgabe von InstallUtil.exe



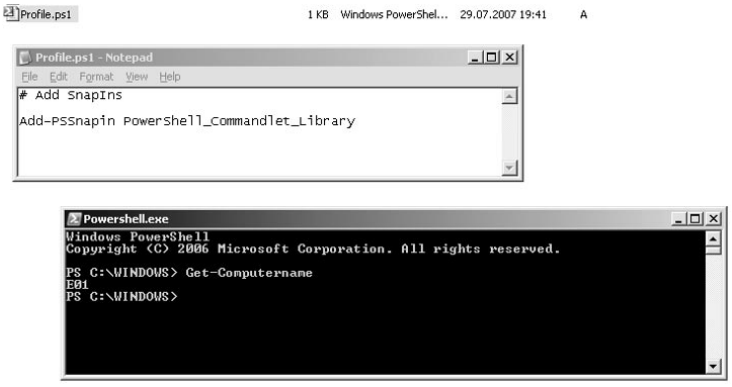
Add-PSSnapin ▶ Hinzufügen des Snap-In zur PowerShell-Konsole

```
Add-PSSnapin PowerShell_Commandlet_Library
```

Zum permanenten Laden der Erweiterung beim Start der Konsole hat man zwei Möglichkeiten:

- Aufnahme der entsprechenden `Add-PSSnapIn`-Anweisungen in Ihre systemweite oder benutzerspezifische Profildatei (*Profile.ps1*, siehe auch *Abschnitt 27.5 „Profileinstellungen für die PowerShell-Konsole“*)

Abbildung 47.6
Laden eines Snap-In in der Profildatei



Export-Console

- Exportieren einer Konsolenkonfigurationsdatei mit `Export-Console`. Sie müssen allerdings vorher erst das Snap-In in der aktuellen Konsole hinzufügen und dann diese aktuelle Konsole exportieren. Dabei entsteht eine XML-Datei mit der Dateinamenserweiterung *.psc1*. Diese *.psc1*-Datei muss dann beim Starten der PowerShell über den Kommandozeilenparameter *–PSConsoleFile* angegeben werden.

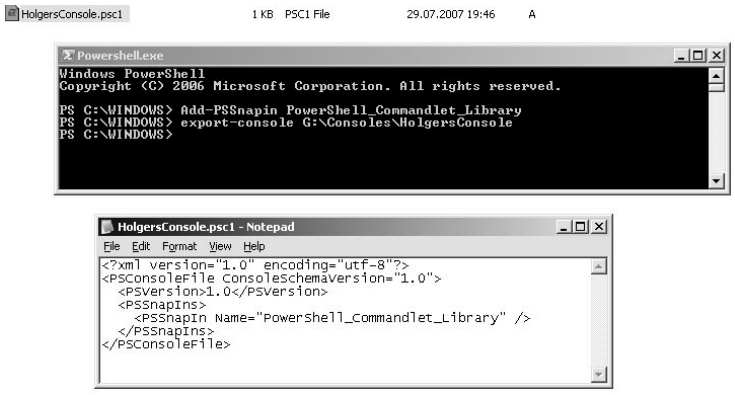


Abbildung 47.7 *Exportieren einer Konsolenkonfigurationsdatei*

Am besten legt man sich eine Verknüpfung im Dateisystem an mit folgendem Ziel:

```
%SystemRoot%\system32\WindowsPowerShell\v1.0\powershell.exe -
PSConsoleFile "G:\Consoles\HolgersConsole.psc1"
```

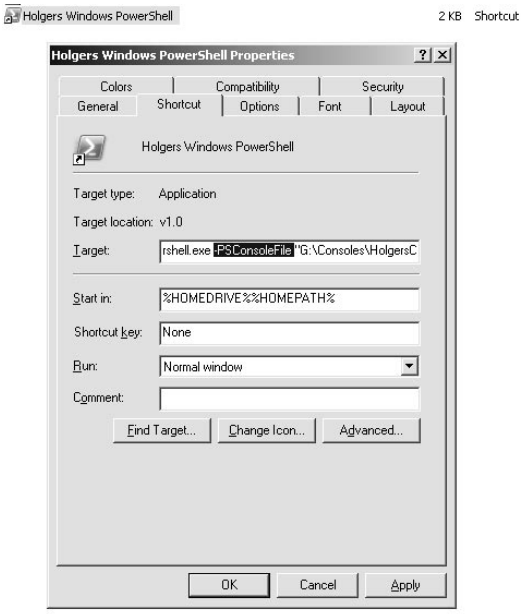
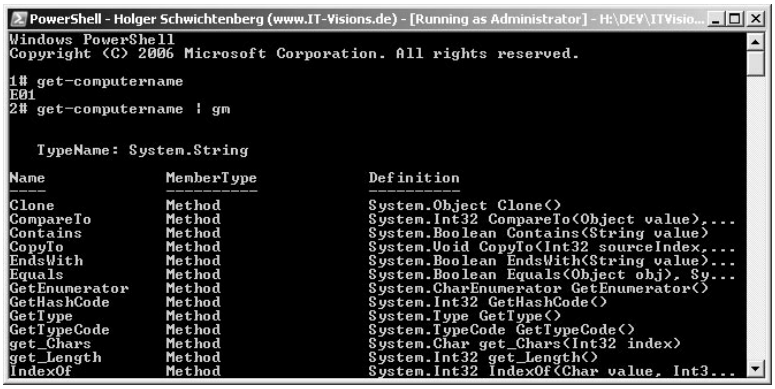


Abbildung 47.8 *Anlegen einer Verknüpfung zur PowerShell-Konsole, die automatisch eine bestimmte Konsolenkonfigurationsdatei mitlädt*

Verwendung des Commandlets

Die nachfolgende Bildschirmabbildung zeigt, dass das neue Commandlet sofort nach dem Start der PowerShell zur Verfügung steht, wenn die `Add-PSSnapIn`-Anweisung in die Profildatei aufgenommen oder eine Konsolendatei erstellt und eingebunden wurde.

Abbildung 47.9
Einsatz von Get-Computername



Weiterentwicklung der Commandlet-Erweiterungen

Weiteres Vorgehen

Im Folgenden werden Sie weitere Commandlet-Implementierungen kennenlernen, die Sie in eigenen Commandlet-Bibliotheken oder in derselben Commandlet-Bibliothek speichern können. Wenn Sie dieselbe Command-Bibliothek verwenden, müssen Sie einige Schritte nicht wiederholen. Sie müssen immer nur ein neues Commandlet anlegen, dieses implementieren und kompilieren. Die Installation und das Laden der Commandlet-Bibliothek entfallen, solange Sie nicht die Pfade im Dateisystem ändern.



Wenn Sie die Commandlet-Bibliothek erneut kompilieren, darf die Bibliothek in keinem Konsolenfenster geladen sein. Falls die Commandlet-Bibliothek geladen ist, scheitert der Übersetzungsvorgang mit dem Fehler:

"Error2Unable to copy file "obj\Debug\PowerShell_Commandlet_Library.dll" to "bin\ Debug\PowerShell_Commandlet_Library.dll". The process cannot access the file 'bin\ Debug\PowerShell_Commandlet_Library.dll' because it is being used by another process.PowerShell_Commandlet_Library"

47.4 Erstellung eines Commandlets mit einem Rückgabeobjekt

Commandlets geben in der Regel nicht einzelne elementare Typen wie Zeichenketten, sondern komplexere Objekte zurück. Ein Commandlet kann Instanzen jeder beliebigen .NET-Klasse in die Pipeline legen. Es gibt keine besonderen Voraussetzungen. Die Klasse der Ausgabeobjekte (alias Rückgabeobjekte) muss weder öffentlich noch serialisierbar sein.

Beispiel

Get-ComputerInfo

In dem folgenden Beispiel wird eine Datenstruktur mit Namen `ComputerInfo` durch das Commandlet `Get-ComputerInfo` geliefert. Die Datenstruktur ist in Form der selbst definierten Klasse `ComputerInfo` realisiert und wird mit Informationen aus verschiedenen Klassen der .NET-Klassenbibliothek (`System.Environment`, `System.Globalization.CultureInfo` und `System.Net.NetworkInformation.NetworkInterface`) „gefüttert".

Listing 47.3 Implementierung des Commandlets Get-ComputerInfo

```
using System;
using System.Text;
using System.Management.Automation;

namespace de.ITVisions.PowerShell
{

 internal class ComputerInfo
 {
  public string ComputerName;
  public int ProcessorCount;
  public string Domain;
  public string Culture;
  public bool Connected;
 }
 /// <summary>
 /// Commandlet-Klasse
 /// </summary>

 [Cmdlet(VerbsCommon.Get, "ComputerInfo")]

 public class GetComputerInfoCommand : PSCmdlet
 {

  protected override void BeginProcessing()
  {
  }

  protected override void ProcessRecord()
  {
```

```
  }

  protected override void EndProcessing()
  {
   ComputerInfo i = new ComputerInfo();
   i.ComputerName = Environment.MachineName;
   i.ProcessorCount = Environment.ProcessorCount;
   i.Domain = Environment.UserDomainName;
   i.Culture =
System.Globalization.CultureInfo.CurrentUICulture.ToString();
   i.Connected = System.Net.NetworkInformation.NetworkInterface.
                 GetIsNetworkAvailable();
   this.WriteObject(i);
  }

  protected override void StopProcessing()
  {
  }

 }
}
```



Bitte beachten Sie, dass zu diesem Beispiel nicht noch einmal alle Schritte zum Anlegen eines Projekts, Kompilieren und Installieren besprochen werden können. Sie können `Get-ComputerInfo` und alle weiteren Commandlets in das bestehende Projekt als zusätzliche C#-Quellcodedatei einfügen.

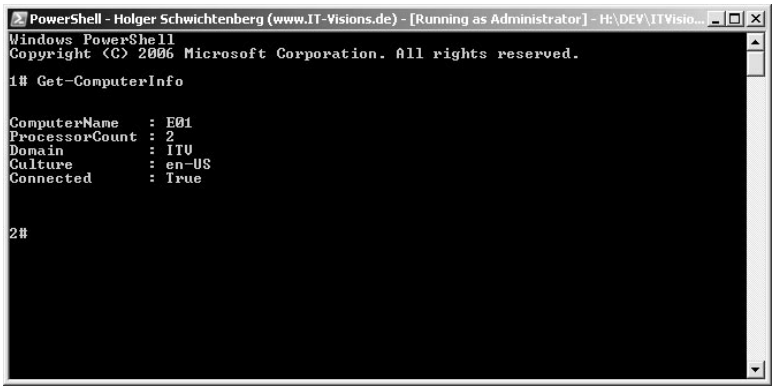


Abbildung 47.10
Einsatz von Get-ComputerInfo

47.5 Erstellung eines Commandlets mit mehreren Rückgabeobjekten

Get-Disk

Ein einzelnes Commandlet darf beliebig viele Objekte in die Ausgabepipeline legen. Das folgende Listing implementiert `Get-Disk`, ein Commandlet, das Informationen über die verfügbaren Laufwerke liefert. Diese Information gewinnt das Commandlet aus der Windows Management Instrumentation (WMI) in Form von Instanzen der WMI-Klasse `Win32_LogicalDisk`.



Bitte verwechseln Sie nicht `System.Management.Automation` und `System.Management`. Der Namensraum `System.Management.Automation` gehört zur Windows PowerShell und enthält Klassen für die PowerShell, z.B. die Basisklasse `PSCmdlet`. Der Namensraum `System.Management` gehört zum .NET Framework und enthält Klassen für den Zugriff auf WMI, z.B. `ManagementClass`, `ManagementObject`, `ManagementObjectCollection`, `ManagementObjectSearcher`. Hier muss sich Microsoft eine Namensgebung vorwerfen lassen, die Einsteigern das Leben nur erschwert. Nach einer Zeit des Arbeitens mit .NET hat man aber solche Ungereimtheiten verinnerlicht.

Aufspalten einer Menge

Die Instanzen der WMI-Klassen werden mit Hilfe einer WMI-Query-Language-(WQL-)Abfrage ermittelt und – genau wie die Einzelobjekte in den bisherigen Beispielen – durch `WriteObject()` in die Ausgabepipeline gelegt.

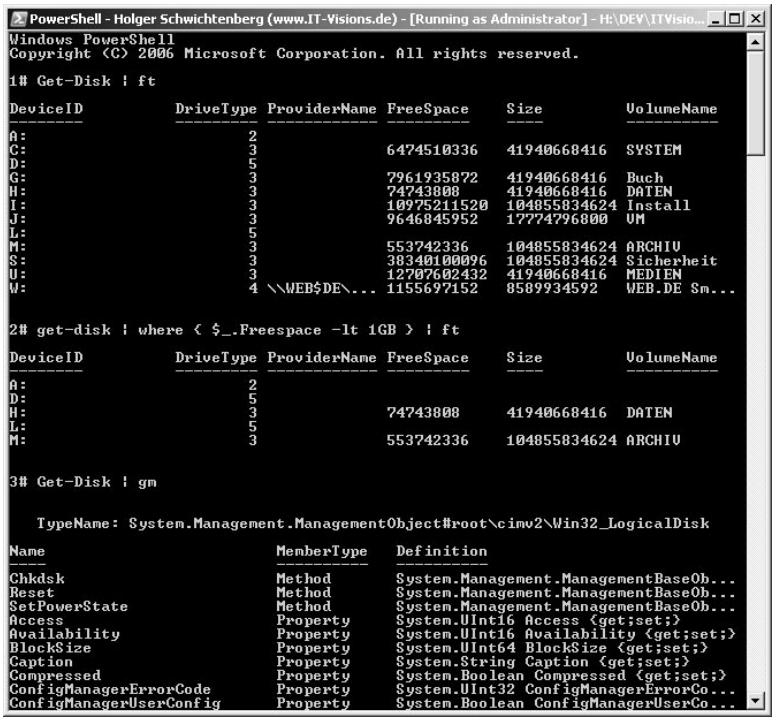
EnumerateCollection

Bei `WriteObject()` sieht man in dem Beispiel eine Besonderheit: Neben der zurückzugebenden Menge ist noch als zweiter Wert ein `true` für den Parameter `EnumerateCollection` angegeben. Dies bedeutet, dass nicht die Menge der WMI-Objekte als solches in die Pipeline gelegt werden soll, sondern jedes einzelne in der Menge enthaltene Objekt.

Die beiden folgenden Bildschirmabbildungen zeigen, dass auf den ersten Blick kein Unterschied besteht, aber bei weiterer Verwendung diese Option einen großen Unterschied macht. Die Eingabe `Get-Disk` liefert – sowohl für den Fall, dass `true` gesetzt wurde, als auch im Fall, dass `false` gesetzt würde, das gleiche Ergebnis, weil am Ende der Pipeline die Standardausgabe der PowerShell Objektmengen in ihre Einzelteile zerbricht. Wenn man allerdings vorher in der Pipeline die Objekte verarbeiten möchte, z.B. filtern mit `Where-Object`, dann funktioniert die Pipeline ohne das „true“ nicht wie gewünscht. Die Bildschirmabbildung zeigt, dass alle Laufwerke ungefiltert durch `Where-Object` gehen, auch wenn sie mehr als den in der Bedingung angegebenen Speicher von einem GB haben. Dies erklärt sich bei der Betrachtung des Pipeline-Inhalts mit `Get-Member`: Wenn `EnumerateCollection` auf `true` gesetzt

ist, dann liegen Instanzen der Klasse `System.Management.ManagementObject#root\cimv2\Win32_LogicalDisk` in der Pipeline. Wenn `false` oder kein Wert angegeben ist, dann liegt eine Instanz von `System.Management.ManagementObjectCollection` in der Pipeline. Die Klasse `System.Management.ManagementObjectCollection` besetzt aber kein Attribut `FreeSpace` und die meisten Commandlets (so auch `Where-Object`) spalten Mengen nicht selbstständig auf. In diesem Fall müsste der Nutzer erst die Menge mit `Foreach-Object` spalten und das wäre unschön und entspricht nicht der Philosophie der PowerShell.

Abbildung 47.11
Verwendung von Get-Disk bei richtiger Implementierung



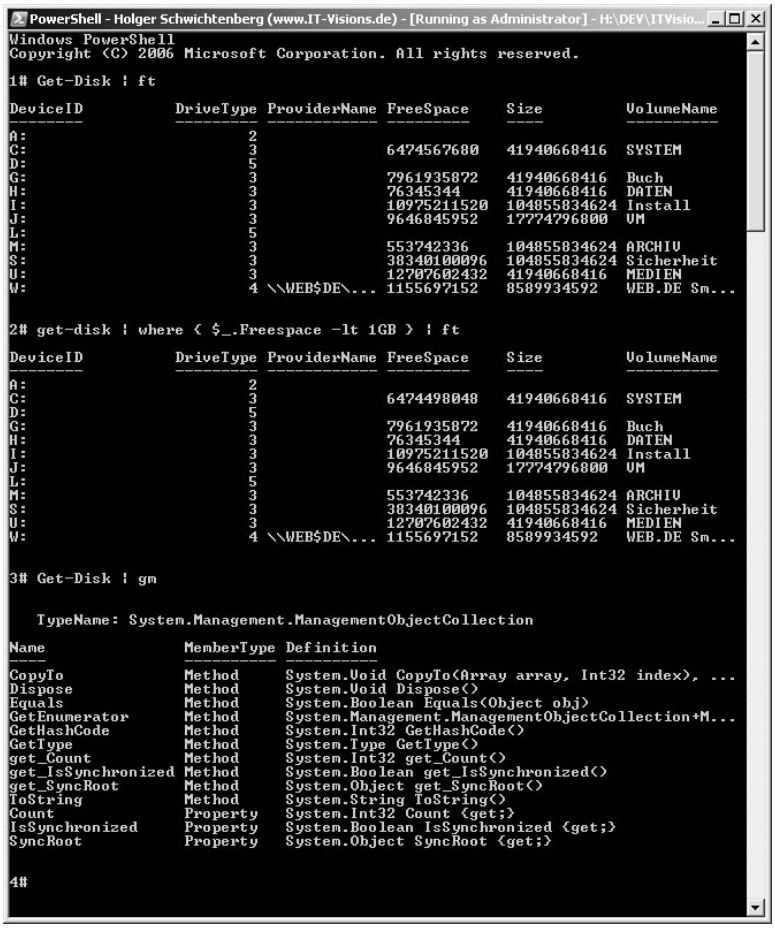


Abbildung 47.12
Verwendung von Get-Disk bei falscher Implementierung



Alternativ zum Befehl `this.WriteObject(DiskMenge, true)` könnte man auch die Objektmenge selbst mit einer `foreach`-Schleife innerhalb des Commandlets aufspalten:

```
foreach (ManagementObject Disk in DiskMenge)
{
this.WriteObject(Disk);
}
```

Dies führt zum exakt gleichen Ergebnis aus der Sicht des Nutzers des Commandlets. Der Entwickler hätte hiermit also nur mehr unnötige Arbeit. An dieser Stelle ist diese Option nur wiedergegeben, um das Verständnis für die Funktionsweise von `WriteObject()` zu fördern.

Listing 47.4 *Implementierung des Commandlets Get-Disk*

```
using System;
using System.Text;
using System.Management.Automation;
using System.Management.Automation.Provider;
using System.Management;

namespace de.ITVisions.PowerShell
{

 [Cmdlet(VerbsCommon.Get, "Disk")]
 public class GetDiskCommand : PSCmdlet
 {

  protected override void BeginProcessing()
  {
  }

  protected override void ProcessRecord()
  {
   string ABFRAGE = "Select * from Win32_LogicalDisk";
   string Computer = ".";

   System.Management.ManagementObjectCollection DiskMenge;
   ManagementScope ms = new ManagementScope("\\\\" + Computer +
                                             "\\root\\cimv2");
   ObjectQuery oq = new ObjectQuery(ABFRAGE);
   System.Management.ManagementObjectSearcher suche =
          new System.Management.ManagementObjectSearcher(ms, oq);
   DiskMenge = suche.Get();

   // Objekte einzeln zurückgeben
   this.WriteObject(DiskMenge, true);

   // Alternative
   //foreach (ManagementObject Disk in DiskMenge)
   //{
   // this.WriteObject(Disk);
   //}
  }

  protected override void StopProcessing()
  {
  }

 }
}
```

47.6 Erstellen eines Commandlets mit Parametern

Alle bisherigen Commandlets besitzen überhaupt keine Parameter. Am Beispiel von `Get-Disk` soll die Parametrisierung gezeigt werden. Dies ist einfach realisierbar, da WMI den Fernzugriff auf andere Computer über das Protokoll Distributed Component Object Model (DCOM) unterstützt. An der WQL-Abfrage muss nichts verändert werden, es muss nur dafür gesorgt werden, dass der Computername nicht statisch im Programmcode gesetzt wird (bisher war das ein Punkt „.“ für den lokalen Computer), sondern von der PowerShell-Konsole als Parameter empfangen wird.

[Parameter]

Dazu definiert man `Computer` nicht mehr als lokale Variable in `ProcessRecord()`, sondern als ein Attribut der Commandlet-Klassen. Damit dieses Attribut von der PowerShell mit Parametern befüllt werden kann, muss die Annotation `[Parameter]` hinzugefügt werden. Außerdem muss das Attribut öffentlich (`public`) sein.

```
[Parameter()]
public string Computer = ".";
```

Die obige Implementierung ist ausreichend, erzwingt aber immer einen benannten Parameter der Form:

```
Get-Disk -Computer E02
```

Position

Soll der Anwender den Parameternamen auch weglassen können, muss der Entwickler des Commandlets eine Positionsangabe ergänzen, die wichtig ist, um bei mehreren unbenannten Parametern die Positionszuordnung zu machen.

```
[Parameter(Position = 0)]
public string Computer = „.";
```

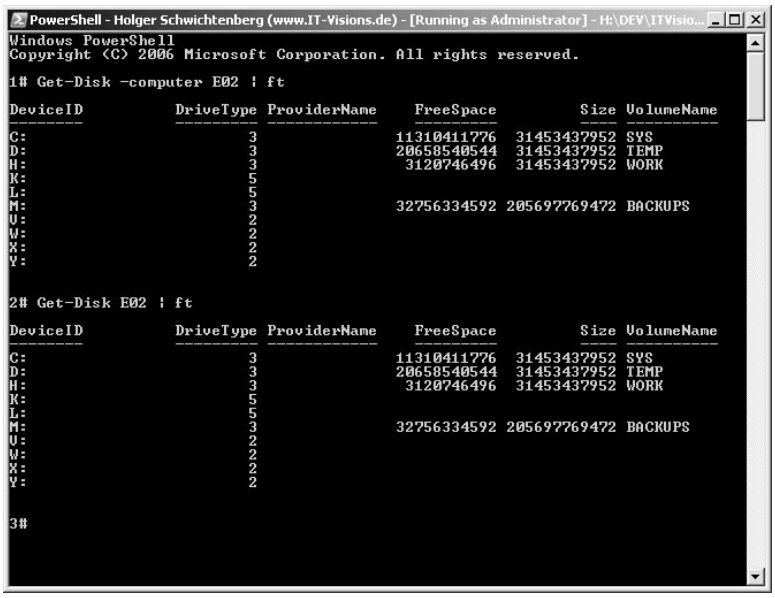
Die folgende Bildschirmabbildung zeigt, dass nun `Get-Disk` bei sowohl benannten als auch unbenannten Parametern funktioniert und auf andere Systeme zugreifen kann.



Die Standardvorgabe (`= "."`, wobei ein Punkt in WMI für den lokalen Computer steht) sorgt dafür, dass beim Weglassen eines Werts für diesen Parameter der Zugriff auf das lokale System erfolgt.

Alternativ kann der Parameter durch den Zusatz `Mandatory = true` in der Annotation als Pflichtattribut deklariert werden. Dann fordert die PowerShell bei ihrem Benutzer einen Wert an, falls er den Parameter weglässt.

Abbildung 47.13
Get-Disk mit Parameter



Listing 47.5
Variante der Implementierung von Get-Disk mit einem Parameter

```
using System;
using System.Text;
using System.Management.Automation;
using System.Management.Automation.Provider;
using System.Management;

namespace de.ITVisions.PowerShell
{
 [Cmdlet(VerbsCommon.Get, "Disk")]
 public class GetDiskCommand : PSCmdlet
 {

  [Parameter(Position = 0)]
  public string Computer = ".";

  protected override void BeginProcessing()
  {
  }

  protected override void ProcessRecord()
  {
   string ABFRAGE = "Select * from Win32_LogicalDisk";

   System.Management.ManagementObjectCollection DiskMenge;
   ManagementScope ms = new ManagementScope("\\\\" + Computer +
                                            "\\root\\cimv2");
   ObjectQuery oq = new ObjectQuery(ABFRAGE);
   System.Management.ManagementObjectSearcher suche =
```

```
      new System.Management.ManagementObjectSearcher(ms, oq);
   DiskMenge = suche.Get();

   // Objekte einzeln zurückgeben
   this.WriteObject(DiskMenge, true);

   // Alternative
   //foreach (ManagementObject Disk in DiskMenge)
   //{
   // this.WriteObject(Disk);
   //}
  }
  protected override void StopProcessing()
  {
  }
 }
}
```

47.7 Verarbeiten von Pipeline-Eingaben

Viele eingebaute Commandlets können Werte aus der Pipeline beziehen. Für das selbst definierte Commandlet `Get-Disk` würde man sich wünschen, den Namen eines Computers und auch die Namen mehrerer Computer über die Pipeline zu übergeben, z.B.:

```
"E02" | Get-Disk
```

```
"E01", "E02", "E04" | Get-Disk
```

Die Unterstützung für Parameter bedeutet nicht automatisch auch, dass ein Commandlet Eingaben von der Pipeline akzeptiert, wie die Fehlermeldungen in *Abbildung 47.14* zeigen.

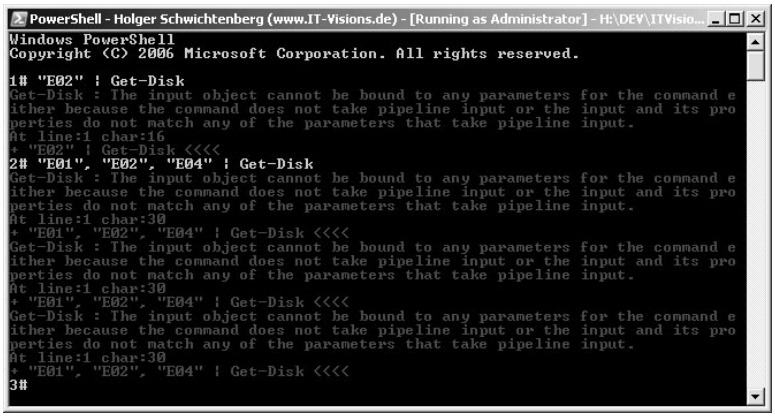


Abbildung 47.14 *Get-Disk akzeptiert keine Pipeline-Inhalte – weder einzelne Zeichenketten noch Mengen.*

ValueFrom-Pipeline

Die Unterstützung für Eingabeobjekte aus der PowerShell-Pipeline ist jedoch sehr einfach herstellbar, indem man die Annotation `[Parameter]` um `ValueFromPipeline = true` ergänzt:

```
[Parameter(Position = 0, ValueFromPipeline = true)]
public string Computer = ".";
```

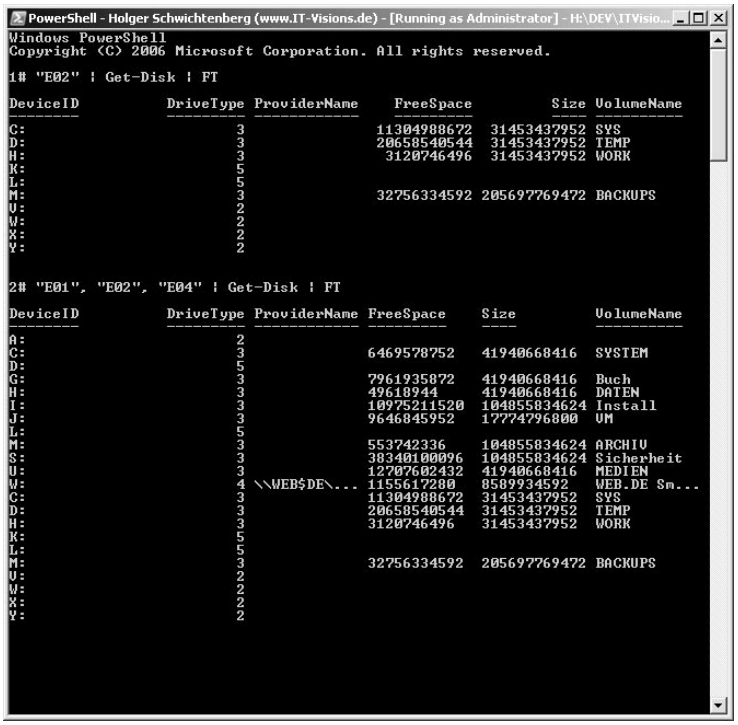


Abbildung 47.15 *Get-Disk akzeptiert nun Pipeline-Inhalte.*

Listing 47.6 *Variante der Implementierung von Get-Disk mit Akzeptanz von Pipeline-Inhalten*

```
using System;
using System.Text;
using System.Management.Automation;
using System.Management.Automation.Provider;
using System.Management;

namespace de.ITVisions.PowerShell
{


 [Cmdlet(VerbsCommon.Get, "Disk")]
 public class GetDiskCommand : PSCmdlet
 {
  [Parameter(Position = 0, ValueFromPipeline = true)]
  public string Computer = ".";

  protected override void BeginProcessing()
```

```
  {
  }

  protected override void ProcessRecord()
  {
   string ABFRAGE = "Select * from Win32_LogicalDisk";

   System.Management.ManagementObjectCollection DiskMenge;
   ManagementScope ms = new ManagementScope("\\\\" + Computer +
                                            "\\root\\cimv2");
   ObjectQuery oq = new ObjectQuery(ABFRAGE);
   System.Management.ManagementObjectSearcher suche =
      new System.Management.ManagementObjectSearcher(ms, oq);
   DiskMenge = suche.Get();

   // Objekte einzeln zurückgeben
   this.WriteObject(DiskMenge, true);

   // Alternative
   //foreach (ManagementObject Disk in DiskMenge)
   //{
   // this.WriteObject(Disk);
   //}
  }

  protected override void StopProcessing()
  {
  }

 }
}
```



Alternativ zur Angabe `ValueFromPipeline = true` kann man durch `ValueFromPipelineByPropertyName = true` festlegen, dass nicht ein ganzes Objekt aus der Eingabepipeline auf ein Attribut der Commandlet-Klasse abgebildet wird, sondern nur ein einzelnes gleichnamiges Attribut des Eingabeobjekts.

47.8 Verkettung von Commandlets

Drei Kopplungsarten

Ihre ganze Macht können Commandlets im Zusammenspiel mit anderen Commandlets entfalten. Es gibt drei Kopplungsformen:

- Kopplung auf Basis elementarer Datentypen (insbesondere String)
- Kopplung auf Basis von typisierten Objekten
- Generische Kopplung

Kopplung auf Basis elementarer Datentypen

Die Kopplung auf Basis elementarer Datentypen (insbesondere String) ist die einfachste Verbindungsform, die aber die Möglichkeiten der PowerShell nicht ausschöpft. Viele Commandlets nehmen Eingaben als Zeichenkette entgegen und auch viele klassische Kommandozeilenwerkzeuge können damit arbeiten.

Die Akzeptanz von Pipeline-Inhalten funktioniert natürlich nur, wenn die Pipeline auch sinnvolle Daten enthält. Der Befehl `Get-Process | Get-Disk` würde zwar von der PowerShell ausgeführt werden, es würde aber so viele Fehler geben, wie es Prozesse gibt, denn `Get-Disk` würde Computer suchen, die „System.Diagnostics.Process (xy)“ heißen, wobei xy der Prozessname ist.

Umwandlung in Zeichenketten

In diesem Fall macht es keinen Sinn, die beiden Commandlets zu koppeln. Da es grundsätzlich aber sein könnte, dass ein Commandlet sinnvolle Textdaten für ein anderes Commandlet enthält, stellt sich die Frage, woher der Text „System.Diagnostics.Process (xy)“ kommt. Hinter den Kulissen passiert Folgendes: `Get-Process` legt Instanzen von `System.Diagnostics.Process` in die Pipeline und übergibt diese nacheinander an `Get-Disk`. `Get-Disk` erwartet aber eine Zeichenkette und die PowerShell konvertiert nun mit Hilfe der Methode `ToString()` die Prozessobjekte in Zeichenketten. Im Fall von `System.Diagnostics.Process` werden der Klassenname und der Prozessname ausgegeben. Dies kann man leicht mit `gps | foreach { $_.tostring() }` ermitteln (siehe *Abbildung 47.16*).

Jedes .NET-Objekt besitzt eine Methode `ToString()`, denn `ToString()` ist in der „Mutter aller .NET-Klassen“ `System.Object` implementiert und wird an alle .NET-Klassen und somit auch deren Instanzen weitergegeben. Ob `ToString()` eine sinnvolle Ausgabe liefert, hängt von der jeweiligen Klasse ab. Bei der Klasse `System.ServiceProcess.ServiceController`, deren Instanzen von `Get-Service` geliefert werden, ist die Konvertierung hingegen nicht so gut, denn die Zeichenkette enthält nur den Klassennamen, so dass die einzelnen Instanzen gar nicht unterschieden werden können.



Die Konvertierung in den Klassennamen ist das Standardverhalten, das von `System.Object` geerbt wird, und dieses Standardverhalten ist leider auch üblich, da sich die Entwickler der meisten .NET-Klassen bei Microsoft nicht die „Mühe“ gemacht haben, eine sinnvolle Zeichenkettenrepräsentanz zu definieren.

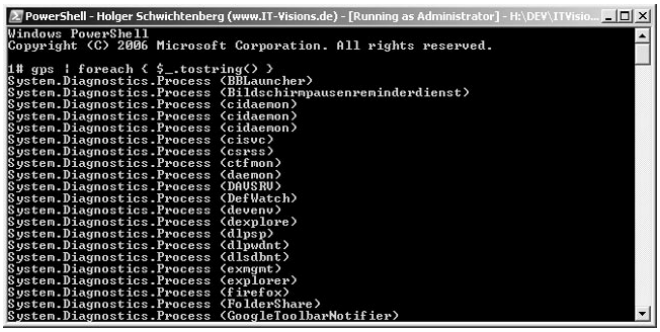


Abbildung 47.16
Anwendung von ToString() auf Instanzen der Klasse System.Diagnostics.Process

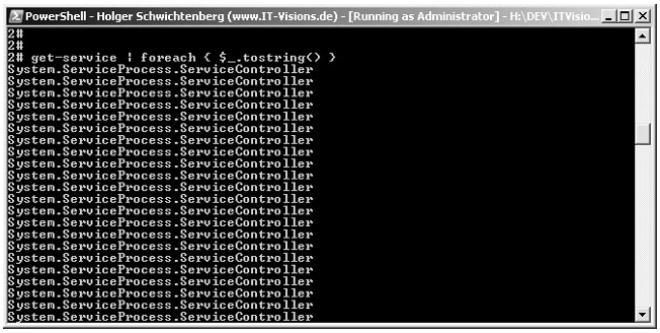


Abbildung 47.17
Anwendung von ToString() auf Instanzen der Klasse System.ServiceProcess.ServiceController



Die PowerShell Extensions von *www.IT-Visions.de* enthalten das Commandlet `Get-PipelineInfo`, mit dem man sowohl den Klassennamen als auch die Zeichenkettenrepräsentation aller Objekte in der Pipeline auf einen Blick sehen kann.

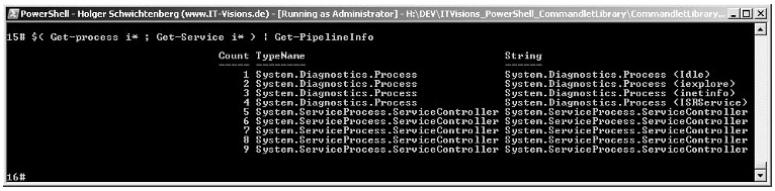


Abbildung 47.18
Anwendung von GetPipelineInfo

Kopplung auf Basis von typisierten Objekten

Typisierte Kopplung bedeutet, dass ein Objekt eines bestimmten Typs von einem Commandlet in die Pipeline gelegt und von einem anderen ausgewertet werden kann. (Beispiel: `Get-Process` erzeugt `Process`-Objekte, die `Stop-Process` verarbeiten kann.)

InputObject

Für die typisierte Kopplungsform müssen beide Commandlets die betreffende Klasse kennen, d.h., die Klasse liegt entweder in der .NET-Klassenbibliothek vor oder die selbst definierte Klasse liegt zusammen mit beiden Commandlets in einem Snap-In. Das sendende Commandlet muss nicht deklarieren, welche Objekttypen es in die

Pipeline legt. Genau genommen kann das sendende Commandlet das auch gar nicht deklarieren, denn einen solchen Mechanismus gibt es nicht in der PowerShell (was übrigens die Implementierung von komfortablen Entwicklungsumgebungen für die PowerShell erschwert bzw. das vorherige schrittweise Durchlaufen erfordert – vgl. Einzelschritt-Debugging im PowerShell Plus Editor). Das empfangene Commandlet muss einen Parameter dieses Typs deklarieren, der von der Pipeline empfangen werden darf. Diesen Parameter soll man laut den Commandlet-Konventionen [MSDN07] von Microsoft `InputObject` nennen. Ein anderer Name ist aber technisch möglich.

Beispiel

Get-DirectoryEntry und Remove-DirectoryEntry

Als Beispiel finden Sie im folgenden Listing die Implementierung von `Get-DirectoryEntry` und `Remove-DirectoryEntry`. `Get-DirectoryEntry` besitzt ein Attribut `Path`, dem per Parameter oder Pipeline eine Zeichenkette mit der Pfadangabe für einen Verzeichniseintrag übergeben werden kann. `Get-DirectoryEntry` liefert als Rückgabe das zugehörige Verzeichnisobjekt.

Die Commandlet-Klasse `RemoveDirectoryEntryCommand` besitzt zwei Attribute:

- Wenn ein Verzeichnisobjekt über die Pipeline geliefert wird, landet es in dem Attribut `InputObject` und der Verzeichniseintrag wird durch Aufruf von `Remove()` bei Einzelobjekten bzw. `DeleteTree()` bei Containern gelöscht.
- Wenn kein Verzeichnisattribut geliefert wird, wird nachgesehen, ob es einen Inhalt im Attribut `Path` (entweder über Parameter oder aus einer Zeichenkette in der Pipeline) gibt. Wenn es einen Pfad gibt, wird das zugehörige Verzeichnisobjekt instanziiert, um dann den Eintrag zu löschen.



`InputObject` hat durch die Reihenfolge der Bedingung im Programmcode Priorität, d.h., wenn es von der Pipeline schon ein Verzeichnisobjekt gibt, wird ein möglicher zusätzlicher Parameter ignoriert.

Bei der Ausführung von

```
Get-DirectoryEntry "WinNT://XFilesServer/FoxMulder" | Remove-
DirectoryEntry "WinNT://XFilesServer/DanaScully"
```

würde also nur das Konto „FoxMulder“ gelöscht. „DanaScully“ bliebe unangetastet.

Listing 47.7
Implementierung von Get-Directory-Object

```
[Cmdlet(VerbsCommon.Get, "DirectoryEntry")]
public class GetDirectoryEntryCommand : PSCmdlet
{
 [Parameter(Position = 0, ValueFromPipeline = true)]
 public string Path = "WinNT://localhost";
 protected override void ProcessRecord()
 {
  System.DirectoryServices.DirectoryEntry d =
     new System.DirectoryServices.DirectoryEntry(Path);

  this.WriteObject(d, false);
 }
}
```

Listing 47.8
Implementierung von Remove-DirectoryObject

```
[Cmdlet(VerbsCommon.Remove, "DirectoryEntry")]
public class RemoveDirectoryEntryCommand : PSCmdlet
{
 [Parameter(ValueFromPipeline = true)]
 public System.DirectoryServices.DirectoryEntry InputObject =
null;

 [Parameter(Position = 0, ValueFromPipeline = true)]
 public string Path = "";
 protected override void ProcessRecord()
 {
  System.DirectoryServices.DirectoryEntry d = null;
  if (InputObject != null)
  {
   d = InputObject;
  }
  else
  {
   if (Path != "")    d = new
   System.DirectoryServices.DirectoryEntry(Path);
  }
  if (d != null)
  {
   try
   {
    (d.Parent.Children as
       System.DirectoryServices.DirectoryEntries).Remove(d);
   }
   catch (Exception)
   {
    try
    {
         d.DeleteTree();
   this.WriteObject(true);
    }
    catch (Exception ex)
```

```
    {
     throw ex;
    }
   }

  }
  else
  {
   this.WriteObject(false);
  }
 }
```

Generische Kopplung

Bei der generischen Kopplung kann jedes beliebige Objekt weitergegeben werden. In der Regel wird durch Parametrisierung des empfangenen Commandlets definiert, was es in den Objekten in der Pipeline suchen soll. Beispiele für die zweite Kategorie sind `Where-Object` und `Select-Object`.

System.Object

Wenn Sie eine generische Kopplung implementieren wollen, müssen Sie einen Parameter anlegen, der als Typ `System.Object` hat und der seinen Inhalt von der Pipeline beziehen kann:

```
Parameter(ValueFromPipeline = true)]
public System.Object InputObject;
```

Die Arbeit mit diesem Objekt erfordert einen Mechanismus namens Reflection. Dies sind höhere Weihen der .NET-Programmierung, für die an dieser Stelle auf [SCH01] und [SCH02] verwiesen sei.

47.9 Fehlersuche in Commandlets

Visual Studio Debugger

Mit dem in Visual Studio 2005/2008 integrierten Debugger können Sie PowerShell-Commandlets „debuggen", d.h. schrittweise ablaufen lassen und währenddessen die Zustände aller Variablen verfolgen und verändern.

Dazu sind folgende Schritte notwendig:

- Festlegung der *powershell.exe* als Startanwendung *C:\WINDOWS\system32\windowspowershell\v1.0\powershell.exe* (in den Projekteigenschaften – siehe *Abbildung 47.19*)
- Laden des Snap-In durch *Profile.ps1* oder durch Festlegung des Kommandozeilenarguments *–PSConsoleFile* (in den Projekteigenschaften – siehe *Abbildung 47.20*) oder durch `-noexit` und `-command` Angabe einer Skriptdatei, die `Add-PSSnapIn` aufruft.

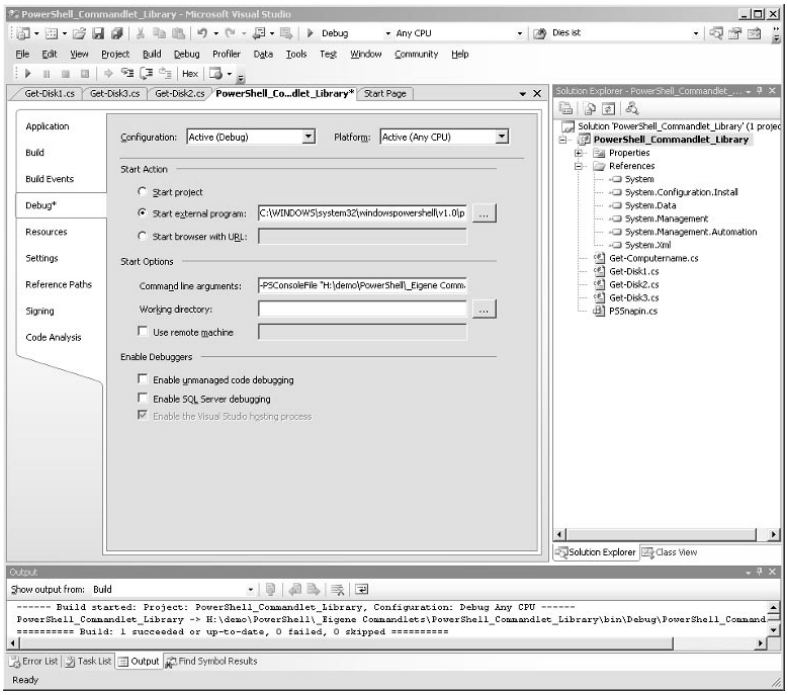


***Abbildung 47.19**: Konfiguration für das Commandlet-Debugging*

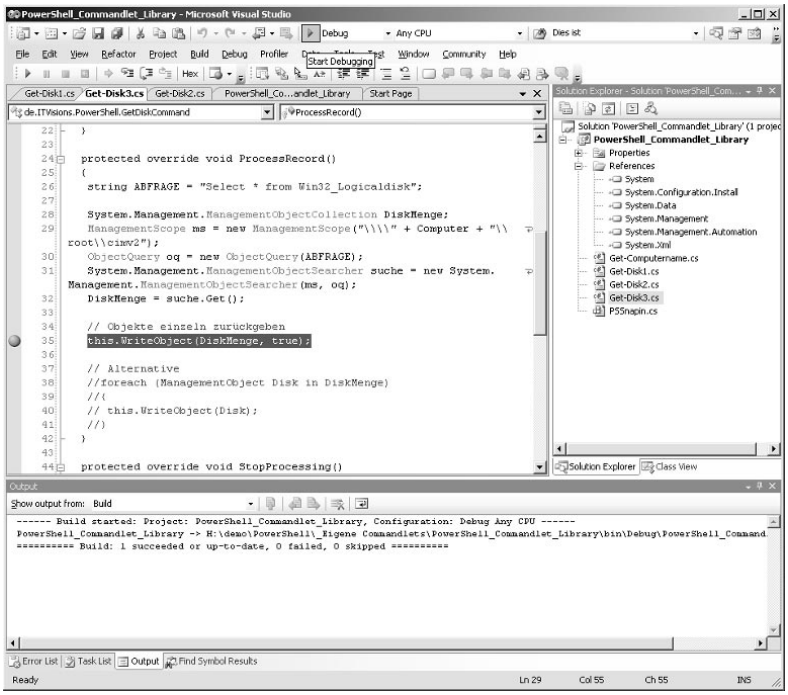


***Abbildung 47.20**: Start des Debuggings mit einem Haltepunkt*

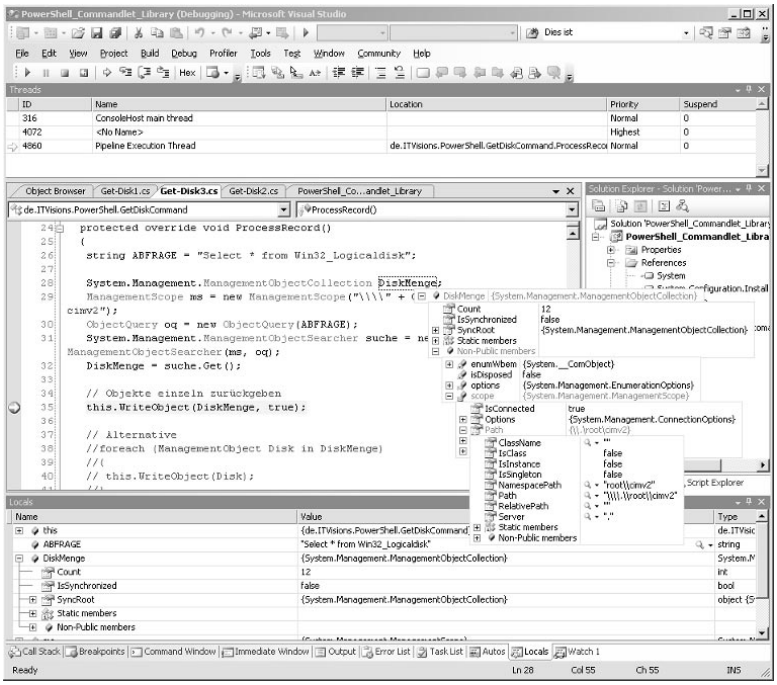


***Abbildung 47.21**: Ansicht von Variablen während des Debuggings*

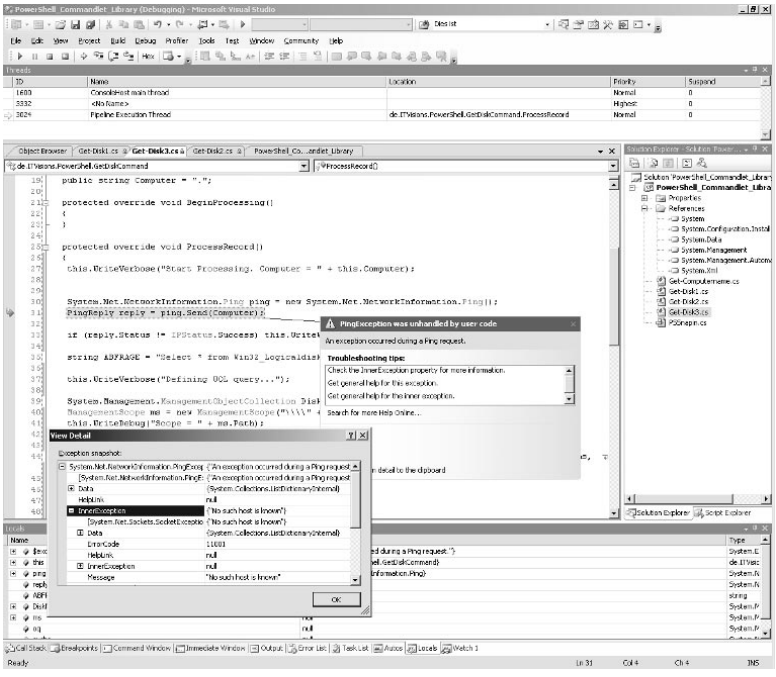


***Abbildung 47.22**: Im Fehlerfall zeigt der Debugger die Details zum Fehler direkt an der betreffenden Programmcodezeile an.*

47.10 Statusinformationen

Verbose, Debug, Error, Progress

Ein Commandlet sollte niemand direkt ins Konsolenfenster mit `Console.WriteLine()` schreiben. Microsoft hat vorgesehen, dass es abseits der Übergabe von Objekten durch die Pipeline noch andere Wege gibt, wie sich ein Commandlet der Außenwelt mitteilen kann. Ob und wie diese Mitteilungen angezeigt werden, hängt von der PowerShell-Konsole ab. Der Endbenutzer kann das Verhalten durch die Parameter –`verbose` und –`debug` sowie die Umgebungsvariablen `$VerbosePreference` und `$DebugPreference` beeinflussen.

- `WriteVerbose(String)`: Texte werden ausgegeben, wenn das Commandlet mit dem Parameter –`Verbose` oder `-Debug` gestartet wird.
- `WriteDebug(String)`: Texte werden ausgegeben, wenn das Commandlet mit dem Parameter `-Debug` gestartet wird.
- `WriteWarning(String)`: Texte werden immer ausgegeben.
- `WriteError(ErrorRecord)`: Fehlerinformationen werden ausgegeben.
- `WriteProgress(ProgressRecord)`: wird in Form einer Fortschrittsanzeige ausgegeben.



Der Unterschied zwischen `WriteError()` und dem Auslösen eines Fehlers mit dem C#-Sprachkonstrukt `throw` ist, dass nach `WriteError()` zwar ein Fehler ausgegeben wird, aber das Commandlet arbeitet weiter (Non-Terminating Error). Bei einem mit `throw` (oder `this.ThrowTerminatingError()`) ausgelösten Fehler (und allen vom .NET Framework ausgelösten Fehlern, die nicht abgefangen wurden) bricht das Commandlet sofort ab (Terminating Error).

Beispiel

Get-Disk

In der folgenden Variante der Implementierung von `Get-Disk` sind Aufrufe von `WriteVerbose()`, `WriteDebug()`, `WriteWarning()`, `WriteError()` eingebaut. In der Implementierung wird eine Warnung ausgegeben, wenn ein Ping an einem Computer nicht möglich ist. Dennoch wird anschließend ein WMI-Aufruf probiert (da Ping gesperrt sein könnte, DCOM aber möglich ist).

Listing 47.9
Implementierung von Get-Disk mit Statusinformationen

```
using System;
using System.Text;
using System.Management.Automation;
using System.Management.Automation.Provider;
using System.Management;
using System.Net.NetworkInformation;

namespace de.ITVisions.PowerShell
{
 [Cmdlet(VerbsCommon.Get, "Disk")]
 public class GetDiskCommand : PSCmdlet
```

```
{
 //   [Parameter()]
 //[Parameter(Position = 0, ValueFromPipeline = true)]

 [Parameter(Position = 0, ValueFromPipeline = true)]
 public string Computer = ".";

 protected override void BeginProcessing()
 {
 }

 protected override void ProcessRecord()
 {
  this.WriteVerbose("Start Processing. Computer = " +
this.Computer);

  // Check if computer is reachable
  if (Computer != ".")
  {
   System.Net.NetworkInformation.Ping ping = new
      System.Net.NetworkInformation.Ping();
   PingReply reply = ping.Send(Computer);
   if (reply.Status != IPStatus.Success) this.
      WriteWarning("Cannot ping computer!");
  }

  // define WQL query

  string ABFRAGE = "Select * from Win32_LogicalDisk";

  this.WriteVerbose("Defining WQL query...");
  System.Management.ManagementObjectCollection DiskMenge = null;
  ManagementScope ms = new ManagementScope("\\\\" + Computer +
                                           "\\root\\cimv2");
  this.WriteDebug("Scope = " + ms.Path);
  ObjectQuery oq = new ObjectQuery(ABFRAGE);
  this.WriteDebug("Query = " + ms.Path);
  System.Management.ManagementObjectSearcher suche =
     new System.Management.ManagementObjectSearcher(ms, oq);
  // Run WQL query

  this.WriteVerbose("Starting WQL query...");
  try
  {
   DiskMenge = suche.Get();
  }
  catch (Exception ex)
  {
   ErrorRecord er = new ErrorRecord(ex, "WQL Query Error",
```

```
        ErrorCategory.ResourceUnavailable, Computer);
     this.WriteError(er);
    }


    if (DiskMenge != null)
    {
     this.WriteDebug("Count = " + DiskMenge.Count);
     // Put objects to pipeline
     this.WriteObject(DiskMenge, true);
    }

    // Alternative
    //foreach (ManagementObject Disk in DiskMenge)
    //{
    // this.WriteObject(Disk);
    //}

    this.WriteVerbose("Finished Processing. Computer = " +
this.Computer);
   }

   protected override void StopProcessing()
   {
   }

 }
}
```

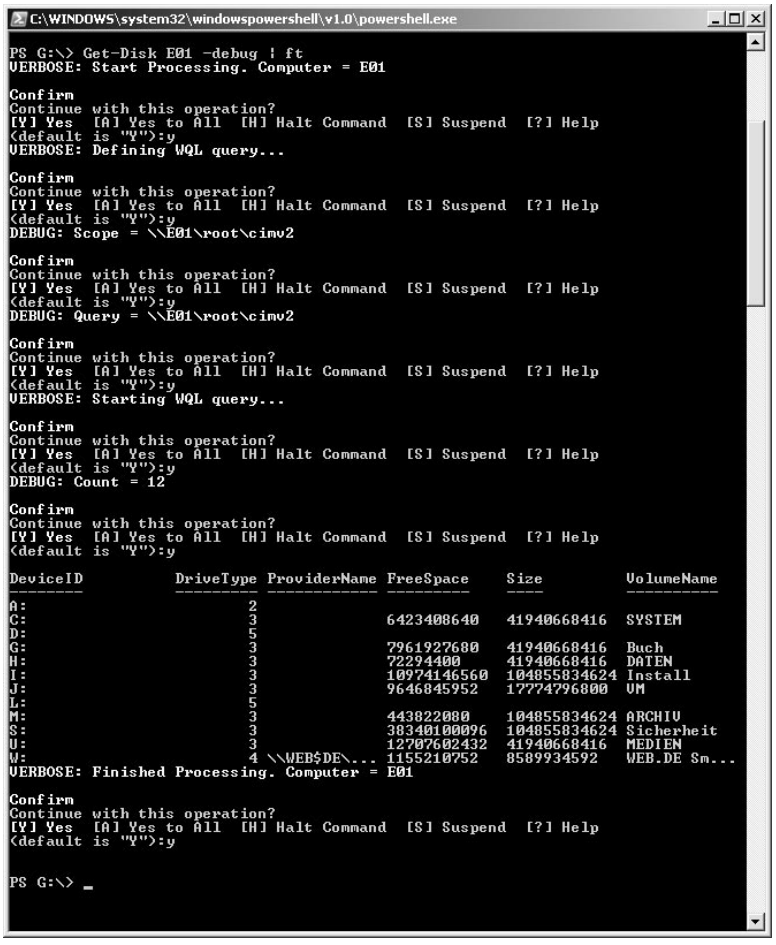
Die folgenden Bildschirmabbildungen zeigen die Auswirkungen der Statusinformationen.

- Durch Angabe des Parameters `-verbose` werden alle Texte, die mit `WriteVerbose()` übergeben wurden, in gelber Farbe ausgegeben.
- Durch Angabe des Parameters `-debug` werden alle Texte, die mit `WriteVerbose()` und `WriteDebug()` übergeben wurden, in gelber Farbe ausgegeben. Außerdem erfolgt nach jeder Ausgabe die Nachfrage, ob die Ausführung fortgesetzt werden soll.
- An `WriteWarning()` übergebene Texte werden immer ausgegeben.
- Das an `WriteError()` übergebene Fehlerobjekt wird in roter Farbe ausgegeben. Danach wird die Abarbeitung fortgesetzt (Non-Terminating Error).

Abbildung 47.23
Start von Get-Disk mit der Option –verbose



Abbildung 47.24
Start von Get-Disk mit der Option –debug



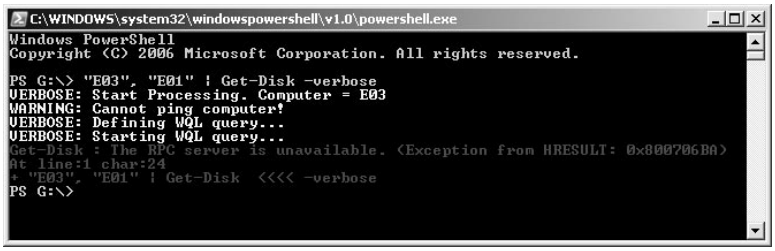


Abbildung 47.25 *Wenn die Möglichkeit, dass ein Computer nicht erreichbar ist, nicht abgefangen wird, bricht das Commandlet ab. Alle folgenden Objekte in der Pipeline werden nicht mehr beachtet.*

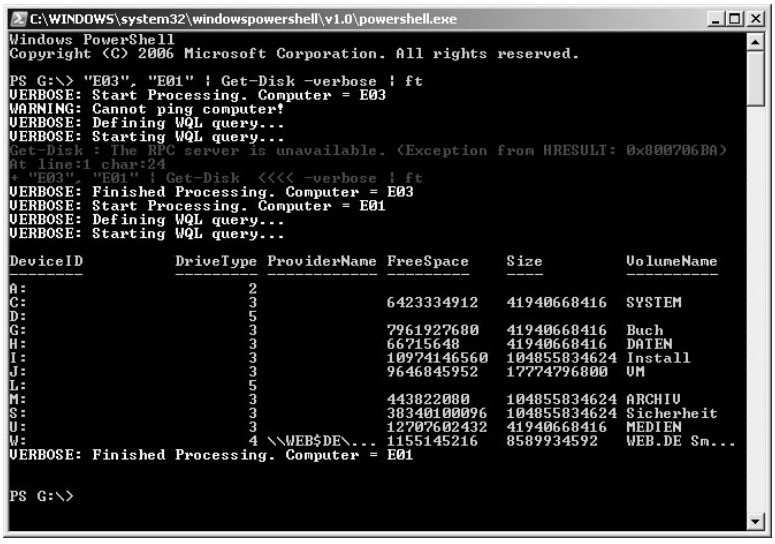


Abbildung 47.26 *Wenn ein Fehler abgefangen und mit WriteError() ausgegeben wird, arbeitet das Commandlet weiter.*

47.11 Unterstützung für Sicherheitsabfragen

Confirm und WhatIf

Viele Commandlets, die Änderungen am System ausführen, unterstützen die Sicherheitsabfrage mit `-Confirm` und `-WhatIf`. Diese Sicherheitsabfrage kann man auch eigenen Commandlets hinzufügen. Dazu sind zwei Dinge zu tun:

- In der Annotation vor der Commandlet-Klasse ist der Parameter `SupportsShouldProcess = true` anzugeben.

```
[Cmdlet(VerbsCommon.Remove, "DirectoryEntry",
SupportsShouldProcess = true)]
 public class Remove_DirectoryEntry : PSCmdlet
```

- Im Programmcode ist vor der eigentlichen Aktion, die das System verändert, `ShouldProcess()` aufzurufen. Dabei kann man wahlweise einen, zwei oder drei Parameter angeben, welche die Aktion beschreiben. `ShouldProcess()` erzeugt in Abhängigkeit von dem

angegebenen Parameter die Ausgabe. Wenn `ShouldProcess()` den Wert `false` liefert, sollte das Commandlet dann die Aktion nicht ausführen. Im Fall von `-whatif` liefert `ShouldProcess` immer `false`. Bei `-confirm` kommt `false`, wenn der Benutzer die Aktion nicht explizit bestätigt hat.

Das folgende Beispiel zeigt `ShouldProcess()` mit einem Parameter und der zugehörigen Ausgabe von `-WhatIf` und `-Confirm`.

Listing 47.10 *Bedingte Aktionsausführung mit einem Parameter bei ShouldProcess(), der den Pfad des zu löschenden Objekts enthält*

```
if (ShouldProcess(d.Path))
{
// Aktion
}
```

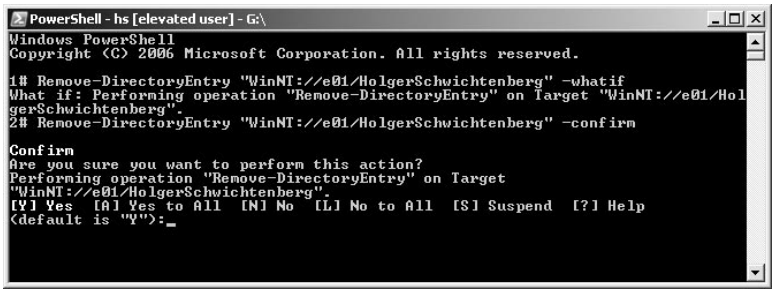


Abbildung 47.27 *Auswirkungen der obigen Verwendung von ShouldProcess() auf –whatif und -confirm*

Listing 47.11 *Bedingte Aktionsausführung mit drei Parametern bei ShouldProcess()*

```
if (ShouldProcess("Remove: " + d.Path, "Do you want to delete this
   entry permanently?", "Please confirm:"))
{

// Aktion
}
```

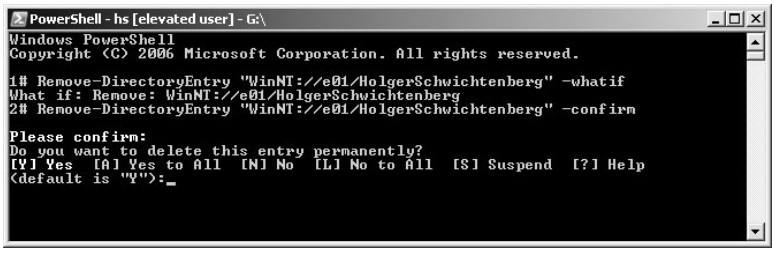


Abbildung 47.28 *Auswirkungen der obigen Verwendung von ShouldProcess() auf –whatif und -confirm*

Beispiel

Das folgende Listing zeigt ein komplettes Beispiel zur Anwendung von `ShouldProcess()`.

Listing 47.12 *Einsatz von ShouldProcess() beim Commandlet Remove-DirectoryEntry*

```
[Cmdlet(VerbsCommon.Remove, "DirectoryEntry", SupportsShouldProcess
= true)]
 public class Remove_DirectoryEntry : PSCmdlet
 {
  [Parameter(ValueFromPipeline = true)]
```

```
  public System.DirectoryServices.DirectoryEntry DirectoryObject =
null;

  [Parameter(Position = 0, ValueFromPipeline = true)]
  public string Path = "";
  protected override void ProcessRecord()
  {

   System.DirectoryServices.DirectoryEntry d = null;

   if (DirectoryObject != null)
   { // Input was a Directory Object
    d = DirectoryObject;
   }
   else

   { // Input was a Directory Path

    if (Path != "") d = new
System.DirectoryServices.DirectoryEntry(Path);
   }

   if (d != null)
   {


    if (ShouldProcess("Remove: " + d.Path,
          "Do you want to delete this entry permanently?", "Please
confirm:"))
    {
     try
     {
      (d.Parent.Children as
          System.DirectoryServices.DirectoryEntries).Remove(d);
     }

     catch (Exception)
     {
      try
      {
       d.DeleteTree();
       this.WriteObject(true);
      }
      catch (Exception ex)
      {
       throw ex;
      }
     }
    }
```

```
  }

  else

  {
   this.WriteObject(false);
  }
 }
```

47.12 Festlegung der Hilfeinformationen

MAML Hilfeinformationen für die Commandlets, die durch `Get-Help` ausgelesen werden können, sind in XML-Dateien gespeichert. Das verwendete XML-Format heißt Microsoft Assistance Markup Language (MAML).

Zur Erstellung solcher MAML-Dateien existiert eine Anwendung (Cmdlet Help Editor, siehe [FAY01]), die ein Microsoft-Mitarbeiter entwickelt hat, die aber keine offizielle Microsoft-Anwendung ist und nicht vom Microsoft Support betreut wird.



Eine Hilfedatei zu einer Snap-In-Assembly muss den Namen der Snap-In-Assembly mit angehängtem *–help.xml* haben, also:

PowerShell_Commandlet_Library.dll-Help.xml

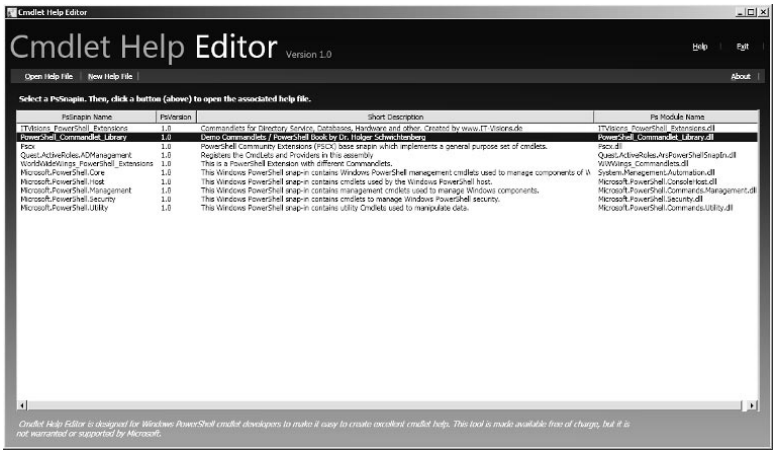


Abbildung 47.29 *Auswahl der Commandlet-Bibliothek, für die eine Hilfedatei erstellt werden soll*

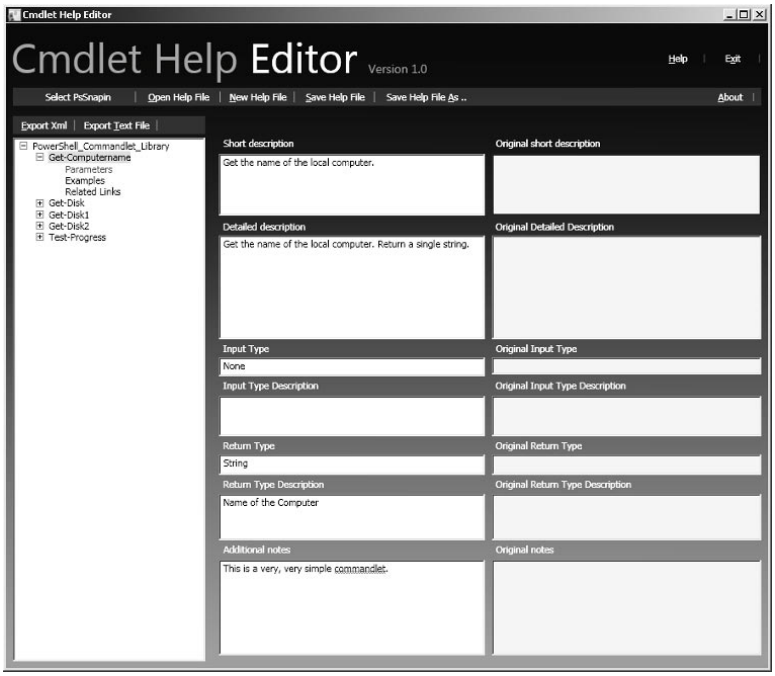


Abbildung 47.30
Festlegung der Hilfeinformationen für das Commandlet Get-Computername

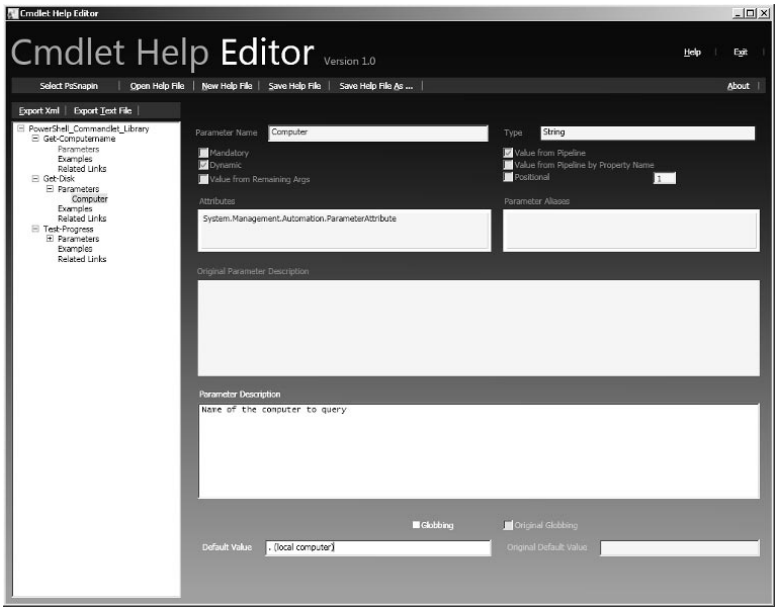


Abbildung 47.31
Festlegung der Hilfeinformationen für einen Parameter für das Commandlet Get-Disk

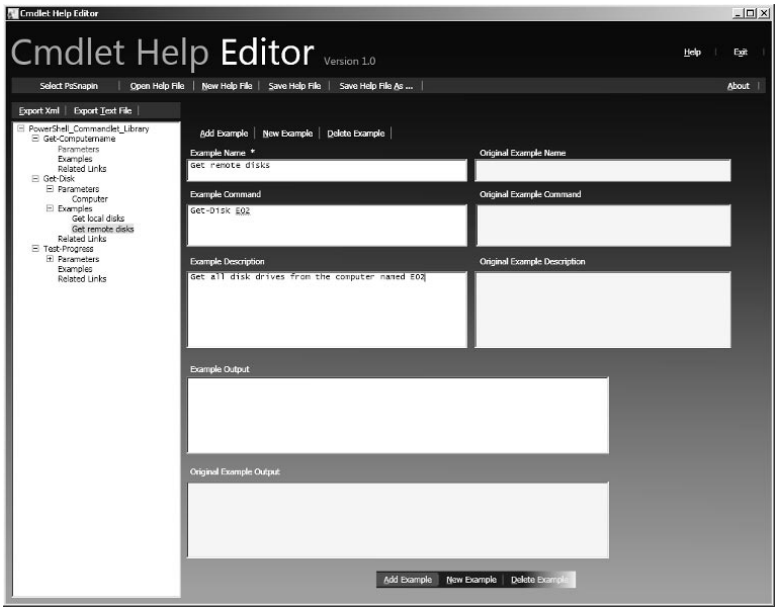


Abbildung 47.32 *Definition eines Anwendungsbeispiels für das Commandlet Get-Disk*



Die Hilfedatei muss in dem gleichen Verzeichnis gespeichert werden wie die DLL und den Namen der DLL mit dem Zusatz „-Help.xml“ tragen, z.B. *PowerShell_Commandlet_ Library.dll-Help.xml*.

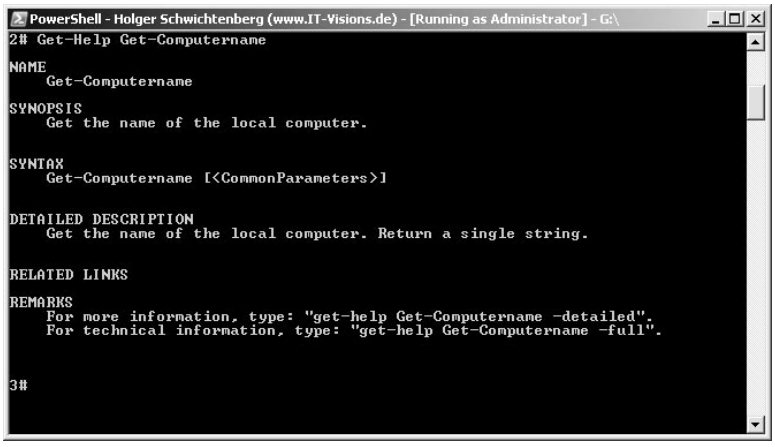


Abbildung 47.33 *Ausgabe der Hilfeinformationen für das Commandlet Get-Computername*

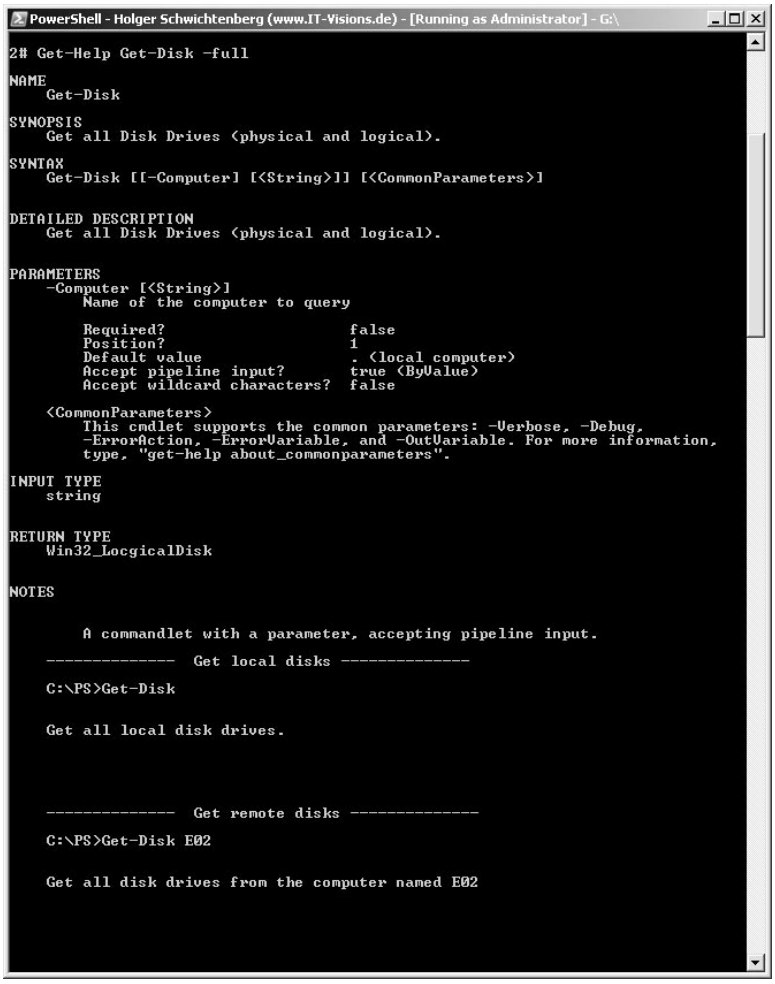


Abbildung 47.34
Ausgabe der Hilfeinformationen für das Commandlet Get-Disk

Weitere Information über das MAML-Format der PowerShell-Hilfe finden Sie unter [MSDN02].



47.13 Erstellung von Commandlets für den Zugriff auf eine Geschäftsanwendung

Alle bisherigen Beispiele zur Erstellung eigener Commandlets in diesem Buch beziehen sich auf die Windows-Infrastruktur, also das Auslesen und Verändern von Systeminformationen. In Zukunft werden auch Geschäftsprozessanwendungen kommandozeilenbasierte Schnittstellen auf Basis der Windows PowerShell bereitstellen, um bestehende Geschäftslogiken für iterative Systemadministration und Scripting bereitzustellen.

Für eine Besprechung dieses Themas stehen leider in diesem Buchprojekt nicht ausreichend Seiten zur Verfügung. Sie finden auf der Begleit-CD-ROM zu diesem Buch jedoch ein mehrschichtiges .NET-Fallbeispiel, das auch eine Unterstützung für die PowerShell bietet.

World Wide Wings

„World Wide Wings" ist eine fiktive Fluggesellschaft, die als Fallbeispiel für eine mehrschichtige, komponentenbasierte .NET-Anwendung dient. Dieses Fallbeispiel wird vom Autor dieses Buchs in einigen .NET-Büchern sowie seinen .NET-Schulungen verwendet.

Die Fluggesellschaft World Wide Wings (kurz WWWings oder WWW) ist eine Charter-Fluggesellschaft. Sie bietet nationale, europäische und interkontinentale Flüge an. Die verschiedenen Desktop-, Konsolen- und webbasierten WWWings-Anwendungen verwalten Flüge, Passagiere, Flugbuchungen und Mitarbeiter der Fluggesellschaft.

In dem Beispiel sind u.a. folgende Commandlets für den Zugriff auf die Geschäftslogik realisiert:

- `Get-Flugziel`: liefert eine Liste der Abflughäfen, von denen die Gesellschaft losfliegt, oder bei Angabe eines Flughafens die Ziele, welche die Gesellschaft von dort ansteuert. Die Rückgabe ist immer eine Menge von Zeichenketten.
- `Get-Flug`: liefert Informationen über einen Flug anhand der Flugnummer, alle Flüge auf einer Route anhand zweier Flughäfen oder die Flugziele.
- `Get-Passagier`: liefert Informationen über einen Passagier inklusive einer Liste seiner Buchungen.
- `New-Buchung`: erstellt eine neue Buchung für einen Passagier.
- `Remove-Buchung`: löscht eine Buchung für einen Passagier.

Alle Commandlets haben direkt Auswirkungen auf die Geschäftsprozessdatenbank von World Wide Wings, die sich in einem Microsoft SQL Server 2005 befindet.

Die folgende Bildschirmabbildung zeigt den Einsatz dieses Commandlets, insbesondere auch die Kopplung der Commandlets durch typisierte Objekte. So nimmt z.B. das Commandlet `New-Buchung` wahlweise für den Flug eine Flugnummer oder ein `Flug`-Objekt und für den Passagier eine Passagiernummer oder ein `Passagier`-Objekt entgegen.

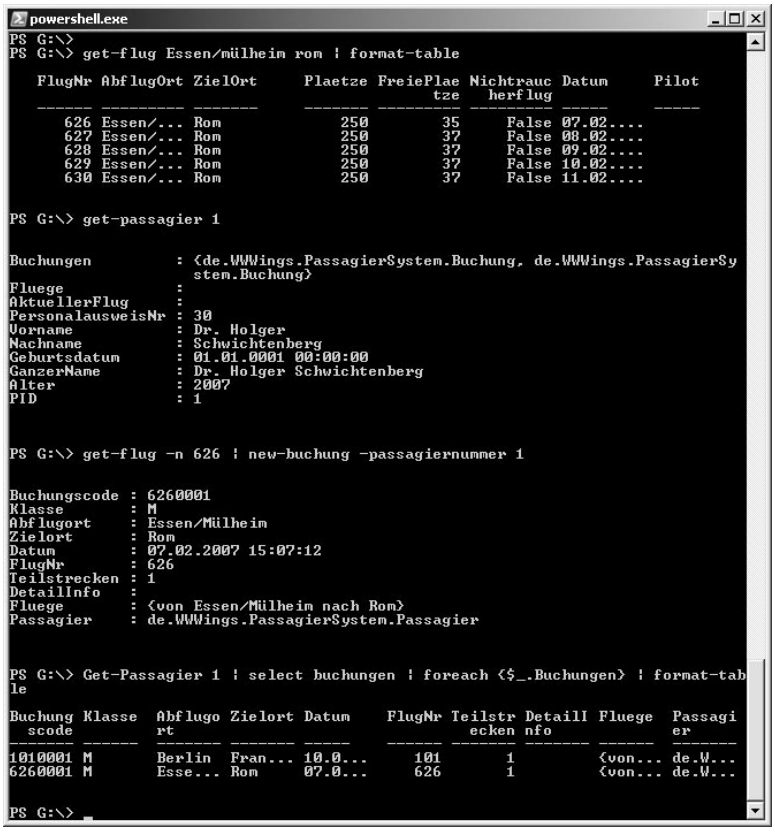


Abbildung 47.35 *Einsatz der World Wide Wings-Commandlets*

47.14 Konventionen für Commandlets

Namenskonventionen

Bei den Verbnamen sollte man möglichst auf die bestehenden Verbnamen zurückgreifen, die Microsoft in sechs Gruppen vordefiniert hat (siehe *Tabelle 47.1*).

Tabelle 47.1 *Vordefinierte Verben für Commandlets*

VerbsCommon	Add	Clear
	Copy	Get
	Join	Lock
	Move	New
	Remove	Rename
	Select	Set
	Split	Unlock
VerbsCommunications	Connect	Disconnect
	Read	Receive
	Send	Write
VerbsData	Backup	Checkpoint
	Compare	Convert
	ConvertFrom	ConvertTo
	Dismount	Export
	Import	Initialize
	Limit	Merge
	Mount	Out
	Restore	Update
VerbsDiagnostic	Debug	Measure
	Ping	Resolve
	Test	Trace
VerbsLifeCycle	Disable	Enable
	Install	Restart
	Resume	Start
	Stop	Suspend
	Uninstall	
VerbsSecurity	Block	Grant
	Revoke	Unblock

Außerdem gibt es folgende Grundsätze für die Erstellung von Commandlets:

- Ein Commandlet sollte Objekte zurückliefern und keine Zeichenketten.
- Ein Commandlet sollte auch „in der Mitte“ in einer Pipeline eingesetzt werden können, d.h., es sollte Objekte entgegennehmen und wieder Objekte erzeugen.
- Ein Commandlet sollte niemals direkt Ausgaben mit .NET Framework-Funktionen wie `Console.WriteLine()` etc. erzeugen.
- Ein Commandlet sollte Objekte zurückliefern und keine Zeichenketten.
- Ein Commandlet sollte die Ausgabeobjekte immer einzeln in die Pipeline legen.
- Nach der Übergabe von Objekten an die Pipeline sollte das Commandlet die Objekte nicht mehr verwenden, da diese bereits von dem nachfolgenden Commandlet bearbeitet werden könnten.
- Ein Commandlet sollte sich nicht auf eine bestimmte Ausführungsreihenfolge von Commandlets in einem PowerShell-Befehl verlassen.

- Wenngleich der Name der Commandlet-Klasse nicht maßgeblich ist, schlägt Microsoft vor, die Klasse nach der Konvention „VerbSubstantivCommand“ zu nennen, also für das Commandlet Add-DirectoryObject sollte die Klasse AddDirectoryObjectCommand heißen.
- Bei der Namensgebung für Parameter sollte PascalCasing (am Anfang und für jedes neue Wort ein Großbuchstabe) verwendet werden.
- Die folgenden Zeichen dürfen in Commandlet-Namen nicht verwendet werden: # , () {} [] & - / \ $; : ” ’ <> | ? @ `
- Für Ja-/Nein-Parameter (alias „Schalter“) soll die Klasse `SwitchParameter` anstelle von Boolean zum Einsatz kommen, damit man solche Schalter einfach durch `-Schalter` anstelle von `-Schalter true` setzen kann.
- Ein Commandlet sollte immer `-verbose` und `–debug` unterstützen.
- Ein Commandlet, das Veränderungen am System vornimmt, sollte `-whatif` und `-confirm` unterstützen.

Weitere Empfehlungen finden Sie in den „Cmdlet Development Guidelines“ [MSDN07].



47.15 Weitere Möglichkeiten

Noch mehr Möglichkeiten

Folgende Möglichkeiten bei der Erstellung von Commandlets können in diesem Buch aus Platzgründen nicht weiter thematisiert werden.

- Ein Commandlet kann Objekte, die es aus der Pipeline empfängt, um neue Attribute und Methoden anreichern. Anders als im Kern von .NET geht das nicht nur mit Methoden. Die PowerShell hat dazu ihr eigenes Extended Type System (ETS).
- Commandlets mit vielen Parametern können Parametergruppen (Parameter Sets) zur Festlegung unterschiedlicher Parametermengen definieren.
- Der Entwickler kann Bedingungen für Parameter deklarieren, die die PowerShell automatisch überprüft, z.B. `[ValidateLength(1,5)]`, `[ValidatePattern("[0-9A-F]*")]` und `[ValidateNotNull()]`.
- Die Standardformatierung für die von den Commandlets gelieferten Objekte kann man in der Typbeschreibungsdatei der PowerShell (*Produktname.Format.ps1xml*) festlegen.
- Ein Commandlet kann auch auf die laufende PowerShell-Konsole zugreifen, z.B. die belegten Variablen mit `this.GetVariableValue()`.
- Ein Commandlet kann in Parametern Platzhalter (Wildcards) unterstützen.

Weitere Informationen zu diesen Punkten finden Sie im PowerShell SDK [MSDN03].



48 Hosting der Windows PowerShell

Architektur der PowerShell

Die Windows PowerShell besteht logisch aus drei Schichten: den Windows-PowerShell-Grundfunktionen wie dem Pipelining und dem Providerkonzept, den Windows-PowerShell-Commandlets sowie -Providern und dem Windows-PowerShell-Host, der den Windows-Prozess und gegebenenfalls eine grafische Benutzeroberfläche bereitstellt. Microsoft liefert mit PowerShell 2.0 zwei verschiedene Hosts aus: die PowerShell-Konsole (powershell.exe) und das Integrated Scripting Environment (powershell_ise.exe). Microsoft erlaubt es lizenzkostenfrei (!), die PowerShell-Grundfunktionen und die von Microsoft gelieferten Commandlets und Provider in eigene Anwendungen zu integrieren.

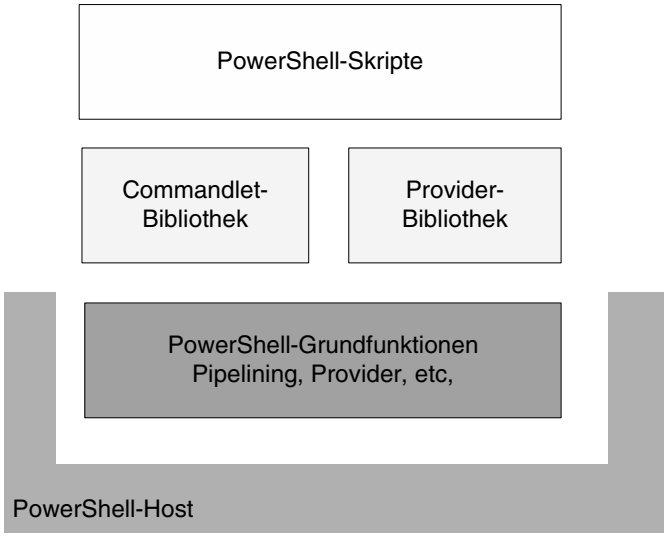


Abbildung 48.1 *Schichtenmodell der Windows PowerShell*

Neben der bereits in PowerShell 1.0 enthaltenen Hosting-Möglichkeit, die relativ komplex war, bietet die PowerShell 2.0 eine zusätzliche vereinfachte Hosting-Schnittstelle.

Jede beliebige .NET-Anwendung kann PowerShell hosten, also sowohl Windows-Forms- und WPF-Anwendungen als auch Konsolenanwendungen und Dienste ohne Benutzerschnittstelle.

48.1 Voraussetzungen für das Hosting

System.Management.Automation

Für das Hosting der PowerShell benötigt man die Assembly *System.Management.Automation.dll*. Unter Windows XP, Vista und Windows Server 2003/2008 findet man diese Assembly unter *C:\Windows\System32\WindowsPowerShell\v1.0*. In Windows 7 und Windows Server 2008 R2 muss man in den Global Assembly Cache sehen unter *C:\Windows\assembly\GAC_MSIL\System.Management.Automation\1.0.0.0__31bf3856ad364e35*.



Leider hat Microsoft die Versionsnummer der Assembly nicht hochgezählt. Auch in PowerShell 2.0 steht daher im Global Assembly Cache noch „1.0“. Man kann die DLLs aber anhand der Dateiversionsnummer unterscheiden: 6.0.6000 für PowerShell 1.0 und 6.1.7600 für PowerShell 2.0.

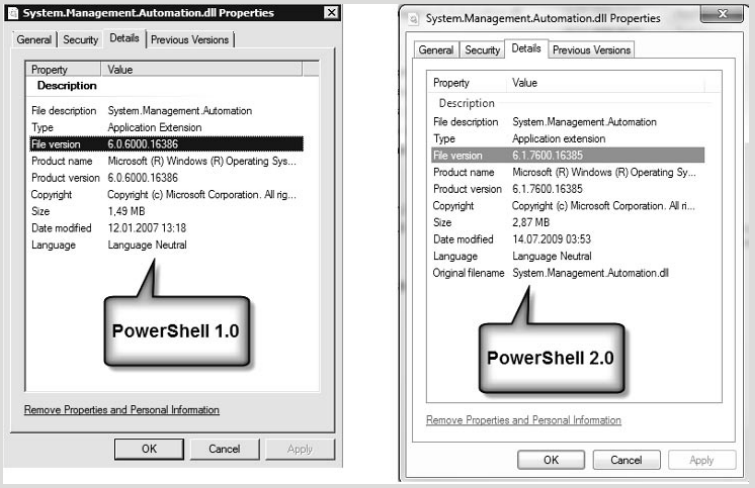


***Abbildung 48.2**: Vergleich der Versionsnummern der System.Management.Automation.dll*

48.2 Hosting mit PSHost

PSHost

Für die „ältere“ Form des Hosting sind drei Klasse zu implementieren:

- Eine Klasse, die von `PSHost` erbt. Hier sind einige wenige Attribute und Methoden zu implementieren mit Basisinformationen über den Host wie z.B. Name, Versionsnummer und aktuelle Sprache.

- Eine Klasse, die von `PSHostUserInterface` erbt. Diese Klasse realisiert die Entgegennahme von Eingaben (Prompt) sowie Ausgaben (Write).
- Eine Klasse, die von `PSHostRawUserInterface` erbt. Hierin sind Funktionen wie Farben und Fenstergröße abzulegen.

Alle drei Basisklassen findet man im Namensraum `System.Management.Automation.Host`. Eine detallierte Diskussion jeder einzelnen zu implementierenden Eigenschaft muss hier aus Platzgründen entfallen. Vielmehr soll anhand eines Beispiels eine praktische Einführung gegeben werden.

Beispiel

Beispiel

Die folgende Bildschirmabbildung zeigt eine Windows-Forms-Anwendung mit einer Eingabezeile und einem größeren Ausgabebereich. Die Eingabezeile erlaubt beliebige PowerShell-Commandlets und auch andere Arten von Befehlen wie Berechnungen.

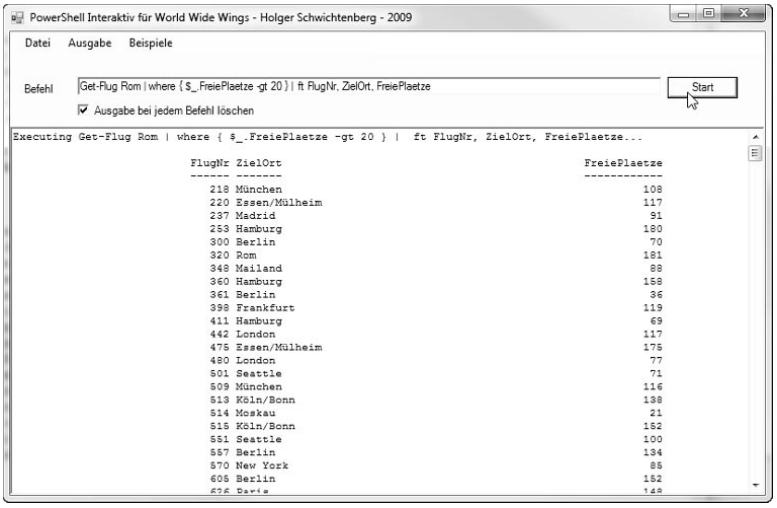


Abbildung 48.3 *Hosting der PowerShell in einer Windows Forms-Anwendung*

Die folgende Abbildung zeigt die Architektur des Beispiels. Neben jeweils einer Klasse zur Implementierung der drei oben genannten Klassen gibt es zwei weitere Klassen. Die Klasse „Program“, die den PSHost und einen sogenannten Runspace erzeugt sowie die grafische Benutzeroberfläche startet. Der Runspace verweist dabei auf Host und der Runspace ist zu öffnen.

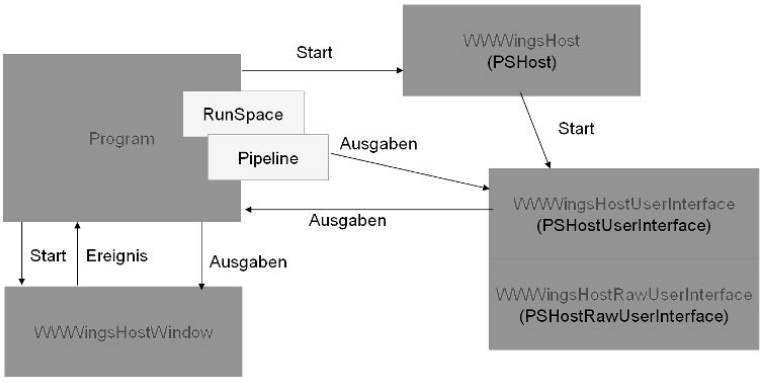
```
pshost = new WWWingsHost(this, w);

runspace = RunspaceFactory.CreateRunspace(pshost);

runspace.Open();
```

Die in `WWWingsHostWindow` implementierte Benutzeroberfläche weiß im Sinne guter Schichtentrennung gar nicht, dass die eingegebenen Befehle von der PowerShell ausgeführt werden. Nach einem Klick auf die Start-Schaltfläche (Name „C_Start“) erzeugt die Benutzeroberfläche ein Ereignis vom Typ `CommandEntered()`, wobei die in der Textbox `C_Eingabe` erfasste Zeichenkette als Parameter übergeben wird. Konsument des Ereignisses ist die Klasse `Program`.

Abbildung 48.4
Architektur des Beispiels



Aufruf der Pipeline

`Program` erzeugt dann in der Methode `Execute()` eine neue Pipeline durch die Methode `CreatePipeline()` in dem `Runspace`-Objekt. Die eingegebene Zeichenkette wird dann durch AddScript() an die Pipeline übergeben::

```
currentPipeline.Commands.AddScript(cmd);
```

Durch das Hinzufügen von `Out-Default` an der Pipeline wird die Standardausgabe für viele Commandlets aktiviert.

```
currentPipeline.Commands.Add("Out-Default");
```

Die Ausführung der Pipeline startet man dann mit

```
currentPipeline.Invoke();
```

Diese Methode liefert kein direktes Ergebnis. Vielmehr sendet die Pipeline die Ausgaben an den Runspace, der Runspace an den Host und der Host nutzt zur Ausgabe das `PSHostUserInterface`, das sich wiederum zum Teil des `PSHostRawUserInterface` bedient. Das `PSHostUserInterface` sendet an das Programm die Ausgabe und das Programm sendet diese an die Benutzerschnittstelle.

Listing 48.1
Methode Execute() in der Klasse Program

```
void Execute(string cmd, object input)
        {
            // Just ignore empty command lines...
            if (String.IsNullOrEmpty(cmd))
                return;
```

```
            // Create the pipeline object and make it available
            // to the ctrl-C handle through the currentPipeline
               instance
            // variable

            lock (instanceLock)
            {
                currentPipeline = myRunSpace.CreatePipeline();
            }

            // Create a pipeline for this execution - place the
               result in the currentPipeline
            // instance variable so it is available to be stopped.
            try
            {
                currentPipeline.Commands.AddScript(cmd);

                // Now add the default outputter to the end of the
                   pipe and indicate
                // that it should handle both output and errors
                   from the previous
                // commands. This will result in the output being
                   written using the PSHost
                // and PSHostUserInterface classes instead of
                   returning objects to the hosting
                // application.

                currentPipeline.Commands.Add("out-default");

currentPipeline.Commands[0].MergeMyResults(PipelineResultTypes.Error,
PipelineResultTypes.Output);

                // If there was any input specified, pass it in,
                   otherwise just
                // execute the pipeline...

                if (input != null)
                {
                    currentPipeline.Invoke(new object[] { input });
                }
                else
                {
                    currentPipeline.Invoke();
                }
            }
            finally
            {
               // Dispose of the pipeline line and set it to null,
                  locked because currentPipeline
```

```
                // may be accessed by the ctrl-C handler...
                lock (instanceLock)
                {
                    currentPipeline.Dispose();
                    currentPipeline = null;
                }
            }
        }
```

48.3 Vereinfachtes Hosting ab PowerShell 2.0

Klasse PowerShell

Die Windows PowerShell 2.0 bietet alternativ eine starke vereinfachte Möglichkeit zur Ausführung von PowerShell-Befehlen, bei der keine einzige der PowerShell-Basisklassen implementiert werden muss. Man erzeugt einfache eine Instanz der PowerShell (mit Hilfe der Klasse `System.Management.Automation.PowerShell`), übergibt den auszuführenden Befehl (oder die Befehlskette) und erhält eine Menge von Objekten zurück mit dem Inhalt der Pipeline nach Ende der Ausführung in Form von Instanzen der Klasse `PSObject`. Leider gibt es auch einen Haken: Um die Ausgabe muss man sich nun selbst kümmern. Dies bedeutet, dass weder die Standardausgabe mit `Out-Default` noch die Format-Commandlets funktionieren.

Das folgende Listing zeigt die Anwendung von `Get-Process` und die anschließende Ausgabe von drei Attributen der Klasse in einer Tabelle.

Listing 48.2 *Vereinfachtes Hosting der PowerShell 2.0*

```
public static void Run1()
  {
   PowerShell ps = PowerShell.Create();

   // Einzelbefehl
   ps.AddCommand("Get-Process");

   // Kopfzeile
   Console.WriteLine("Prozess                ID        Speicher");
   Console.WriteLine("---------------------------------------");

   // Ausführung
   Collection<PSObject> ErgebnisMenge = ps.Invoke();

   // Ergebnismenge darstellen
   foreach (PSObject Ergebnis in ErgebnisMenge)
   {
```

```
    Console.WriteLine(
            "{0,-24}{1,-10}{2}",
            Ergebnis.Members["ProcessName"].Value,
            Ergebnis.Members["Id"].Value,
            Ergebnis.Members["WorkingSet64"].Value);
  }
}
```

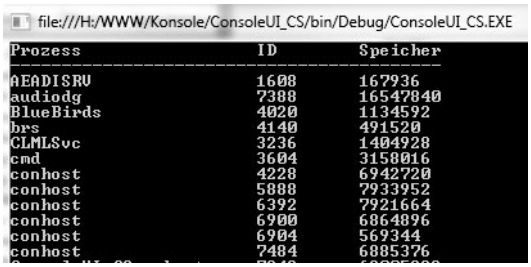


Abbildung 48.5
Ausgabe des obigen Beispiels



Mit `AddCommand()` kann man nur einzelne Befehle der PowerShell übergeben. Mehrere Befehle, die durch eine Pipeline verkettet sind, muss man mit `AddScript()` übergeben:

ps.AddScript("Get-Process | where { $_.workingset64 -gt 100000000} ");

Das folgende Listing zeigt ein Beispiel für eine generische Ausgabe einer Menge von Instanzen der Klasse `PSObject`. Dabei kommt uns zu Gute, dass `PSObject` bereits Metainformationen über den Objektinhalt enthält. Über die Menge „Properties" kann man den Namen der Attribute und deren Wert auslesen. `Print()` druckt aus Platzgründen jeweils nur die ersten fünf Attribute und verkürzt diese jeweils auf 15 Zeichen.

Listing 48.3
Funktionen für die generische Ausgaben für Mengen von PSObject-Objekten

```
const byte Spalten = 5;
  const byte ZeichenProSpalte = 15;
  /// <summary>
  /// Generische Ausgaben für Mengen von PSObject-Objekten
  /// </summary>
  /// <param name="Menge">Auszugebende Menge</param>
  public static void Print(Collection<PSObject> Menge)
  {
   bool HeaderPrinted = false;

   foreach (PSObject result in Menge)
   {

    if (!HeaderPrinted)
    {
     HeaderPrinted = true;
```

```
      byte hcount = 0;
      foreach (PSMemberInfo p in result.Properties)
      {
       hcount++;
       if (hcount > Spalten) break;
       Print(p.Name);
      }
      Console.WriteLine();
      Console.WriteLine("----------------------------------------
--------------------------------");
     }

     byte count = 0;
     foreach (PSMemberInfo p in result.Properties)
     {
      count++;
      if (count > Spalten) break;
      string value = "";
      try
      {
       value = p.Value.ToString();
      }
      catch (Exception)
      {
      }
      Print(value);
     }

     Console.WriteLine("");
    }
   }

   /// <summary>
   /// Ausgabe einer auf 15 Zeichen begrenzten Zeichenkette
   /// </summary>
   /// <param name="s">Zeichenkette</param>
   public static void Print(string s)
   {
    if (s.Length > ZeichenProSpalte) s = s.Substring(0,
ZeichenProSpalte-3) + "...";
    Console.Write(string.Format("{0," + ZeichenProSpalte +"}", s));
   }
```

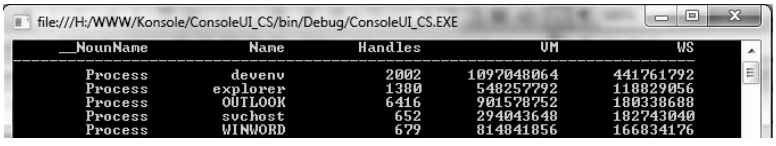


Abbildung 48.6 *Generische Ausgabe der ersten fünf Attribute*

49 PowerShell-Module erstellen

Die Grundlagen und die Anwendung von PowerShell-Modulen finden Sie bereits im *Teil 2 „PowerShell-Aufbauwissen“*. Dieses Kapitel behandelt nun das Erstellen von PowerShell-Modulen.

Modularten

Bei der Erstellung von Modulen sind drei Fälle zu unterscheiden:

- Skriptmodul
- Binärmodul
- Manifestmodul

```
PS C:\Users\hs.ITV> Get-Module -ListAvailable | ft name, ModuleType, Description, Version

Name                                ModuleType Description                    Version
----                                ---------- -----------                    -------
DesktopSearch                          Script                                 0.0
DotNet                                 Script                                 0.0
FileSystem                           Manifest Windows PowerShell module ...   1.0
IsePack                              Manifest A Collection of Tools for ...   1.1
LDAPCommandlets                        Script                                 0.0
PowerShellPack                       Manifest Windows 7 Resource Kit Pow...   1.0
PSCodeGen                            Manifest Scripts to help with manag...   1.0
PSImageTools                         Manifest A powershell module to wor...   1.0
PSRSS                                Manifest Powershell functions to ge...   1.1
PSSystemTools                        Manifest A module to get and modify...   1.0
PSUserTools                          Manifest A Windows PowerShell Modul...   1.0
TaskScheduler                        Manifest A set of scripts to work w...   1.0
WPK                                  Manifest The WPF Powershell Kit is ...   1.0
WWWings_Commandlets                    Binary                                 0.0
WWWings_ProviderAndCommand...        Manifest Modul für World Wide Wings...   1.0
ActiveDirectory                      Manifest Active Directory Module         1.0.0.0
AppLocker                            Manifest Powershell AppLocker Module     1.0.0.0
BitsTransfer                         Manifest                                 1.0.0.0
GroupPolicy                          Manifest                                 1.0.0.0
PSDiagnostics                        Manifest                                 1.0.0.0
TroubleshootingPack                  Manifest Microsoft Windows Troubles...   1.0.0.0
WebAdministration                    Manifest                                 1.0.0.0
```

Abbildung 49.1
Den Modultyp kann man bei Get-Module erkennen.

49.1 Erstellen eines Skriptmoduls

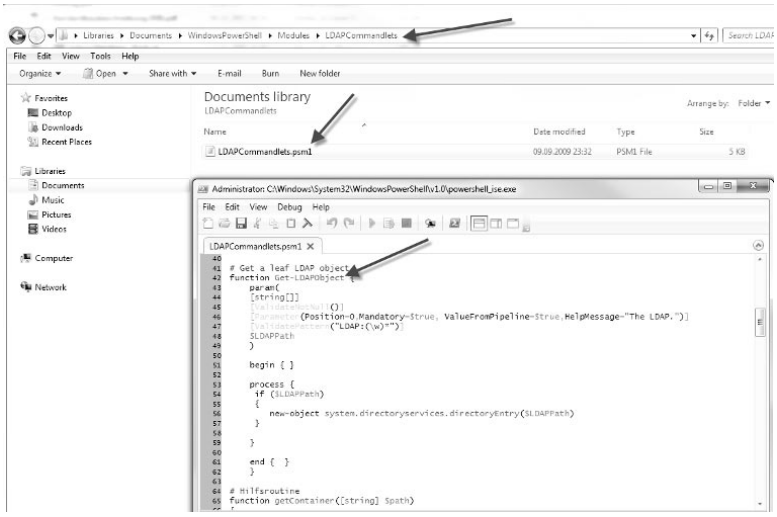
Ein Skriptmodul besteht aus einer PowerShell-Skriptdatei mit der Dateinamenserweiterung .psm1. Die Skriptdatei muss so heißen wie das Modul, also wäre z.B. für das Modul LDAPCommandlets der Dateiname LDAPCommandlets.psm1. An den Aufbau der .psm1-Datei gibt es keine Anforderung. Jede Skriptdatei, die Funktionen definiert, ist möglich. Die definierten Funktionen werden alle als funktionsbasierte Commandlets exportiert. So gesehen ist also für ein Skriptmodul nicht mehr zu tun, als eine .ps1-Datei in .psm1 umzubenennen und die Datei in ein Verzeichnis mit gleichem Namen unterhalb eines der PowerShell-Modulverzeichnisse abzulegen.



Anders als beim „DotSourcing“ einer Skriptdatei werden aber „freie“ Befehle in der .psm1-Datei nicht ausgeführt.

Bei einem Skriptmodul kann man ebenso wie beim „DotSourcing“ nicht auswählen, welche Funktionen exportiert (d.h. zum Aufruf veröffentlicht) werden. Es werden alle enthaltenen Funktionen veröffentlicht.

Abbildung 49.2
Ein PowerShell-Skript-Modul, das die Funktion Get-LDAPObject exportiert, von außen und innen





Modulverzeichnisse sind im Standard \Windows\System32\WindowsPowerShell\v1.0\Modules sowie $home\Documents\WindowsPowerShell\Modules für jeden einzelnen Benutzer. Die Umgebungsvariable `PSModulePath` liefert den Pfad zu den Benutzermodulen. Man kann diesen Pfad ändern oder auch weitere Pfade hinzufügen, z.B. $env:psmodulepath = $env:psmodulepath + „;h:\WPSModules“ (siehe auch *Teil 2 „PowerShell-Aufbauwissen“*).

49.2 Erstellen eines Moduls mit Binärdateien

Analog zum Skriptmodul ist ein Binärmodul ein Modul mit einer DLL, deren Name dem Modulnamen entspricht. Das Binärmodul kann weitere abhängige DLLs im gleichen Verzeichnis haben. Es werden aber nur die in der Haupt-DLL enthaltenen Commandlets und Provider exportiert.

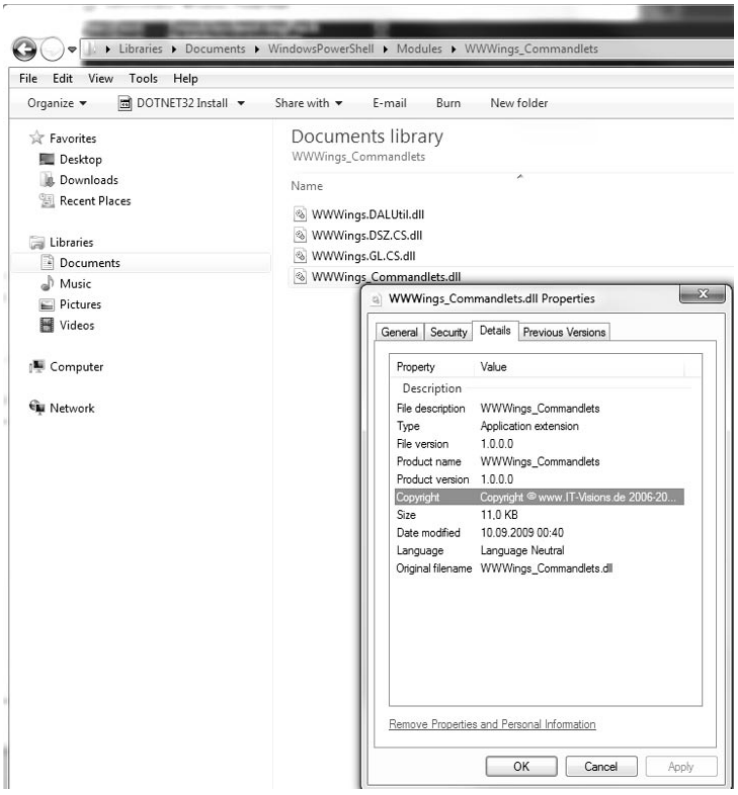


Abbildung 49.3
Ein Binärmodul

49.3 Erstellen eines Moduls mit Manifest

Ein Modul mit Manifest ist die flexibelste Form eines PowerShell-Moduls, weil hier zahlreiche Skript- und/oder Binärdateien in einem Modul zusammengefasst werden können. Ein Modul-Manifest ist eine Textdatei mit Meta-Informationen. Die wesentliche Information im Modul-Manifest sind die Dateien, die zu dem Modul gehören.

Das Manifest ist eine Textdatei mit der Dateinamenserweiterung (.psd1). Inhalt der Datei sind Name-Wert-Paare (in Form einer Hash-Tabelle) sowie optional Kommentare, die mit „#“ beginnen. Beispiele für Name-Wert-Paare sind:

- ModuleVersion = '1.0'
- Author = 'Dr. Holger Schwichtenberg'
- NestedModules = „Access_Provider.dll“, „WWWings_Commandlets.dll“, „Start.ps1“, „StartProvider.ps1“, „StartCommandlets.ps1“

Die folgende Tabelle zeigt alle möglichen Namen. Die einzige Pflichtangabe ist ModuleVersion. Alle anderen Angaben sind optional.

Schlüssel	Datentyp	Beschreibung
ModuleVersion	Zeichenkette im Format von System.Version, z.B. 1.2.3.4	Version des Moduls
GUID	Zeichenkette in Form einer GUID, z.B. 0093ae2d-89e4-494c-81a6-35881fafb6f2	Global Unique Identifier für das Modul
Author	Zeichenkette	Name des Modulautors
CompanyName	Zeichenkette	Firma des Modulautors
Copyright	Zeichenkette	Hinweis auf Rechteinhaber
Description	Zeichenkette	Beliebiger Beschreibungstext
PowerShell-Version	Zeichenkette im Format von System.Version, z.B. 2.0	Notwendige Version der PowerShell. Diese muss mindestens 2.0 sein, denn vorher gab es noch keine Module.
CLRVersion	Zeichenkette im Format von System.Version, z.B. 4.0	Notwendige Version der Common Language Runtime, z.B. 2.0 oder 4.0. Zu beachten ist, dass die CLR-Version für .NET 2.0, 3.0 und 3.5 jeweils 2.0 ist, da sich die CLR in .NET 3.0 und 3.5 nicht verändert hat.
DotNetFrameworkVersion	Zeichenkette im Format von System.Version, z.B. 3.5	Notwendige Version des .NET Frameworks, z.B. 3.0, 3.5 oder 4.0
RequiredModules	Eine Liste von Zeichenketten in der Form @("ModulName", "ModulName")	Liste von abhängigen Modulen, die bereits importiert sein müssen. Zu beachten ist, dass diese Anweisung nicht dazu führt, dass diese Module auch tatsächlich importiert werden.

***Tabelle 49.1**: Elemente eines Modul-Manifests*

Schlüssel	Datentyp	Beschreibung
Required-Assemblies	Eine Liste von Zeichenketten in der Form @("AssemblyName", "AssemblyName")	Liste von .NET-Assemblies, die geladen werden sollen. Die Assemblies können sich im Global Assembly Cache (GAC), dem zentralen Komponentenverzeichnis von .NET oder aber im Modulverzeichnis befinden. Der Assembly-Name darf mit oder ohne „.dll“ am Ende angegeben werden.
ScriptsToProcess	Eine Liste von Zeichenketten in der Form @("Skriptname.ps1", " Skriptname.ps1")	Liste von PowerShell-Skripten, die ausgeführt werden sollen, bevor der Import durchgeführt wird
TypesToProcess	Eine Liste von Zeichenketten in der Form @("Typ. ps1xml", "Typ. ps1xml")	Liste von PowerShell-Typdateien (.ps1xml), die mit Hilfe von Update-TypeData ausgeführt werden sollen
FormatsToProcess	Eine Liste von Zeichenketten in der Form @("Typ. ps1xml", "Typ. ps1xml")	Liste von PowerShell-Typdateien (.ps1xml), die mit Hilfe von Update-FormatData ausgeführt werden sollen
NestedModules	Eine Liste von Zeichenketten in der Form @("Datei.dll", "Skript.ps1")	Liste von Skript- und Binärdateien (.ps1, .psm1, .psd1, .dll), die zum Modul gehören. Die Skriptdateien werden global verfügbar gemacht („Dot-Sourcing“) und die in den DLLs enthaltenen Commandlets und Provider werden aktiviert.
ModuleToProcess	String	Name des untergeordneten Moduls, das das Hauptmodul werden soll. Wenn hier nichts angegeben ist, wird das Manifest selbst zum Hauptmodul.

***Tabelle 49.1**: Elemente eines Modul-Manifests (Forts.)*

Schlüssel	Datentyp	Beschreibung
Exported-Functions	Eine Liste von Zeichenketten in der Form @("Funktionsname", "Funktionsname")	Liste von Funktionen, die das Modul exportieren soll. Das Exportieren ist die Voraussetzung dafür, dass eine Funktion in der PowerShell interaktiv oder in Skripten genutzt werden kann. Nicht exportierte Funktionen können nur innerhalb des Moduls selbst verwendet werden. Meist exportiert man alle Funktionen, was man durch die Angabe „*“ festlegt. Eine leere Zeichenkette "" bedeutet, dass keine Funktionen exportiert werden sollen. In anderen Fällen ist die Liste der zu exportierenden Commandlets explizit anzugeben.
ExportedVariables	Eine Liste von Zeichenketten in der Form @("Variablenname", "Variablenname")	Liste von Variablen, die das Modul exportieren soll. Das Exportieren ist die Voraussetzung dafür, dass Variablen in der PowerShell interaktiv oder in Skripten genutzt werden können. Nicht exportierte Variablen können nur innerhalb des Moduls selbst verwendet werden. Meist exportiert man alle Variablen, was man durch die Angabe „"*“ festlegt. Eine leere Zeichenkette "" bedeutet, dass keine Variable exportiert werden soll. In anderen Fällen ist die Liste der zu exportierenden Variablen explizit anzugeben.

***Tabelle 49.1**: Elemente eines Modul-Manifests (Forts.)*

Schlüssel	Datentyp	Beschreibung
ExportedAliases	Eine Liste von Zeichenketten in der Form @("Aliasname", "Aliasname")	Liste von Aliasen, die das Modul exportieren soll. Das Exportieren ist die Voraussetzung dafür, dass Aliase in der PowerShell interaktiv oder in Skripten genutzt werden können. Nicht exportierte Aliase können nur innerhalb des Moduls selbst verwendet werden. Meist exportiert man alle Aliase, was man durch die Angabe „*“ festlegt. Eine leere Zeichenkette "" bedeutet, dass kein Alias exportiert werden soll. In anderen Fällen ist die Liste der zu exportierenden Aliasen explizit anzugeben.
FileList	Eine Liste von Zeichenketten in der Form @("Datei.dll", "Skript.ps1", "Text.txt")	Eine Liste aller Dateien, die zu dem Modul gehören
PrivateData	Object	Daten, die an das Hauptmodul übergeben werden sollen

***Tabelle 49.1**: Elemente eines Modul-Manifests (Forts.)*

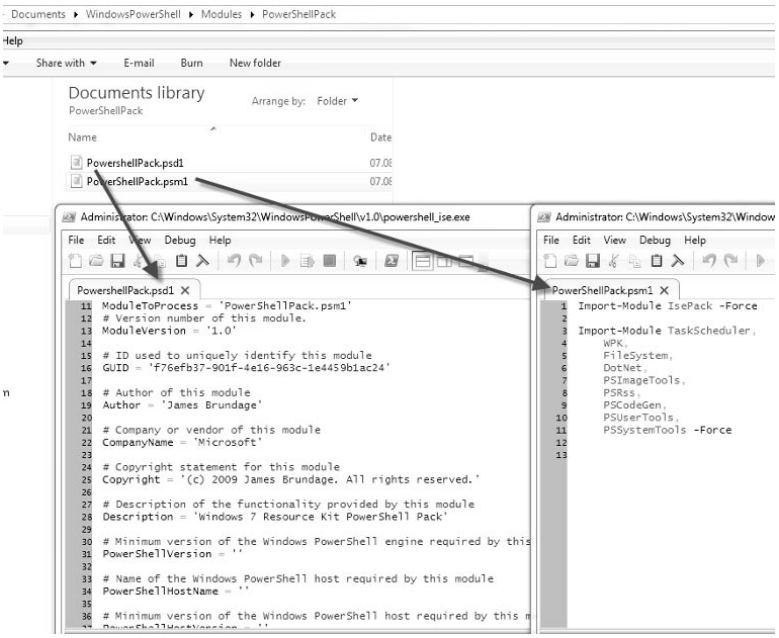
Versionsanforderungen

Die Anforderungen an Versionen, die im Manifest hinterlegt sind, werden tatsächlich beim Importieren des Moduls geprüft. Wenn zum Beispiel dort gefordert ist, dass .NET Framework 4.0 installiert ist, dies aber nicht erfüllt ist, führt `Import-Module` zum Fehler „The module 'C:\Users\hs.ITV\Documents\WindowsPowerShell\Modules\WWWings_ProviderAndCommandlets\WWWings_ProviderAndCommandlets.psd1' requires the following version of the .NET Framework: 4.0. The required version is not installed.“

Dachmodule

Ein Modul kann als „Dachmodul“ für andere unabhängige Module fungieren mit der Zielsetzung, eine Reihe von Modulen entweder einzeln oder zusammen importieren zu können. Dies macht das PowerShellPack vor. Das Modul „PowerShellPack“ ist ein Skriptmodul, das nur aus `Import-Module`-Befehlen für andere Module besteht.

Abbildung 49.4
Aufbau des Moduls „PowerShellPack“



Es macht aber einen kleinen Unterschied, ob man `Import-Module PowerShellPack` oder einen Import eines Einzelmoduls durchführt: Die einzelnen Commandlets verweisen entweder auf das Einzelmodul oder das Dachmodul (siehe Abbildung).

Abbildung 49.5
Auswirkungen des Imports eines Dachmoduls versus Import eines Einzelmoduls

```
PS C:\Users\hs.ITV> Import-Module FileSystem
PS C:\Users\hs.ITV> Get-command mount-specialfolder | fl modulename, {$_.module.modulebase}

ModuleName           : FileSystem
$_.module.modulebase : C:\Users\hs.ITV\Documents\WindowsPowerShell\Modules\FileSystem


PS C:\Users\hs.ITV> Import-Module PowerShellPack
PS C:\Users\hs.ITV> Get-command mount-specialfolder | fl modulename, {$_.module.modulebase}

ModuleName           : PowerShellPack
$_.module.modulebase : C:\Users\hs.ITV\Documents\WindowsPowerShell\Modules\PowerShellPack
```

Beispiel

Es soll ein Manifest für ein Modul mit den nachfolgend dargestellten Bestandteilen erstellt werden. Dabei haben die Dateien folgende Bedeutung:

- „ModulInit.ps1“ ist ein Skript, das vor dem Importieren ausgeführt werden soll (es zeigt einen Hinweistext).
- „Access_Provider.dll“ ist ein PowerShell-Provider.
- „WWW_Commandlets.dll“ implementiert zahlreiche Commandlets.
- „WWWings.DALUtil.dll“,„WWWings.DSZ.CS.dll“ und „WWWings.GL.CS.dll“ sind Bibliotheken, die „WWW_Commandlets.dll“ benötigt. Sie implementieren keine eigenen Commandlets.

- „WWingsVersion6.mdb“ ist eine Beispieldatenbank für „Access_Provider.dll“.
- „StartCommandlets.ps1“ und „StartProvider.ps1“ enthalten Skripte, die „Access_Provider.dll“ und „WWW_Commandlets.dll“ testen.

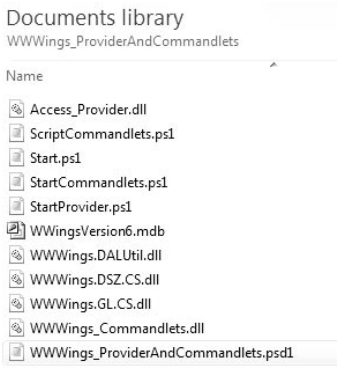


Abbildung 49.6

Das folgende Listing zeigt die zugehörige Manifestdatei.

Listing 49.1
Beispiel für ein Modul-Manifest

```
#
# Module manifest
# Autor: Dr. Holger Schwichtenberg
# 18/01/2010
#

@{

# Script module or binary module file associated with this manifest
ModuleToProcess = 'WWWings_Commandlets.dll'

# Script files (.ps1) that are run in the caller's environment
prior to importing this module
ScriptsToProcess = @("ModulInit.ps1"  )

# Modules to import as nested modules of the module specified in
ModuleToProcess
NestedModules     = @("Access_Provider.dll", "WWWings_
Commandlets.dll", "ScriptCommandlets.ps1", "StartProvider.ps1",
"StartCommandlets.ps1")

# Version number of this module.
ModuleVersion = '1.0'

# ID used to uniquely identify this module
GUID = '0093ae2d-89e4-494c-81a6-35881fafb6f2'

# Author of this module
Author = 'Dr. Holger Schwichtenberg'
```

```
# Company or vendor of this module
CompanyName = 'www.IT-Visions.de'

# Copyright statement for this module
Copyright = '(c) 2009 www.IT-Visions.de. All rights reserved.'

# Description of the functionality provided by this module
Description = 'Modul für World Wide Wings PowerShell-
Erweiterungen.'

# Minimum version of the Windows PowerShell engine required by this
module
PowerShellVersion = '2.0'

# Name of the Windows PowerShell host required by this module
PowerShellHostName = ''

# Minimum version of the Windows PowerShell host required by this
module
PowerShellHostVersion = ''

# Minimum version of the .NET Framework required by this module
DotNetFrameworkVersion = '3.5'

# Minimum version of the common language runtime (CLR) required by
this module
CLRVersion = '2.0'

# Processor architecture (None, X86, Amd64, IA64) required by this
module
ProcessorArchitecture = ''

# Modules that must be imported into the global environment prior
to importing this module
RequiredModules = @()

# Assemblies that must be loaded prior to importing this module
RequiredAssemblies = @("System.Windows.Forms.dll", "WWWings_
Commandlets.dll")

# Type files (.ps1xml) to be loaded when importing this module
TypesToProcess = @()

# Format files (.ps1xml) to be loaded when importing this module
FormatsToProcess = @()

# Functions to export from this module
FunctionsToExport = '*'
```

```
# Cmdlets to export from this module
CmdletsToExport = 'Get-Flug'

# Variables to export from this module
VariablesToExport = '*'

# Aliases to export from this module
AliasesToExport = '*'

# List of all files packaged with this module
FileList = @("Access_Provider.dll", "ModulInit.ps1","WWW_
Commandlets.dll", "StartCommandlets.ps1",  "StartProvider.ps1",
"WWingsVersion6.mdb", "WWWings.DALUtil.dll",
"WWWings.DSZ.CS.dll","WWWings.GL.CS.dll" )

# Private data to pass to the module specified in ModuleToProcess
PrivateData = ''

}
```



Das Commandlet `New-ModuleManifest` hilft beim Erstellen eines Manifests. Das Commandlet fragt die wichtigsten Name-Werte-Paare für das Manifest interaktiv ab. Alternativ kann man die Werte auch als Parameter des Commandlets angeben.

```
PS C:\Users\hs.ITV> New-ModuleManifest

cmdlet New-ModuleManifest at command pipeline position 1
Supply values for the following parameters:
Path: t:\ModulName.psd1
NestedModules[0]: Start.ps1
NestedModules[1]: Commandlets.dll
NestedModules[2]:
Author: Dr. Holger Schwichtenberg
CompanyName: www.IT-Visions.de
Copyright: (C) 2010
ModuleToProcess:
Description: Beispielmodul
TypesToProcess[0]:
FormatsToProcess[0]:
RequiredAssemblies[0]:
FileList[0]:
PS C:\Users\hs.ITV>
```

***Abbildung 49.7**: Interaktive Nutzung von New-ModuleManifest*

A Anhang

A.1 Crashkurs „Objektorientierung“

Systemadministratoren sind oft mit dem Konzept der Objektorientierung (OO), das aus der Welt der Softwareentwicklung stammt, nicht vertraut. Dieser Exkurs bietet eine kompakte Einführung in die Basiskonzepte der Objektorientierung. Wenn Sie damit bereits vertraut sind, überspringen Sie diesen Exkurs.

Was ist ein Objekt?

Objekt

In der Softwareentwicklung hat sich ein Konzept durchgesetzt, das sich Objektorientierung nennt. Dabei programmiert man mit sogenannten Objekten. Dieser Objektbegriff ist hier ähnlich zu sehen wie der Objektbegriff in der menschlichen Sprache:

- Ein Objekt hat Eigenschaften wie beispielsweise einen Namen, eine Farbe und eine Größe.
- Man kann mit einem Objekt Dinge tun, zum Beispiel es bewegen oder seine Farbe verändern.
- Objekte können Signale aussenden, beispielsweise Töne erzeugen.

Mitglieder eines Objekts

Ein Objekt kann drei verschiedene Arten von Bestandteilen haben. Diesen Bestandteilen hat man folgende Fachbegriffe gegeben:

- Attribute (auch: Eigenschaften), engl.: Attributes/Properties
- Methoden, engl.: Methods
- Ereignisse, engl.: Events

Man sagt, ein Objekt **hat** (oder **besitzt**) Attribute, Methoden und Ereignisse, wobei ein Objekt jeweils **beliebig viele** Attribute, Methoden oder Ereignisse besitzen kann. Es ist nicht zwingend notwendig, dass ein Objekt alle drei Arten besitzen muss. Zusammenfassend werden diese drei Konzepte als „Mitglieder“ eines Objekts bezeichnet, damit man nicht immer „Attribute, Methoden und Ereignisse“ aufzählen muss.

Attribute, Methoden, Ereignisse

Attribute sind die Daten des Objekts; sie können gelesen oder gesetzt werden. Methoden können aufgerufen werden und dabei Parameter übergeben bekommen. Sie können entweder einen, keinen oder mehrere Werte zurückliefern. Ereignisse löst das Objekt selbst aus. Auf

Ereignisse kann man reagieren, indem man Programmcode hinterlegt, der für den Fall der Auslösung des Ereignisses abgearbeitet werden soll. Diesen Programmcode nennt man eine Ereignisbehandlungsroutine.

Objekte in der realen Welt sind z.B. ein Haus, ein Baum, ein Tisch, ein Auto oder ein Mensch (an dieser Stelle soll der Begriff „Objekt" als Oberbegriff zu sehen und daher die Versachlichung des Menschen gestattet sein). Bei der objektorientierten Programmierung ist es üblich, Programm-Objekte zu bilden, die realen Objekten entsprechen: ein Haus-Objekt für ein Haus, ein Baum-Objekt für einen Baum etc. Es ist aber natürlich auch möglich, Programm-Objekte zu bilden, die es in der Realität nicht gibt.

Objekte bei der PowerShell

Die PowerShell arbeitet durchweg mit Objektorientierung und Objekten. Objekte bei der PowerShell sind zum Beispiel eine Datei (engl. File), ein Benutzer (engl. User), eine Domäne (engl. Domain) oder eine Netzwerkkarte (engl. Network Adapter). Dies sind Objekte, die in der Realwelt „Betriebssystem" vorkommen. Daneben gibt es bei der PowerShell auch Objekte wie `System.Environment`, das verschiedene Funktionen zusammenfasst, die in der Realwelt so nicht zusammengefasst existieren.



Ein Konstruktor ist Programmcode in einem Objekt, der beim Erzeugen eines Objekts („Instanziieren") aufgerufen wird. Ein Konstruktor enthält Initialisierungscode. Ein Destruktor ist das Gegenteil: Er wird durch die Vernichtung des Objekts aufgerufen, um „aufzuräumen".

Was ist eine Klasse?

Klassen

Es gibt Objekte (z.B. Datei, Benutzer), von denen nicht nur ein, sondern mehrere oder sogar beliebig viele Exemplare existieren können. Mit dem Begriff **Klasse** fasst man alle gleichartigen Objekte zusammen. Ein Beispiel: In der Klasse „Datei" gibt es die Objekte „abc.doc", „xyz.txt" und „rst.xls". Ein Objekt bezeichnet man auch als **Instanz** einer Klasse. Eine Klasse definiert, welche Mitglieder (Attribute, Methoden und Ereignisse) jede Instanz oder jedes Exemplar der Klasse haben soll. Und eine Klasse enthält auch den Programmcode, der ausgeführt werden soll, wenn eine Methode aufgerufen wird. Dieser Programmcode ist in allen Instanzen einer Klasse gleich (und wird daher auch nur einmal im Speicher abgelegt). Eine Klasse ist eine Schablone zur Erzeugung von Objekten. Synonym zur Klasse wird oft auch der Begriff **Objekttyp** verwendet.

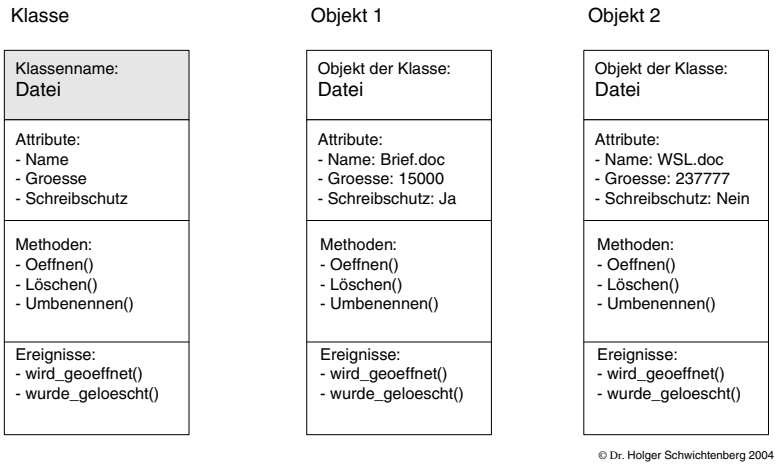


Abbildung A.1
Klasse vs. Objekt



Es ist eine Konvention, dass man nach dem Namen einer Methode und eines Ereignisses ein rundes Klammernpaar schreibt, unabhängig davon, ob hier Parameter erwartet werden. Zudem sind Umlaute in Namen zu vermeiden.

Instanzen

In der objektorientierten Programmierung geht man so vor, dass man zunächst eine Klasse definiert und danach Instanzen einer Klasse erzeugt, die man dann zur Programmierung verwendet. Den Vorgang, aus einer Klasse eine Instanz zu bilden, nennt man **Instanziierung** (teilweise in der Literatur auch mit einem i geschrieben: **Instanzierung**). Man sagt, eine Klasse wird **instanziiert**. Die Klasse ist eine Schablone für Objekte, die aus rohem Computerspeicher ein Objekt einer bestimmten Form erzeugt.

Beispiele für Instanzen sind:

- Der Opel mit dem Kennzeichen „E-GO123“ ist eine Instanz der Klasse `Auto`.
- Der Benutzer mit dem Namen „HS“ ist eine Instanz der Klasse `Benutzer`.



Abbildung A.2
Ein Objekt entsteht aus dem rohen Speicher durch die Anwendung einer Klasse als Schablone.

Klassen bei der PowerShell

Bei der PowerShell arbeiten Sie mit Klassen, die Microsoft (oder andere Hersteller) definiert haben. Das Windows-Betriebssystem und das Microsoft .NET Framework erzeugen automatisch im laufenden Betrieb unzählige Instanzen dieser Klassen. Ihre Aufgabe als Skript-

entwickler ist es lediglich, die gewünschte Instanz zu finden und gemäß Ihren Anforderungen auszulesen oder zu verändern. In vielen Fällen erzeugen Sie auch selbst Instanzen, z.B. wenn Sie eine neue Datei oder einen neuen Benutzer anlegen. Mit der Definition eigener Klassen haben Sie zunächst nichts zu tun: Dies erfordert eine höherwertige Programmiersprache wie C# oder Visual Basic und kommt nur im *Teil 4 „Profiwissen – Erweitern der PowerShell"* vor.

Klassennamen

Die von Microsoft und anderen Herstellern definierten Klassen haben englische Namen (oder Kunstnamen), die oftmals aus mehreren Wörtern bestehen und zum Teil abgekürzt sind. Beispiele für Klassennamen sind:

- `Process`
- `DriveInfo`
- `FileInfo`
- `Service`
- `ManagementObject` (manchmal werden die Begriffe Class oder Object als Teil des Namens einer Klasse verwendet, was verwirrend sein kann)

Damit Sie Klassennamen im Text sofort erkennen, sind diese (ebenso wie Befehle, Variablen etc.) in diesem Buch in einer anderen Schriftart dargestellt.



In der objektorientierten Programmierung gibt es einen Unterschied zwischen einer *Schnittstelle* (engl. *Interface*) und einer Klasse/einem Objekt. Da dieser Unterschied jedoch im Rahmen der in diesem Buch vorgestellten Beispiele kaum Bedeutung hat, wird im Folgenden auf die nähere Erläuterung und Verwendung des Begriffs Schnittstelle verzichtet.

Kapselung

Als Kapselung bezeichnet man in der objektorientierten Welt die Fähigkeiten einer Klasse, darüber zu entscheiden, wer die Mitglieder der Klasse sehen und nutzen darf. Hier werden nicht, wie man es aus dem Betriebssystem kennt, Zugriffsrechte auf Ebene von Benutzern oder Gruppen vergeben, sondern es wird recht pauschal definiert, wie andere Klasse damit umgehen dürfen. Die beiden Standardzugriffsmöglichkeiten sind:

- Privat (Private): Nur die Klasse selbst darf dieses Mitglied nutzen. Es handelt sich um interne Daten oder interne Routinen, die die Außenwelt nichts angehen oder die diese nicht sinnvoll nutzen kann.
- Öffentlich (Public): Auch andere Klassen dürfen dieses Mitglied nutzen.

Darüber hinaus gibt es die Zugriffseinstellung „geschützt“ (Protected), die man aber nur kennen muss, wenn man selbst Klassen definiert. Private und Public kann man auch auf Klassen anwenden.

Objektbeziehungen

In der Realität sind Objekte miteinander verbunden: Ein Baum hat Äste und Zweige, ein Auto hat einen Motor und Räder etc. Auch bei Betriebssystem-Objekten gibt es solche Zusammenhänge: Eine Domäne enthält Benutzer und Computer, ein Computer besteht aus einem Prozessor und mehreren Festplatten usw. Diese Zusammenhänge müssen durch Programmier-Objekte abgebildet werden.

Objekte verweisen auf andere Objekte

Die Lösung zur Abbildung dieser Zusammenhänge ist einfach: Ein Mitglied eines Objekts kann selbst wieder ein Objekt sein. Zum Beispiel kann ein Attribut ein Objekt beinhalten oder eine Methode ein Objekt als Ergebnis liefern.

Daraus ergibt sich eine Objekthierarchie, die man in Form eines Objektbaums darstellen kann. Wir zeigen an einigen Stellen diese Objektbäume, weil sie gut geeignet sind, die Zusammenhänge zwischen den Objekten zu verstehen.

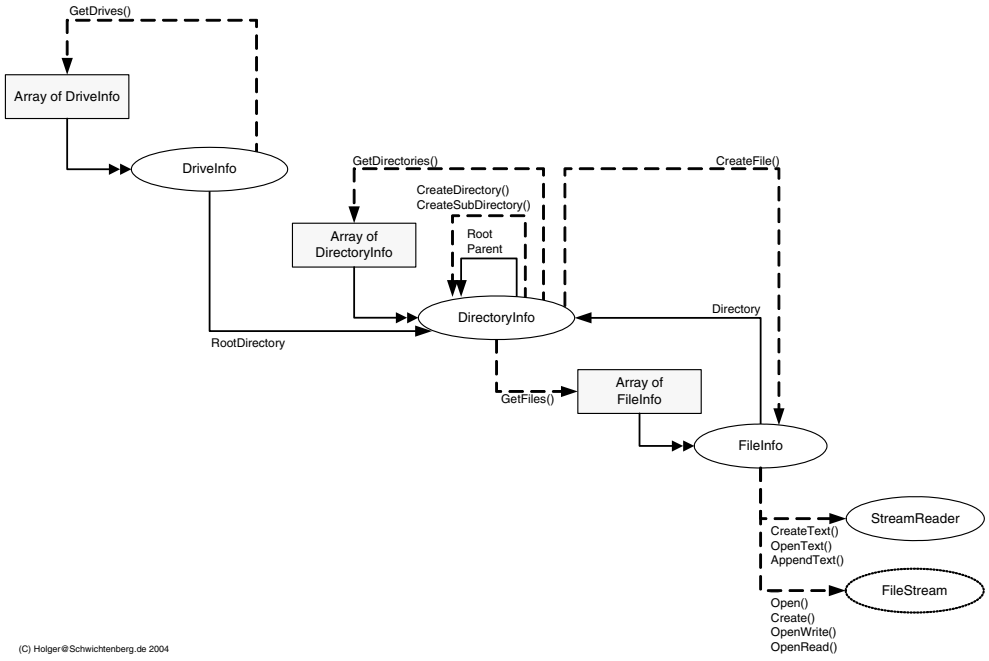


***Abbildung A.3**: Objektbaum für die Klassen, die ein Dateisystem repräsentieren*

Die möglichen Zusammenhänge zwischen den Objekten sind in der Definition einer Klasse hinterlegt. In der Regel steht dort geschrieben, welche Arten von Objekten miteinander verbunden sind. Es ist aber auch möglich, dass in einer Klasse definiert ist, dass ein Mitglied ein beliebiges Objekt enthalten kann.

Befehlskette Bei den Mitgliedern einer Klasse muss man zwischen Mitgliedern unterscheiden, die einfache Daten – wie eine Zahl oder eine Zeichenkette – liefern, und Mitgliedern, die ein Objekt liefern. Mitglieder, die selbst wieder Objekte sind, haben nämlich wieder Mitglieder, so dass eine lange Befehlskette entstehen kann.

```
Objekt1.UnterObjekt2.UnterObjekt3.UnterObjekt4.Methode_von_
Unterobjekt4()
```



Die zur Darstellung eines Objektbaums verwendete Notation lässt sich in fünf einfachen Regeln erklären:

1. Einzelne Objekte sind durch Ovale dargestellt. In dem Oval steht der Name der Klasse.
2. Objektmengen (also Listen von mehreren Objekten) sind durch Rechtecke dargestellt. In dem Rechteck steht der Name der Klasse.
3. Durchgezogene Linien sind Attribute, die auf andere Objekte verweisen.
4. Gestrichelte Linien sind Methoden, die ein Objekt als Ergebnis liefern.
5. Eine einfache Pfeilspitze bedeutet, dass auf genau ein einzelnes Objekt verwiesen wird (1:1-Beziehung). Eine doppelte Pfeilspitze bedeutet, dass eine Objektmenge beliebig viele Objekte dieses Typs enthalten kann (1:n-Beziehung).

Vererbung

Vererbung in der Welt der Objektorientierung bedeutet, dass eine Klasse die Mitglieder (Attribute, Methoden, Ereignisse) einer anderen Klasse übernimmt. Ziel ist die Wiederverwendung: Wenn Klassen sich sehr ähnlich sind, dann soll man nicht alles das, was schon einmal in einer Klasse definiert wurde, erneut definieren müssen. Vererbung ermöglicht es, Attribute, Methoden und Ereignisse, die mehreren Klassen gemein sind, an einer zentralen Stelle zu definieren.

An der Vererbung (in der Welt der PowerShell) sind immer zwei Klassen beteiligt: die vererbende Klasse und die erbende Klasse. Die vererbende Klasse wird auch Oberklasse, Basisklasse, Superklasse oder Elternklasse genannt. Die erbende Klasse wird Unterklasse, abgeleitete Klasse, Subklasse oder Kinderklasse genannt. Unterklassen können wiederum Oberklassen für andere Klassen sein. Daraus ergibt sich eine baumartige *Vererbungshierarchie* (auch Klassenhierarchie genannt). Die visuelle Darstellung heißt Vererbungsdiagramm oder Klassendiagramm.



Es gibt Programmierumgebungen, in denen eine Klasse mehrere Elternklassen besitzen kann. Dies bezeichnet man als Mehrfachvererbung. Mehrfachvererbung wird in .NET und in der PowerShell nicht unterstützt. Hier gibt es nur Einfachvererbung.

Die folgende Abbildung zeigt ein konkretes Vererbungsbeispiel aus dem Microsoft .NET Framework. Die Klasse `DirectoryInfo` repräsentiert ein Verzeichnis (Ordner) im Dateisystem. Die Klasse `FileInfo` repräsentiert eine einzelne Datei. Beide Elemente im Dateisystem haben zahlreiche gemeinsame Eigenschaften, z.B. Name, Pfad (`FullName`), Erzeugungsdatum (`CreationTime`) und letzter Zugriffszeitpunkt (`LastAccessTime`). Ebenso gibt es gemeinsame Aktionen, die man durch Methoden ausdrücken kann, z.B. Löschen (`Delete()`) und Aktualisieren (`Refresh()`). Diese Gemeinsamkeiten sind nicht in den Klassen `FileInfo` oder `DirectoryInfo` definiert, sondern in der gemeinsamen Basisklasse `FileSystemInfo`, von der beide Klassen erben. Die Klasse `FileSystemInfo` hat noch die Besonderheit, dass es keinen Sinn macht, von ihr selbst Instanzen zu erzeugen. Daher hat sie den Zusatz „`abstract`“. Eine abstrakte Klasse ist eine Klasse, von der man keine Instanzen erzeugen kann. Die Klassen `FileInfo` oder `DirectoryInfo` haben den Zusatz `sealed`, was bedeutet, dass man von diesen Klassen nicht mehr erben darf.

Die Klasse `FileSystemInfo` hat wiederum eine Basisklasse namens `MarshalByRefObject`. Die hier implementierten Mitglieder haben nur Bedeutung für die .NET-Infrastruktur und bleiben daher hier außerhalb der Betrachtung. Wichtiger ist dann aber wieder die Basisklasse `Object`, die über `MarshalByRefObject` steht. Die Klasse `Object` ist die ultimative Basisklasse aller .NET-Klassen. Jede .NET-Klasse muss von ihr erben. Sie stellt einige wenige Mitglieder bereit, die jedes .NET-Objekt braucht. Das wichtigste Mitglied ist die Methode `ToString()`, die es ermöglicht, dass man zu jedem Objekt eine Zeichenkette ausgeben kann, die Informationen über das Objekt liefert (oft allerdings nur den Klassennamen).

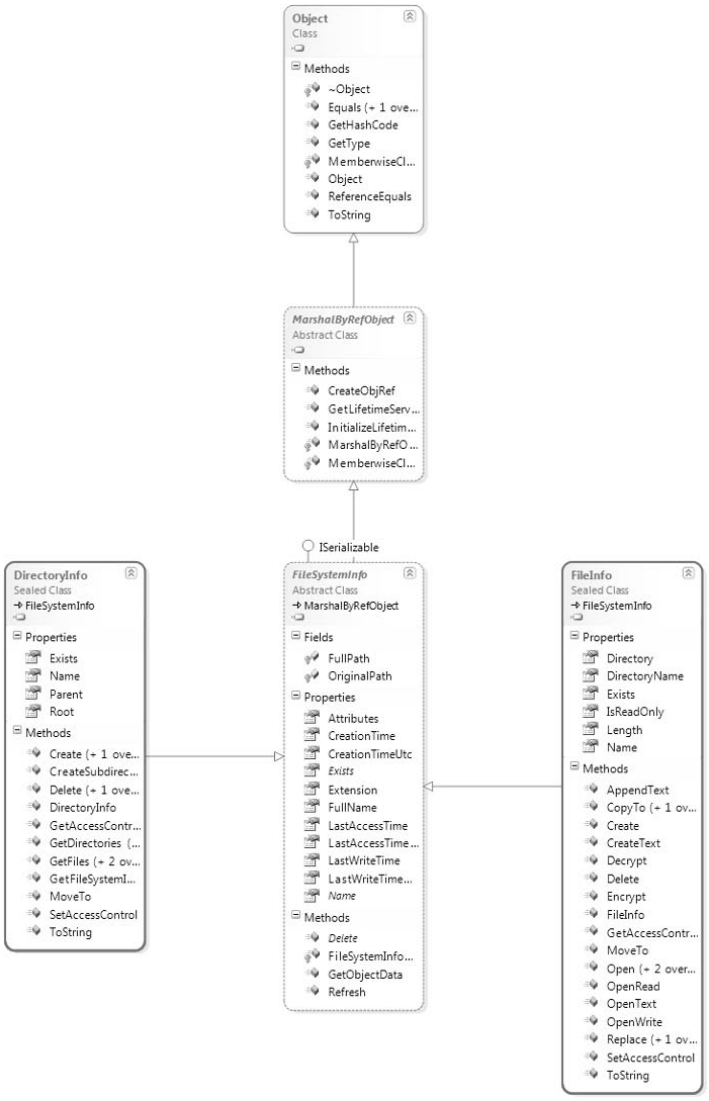


Abbildung A.4 *Vererbungshierarchie für die Klassen FileInfo und DirectoryInfo im .NET Framework, erstellt mit der Klassendiagrammfunktion in Visual Studio 2008*



In der Grafik sind alle Mitglieder der Klasse dargestellt, auch wenn diese nicht „Public" sind. In der PowerShell können Sie ohne Weiteres nur die öffentlichen Mitglieder nutzen. Diese erkennt man daran, dass sie kein kleines Schlüsselsymbol neben dem Namen haben.

Weitere Konzepte

Es gibt weitere Konzepte der Objektorientierung (z.B. Schnittstellen und Polymorphismus), die jedoch nicht so wichtig für die PowerShell sind.

B Crashkurs „.NET Framework“

.NET

Die Windows PowerShell basiert auf dem Microsoft .NET Framework. Man kann wesentlich erfolgreicher mit der PowerShell arbeiten, wenn man Grundlagen des .NET Frameworks beherrscht.

Das Microsoft .NET Framework ist eine Laufzeitumgebung für Anwendungen, die mit einer sogenannten .NET-fähigen Programmiersprache (z.B. C#, Visual Basic ab Version 7.0, J#) entwickelt wurden.

Die aktuelle .NET-Version zum Erscheinen dieses Buchs ist Version 3.5 Service Pack 1. Von Version 4.0 existiert eine Vorabversion im Stadium Beta 2. Die Ausführungen beziehen sich alle auf .NET 3.5 Service Pack 1.

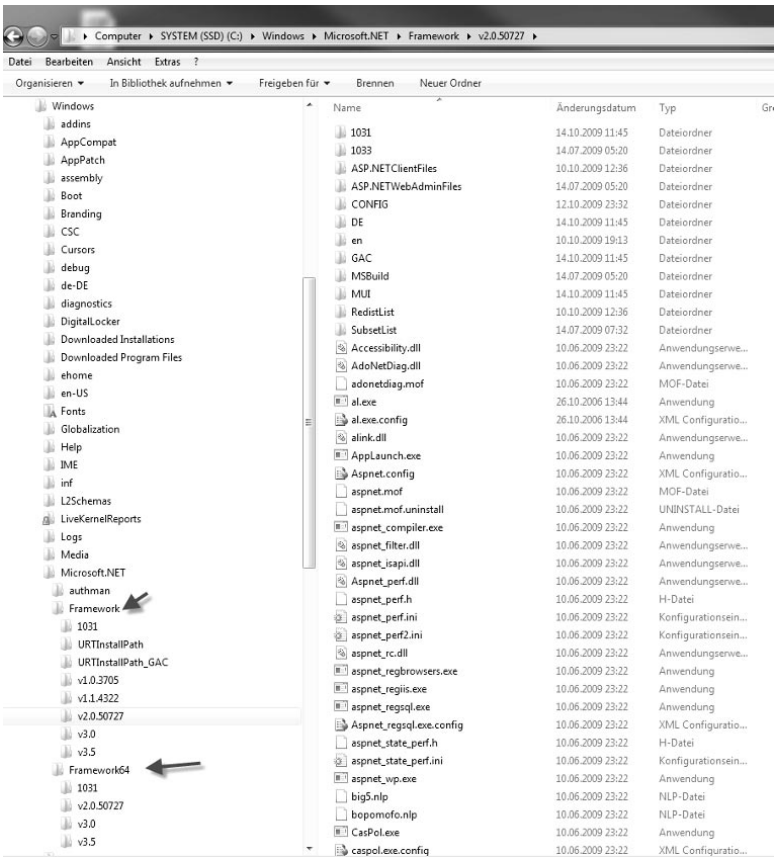


In Windows Vista und Windows Server 2008 R2 ist das .NET Framework in den Versionen 2.0 und 3.0 im Standardinstallationsumfang des Betriebssystems enthalten. In Windows 7 und Windows Server 2008 R2 inbegriffen ist .NET 3.5 Service Pack 1.

10.000 Klassen

Das Microsoft .NET Framework ist ein extrem komplexes Gebilde. Zum Versionsstand 3.5 besteht es aus rund 10000 öffentlichen Klassen (nicht zu verwechseln mit den rund 12000 WMI-Klassen), es gibt mehr als 40 Programmiersprachen und weit mehr als 1000 Werkzeuge und Erweiterungen von Drittanbietern. Die Dokumentation umfasst mehr als schätzungsweise 600.000 Seiten und eine seriöse Schulung dauert mindestens zehn Wochen. Jeglicher Versuch, hier auf drei Seiten .NET komplett erklären zu wollen, ist zum Scheitern verurteilt. Dieses Buch kann hier nur einige Begriffe in den Raum werfen, die Erinnerungen an Ihnen bereits bekannte Konzepte aus Java oder anderen objektorientierten Hochsprachen wecken.

Abbildung B.1 *Ein Blick in den .NET Framework-Installationsordner*





Eine kompakte Einführung in .NET auf rund 750 Seiten finden Sie vom gleichen Autor in [SCH07].

B.1 Was ist das .NET Framework?

Das .NET Framework ist eine plattformunabhängige und programmiersprachenunabhängige Softwareentwicklungsplattform mit Unterstützung für die Programmierparadigmen:

- Objektorientierung (OOP)
- Komponentenorientierung (COP)
- Serviceorientierung (SOA)

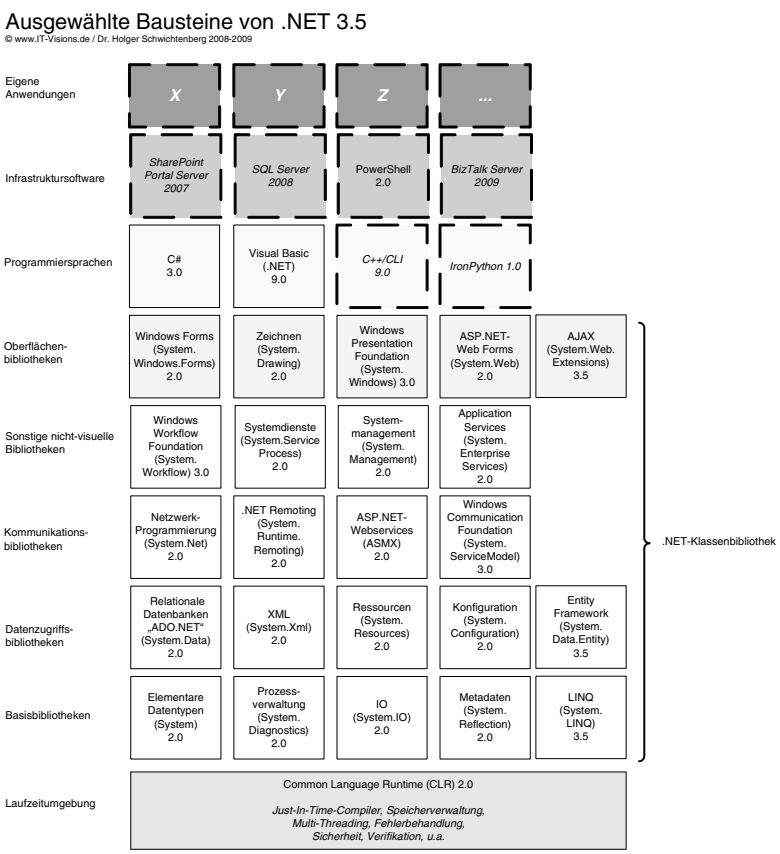


Abbildung B.2
Wichtige Bausteine und Abstraktionsebenen des .NET Frameworks

Common Intermediate Language

Plattformunabhängig bedeutet, dass .NET-Anwendungen ohne Neukompilierung auf jedem Prozessor und jedem Betriebssystem laufen können. .NET-Anwendungen liegen normalerweise nicht in Maschinencode, sondern in einem neutralen Zwischencode (Common Intermediate Language – CIL) vor. CIL wird erst zur Laufzeit durch den Just-In-Time-Compiler der .NET-Laufzeitumgebung (Common Language Runtime – CLR) in plattformspezifischen Maschinencode umgewandelt.

Common Type System

Programmiersprachenunabhängig bedeutet, dass .NET-Anwendungen in einer Vielzahl verschiedener Programmiersprachen geschrieben werden können. Die Sprachcompiler müssen sich dazu lediglich an Regelwerke halten, die Common Type System (CTS) und Common Language Specification (CLS) heißen. Innerhalb einer Anwendung können verschiedene Programmiersprachen gemischt werden.

Beliebige Anwendungstypen

Softwareentwicklungsplattform bedeutet, dass .NET zur Entwicklung von Software eingesetzt wird. Mit .NET kann man fast alle Arten von Software (Konsolenanwendungen, grafische Desktop-Anwendungen, Webanwendungen, Dienste) entwickeln. Ausgenommen sind Betriebssystemtreiber.

Klassen und Objekte

Objektorientierung bedeutet, dass das .NET Framework konsequent die Konzepte *Klasse*, *Objekt*, *Schnittstellen* und *Vererbung* einsetzt. Objekte sind Instanzen von Klassen. Klassen erben von maximal einer Klasse (Einfachvererbung). Mehrfachvererbung wird nicht unterstützt. Klassen haben Schnittstellen, die sie an ihre Instanzen weitergeben. Eine Klasse bzw. Schnittstelle besteht aus Attributen (Daten), Methoden (Operation) und Ereignissen.

Assembly

Komponentenorientierung bedeutet, dass das .NET Framework die Zerlegung von Software in wiederverwendbare Bausteine unterstützt. Diese Bausteine heißen Assemblies und sind von außen betrachtet Bibliotheken (*.dll*) oder startbare Anwendungen (*.exe*).

Services

Serviceorientierung bedeutet, dass das .NET Framework die Entwicklung verteilter Systeme (Verteilung von Programmcode auf mehrere Rechnersysteme) mit lose gekoppelten Diensten unterstützt.

Das .NET Framework ist der Nachfolger des Component Object Model (COM), wobei es mit diesem nicht mehr viel gemein hat. Das .NET Framework ist in weiten Teilen unter der Bezeichnung Common Language Infrastructure (CLI) bei der ECMA und der ISO standardisiert.

Eigenschaften des .NET Frameworks

Basiseigenschaften

Einige weitere Eigenschaften des .NET Frameworks sollen hier erwähnt werden:

- Die Ausführungsgeschwindigkeit von .NET-Anwendungen ist trotz der Zwischensprache hoch. Sie ist langsamer als in C++ geschriebene Anwendungen, aber weit schneller als Anwendungen, die mit Visual Basic 6.0 oder einer Skriptsprache wie VBScript geschrieben wurden.
- Mehrere Versionen des .NET Frameworks und mehrere Versionen einer in .NET geschriebenen Software können auf einem System (problemlos) koexistieren (Side-by-Side Executing).
- Die Laufzeitumgebung entlastet den Programmierer, indem sie Routineaufgaben automatisch erledigt (z.B. automatische Speicherverwaltung und -bereinigung, Verifikation des Codes) bzw. mächtige Grundfunktionen bereitstellt (z.B. Multithreading, Fehlerbehandlung).
- Die .NET-Klassenbibliothek stellt viele Funktionen bereit und ist konsistenter aufgebaut als die bisherigen C++- und COM-basierten Bibliotheken.

- Das .NET Framework setzt XML-Dateien zur Konfiguration von Anwendungen ein. Der Einsatz der Registrierungsdatenbank ist verpönt.
- Viele .NET-Anwendungen können durch einfaches Kopieren der Programmdateien auf einem System betrieben („installiert“) werden. Man spricht vom sogenannten *XCopy-Deployment*.
- Jede .NET-Assembly enthält zahlreiche Metadaten über die enthaltenen Klassen. Diese Metadaten können durch einen Mechanismus mit Namen *Reflection* ausgewertet werden.
- .NET-Anwendungen werden durch die Laufzeitumgebung in ihrem Wirken beschränkt (Sandkastenprinzip, vgl. Java). In vielen Fällen müssen zunächst explizite Sicherheitsfreigaben erfolgen.
- Schnittstellenverträge sind in .NET weniger streng, so dass Ergänzungen an Komponenten möglich sind und die Komponenten dadurch inkompatibel zu Software werden, die ältere Versionen einer Komponente verwendet.
- Das .NET Framework ist interoperabel zu anderen Plattformen: COM, C/C++ (Win32API), XML-Webservices (Drittanbieter: Java, CORBA).



.NET für Unix, Linux und Mac OS

Mit dem OpenSource-Produkt **Novell Mono** [MON01] steht inzwischen eine kompatible Portierung des .NET Frameworks für andere Betriebssysteme zur Verfügung, so dass viele Anwendungen ohne Neukompilierung portiert werden können.

.NET-Klassen

Die PowerShell basiert auf .NET-Klassen. Ein Nutzer der PowerShell benötigt ein Grundwissen über den Aufbau von .NET-Klassen, wenn er die PowerShell optimal nutzen möchte.

Namensgebung von .NET-Klassen (Namensräume)

Namensräume

Klassen werden in .NET nicht mehr durch GUIDs, sondern durch Zeichenketten eindeutig benannt. Diese Zeichenketten sind hierarchische Namen (siehe *Bild 22.19*). Ein absoluter Klassenname besteht aus dem relativen Namen der Klasse und dem Namensraum (engl. Namespace). Ein Wurzelnamensraum (der Namensraum, der vorne steht im absoluten Klassennamen) kann auch Unternamensräume enthalten, so dass eine Namensraumhierarchie entsteht.

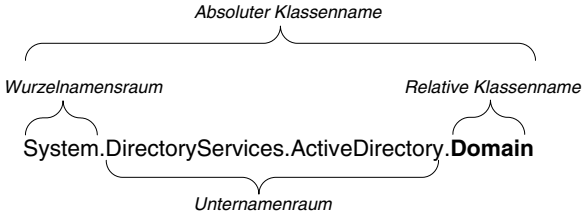


Abbildung B.3 *Beispiel für eine .NET-Klasse mit Namensraumhierarchie*

Der relative Name muss nur innerhalb eines Namensraums eindeutig sein. Über alle Namensräume hinweg kann der Klassenname mehrfach vorkommen, denn der Namensraum ermöglicht im Zweifel die Unterscheidung. Dies ist vergleichbar mit Dateien und Ordnern in einem Dateisystem.

Wurzelnamensräume

Im .NET Framework können beliebig viele Namensraumhierarchien parallel existieren, denn es gibt nicht nur einen, sondern mehrere Wurzelnamensräume. Die .NET-Klassenbibliothek besitzt zwei Wurzelnamensräume: `System` und `Microsoft`. Die speziellen Klassen für die PowerShell befinden sich unterhalb von `System.Management.Automation`.

Im normalen Programmcode (C#, VB.NET etc.) wird eine Klasse über den absoluten Klassennamen oder – wenn der Namensraum vorher eingebunden wurde – über den relativen Klassennamen angesprochen. In der PowerShell muss immer der absolute Klassenname verwendet werden, da es keine Abkürzungsmöglichkeit gibt.

Da es keine zentrale Stelle gibt, welche die Bezeichner für Namensräume vergibt, besteht grundsätzlich die Gefahr, dass es zu doppelten Typnamen kommt. Im Rahmen des CLI-Standards wurde deshalb folgende Notation festgelegt:

```
Firmenname.Technologiename
```

Beispiele für Namensräume sind:

```
Microsoft.Office
PowerSoft.PowerBuilder
Corel.CorelDraw
AddisonWesley.Scripting
```

Es ist auch üblich, den Internetdomänennamen in umgekehrter Reihenfolge zu verwenden, also z.B. `com.Microsoft.Office` oder `de.AddisonWesley.Scripting`.

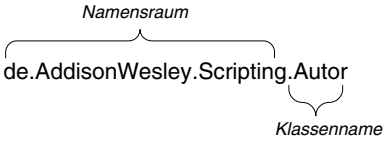


Abbildung B.4 *Vergabe eines Namensraums*

Auch für die Namensgebung von Typen gibt es Regeln, die im CLI-Standard manifestiert sind. Für die Groß-/Kleinschreibung gilt grundsätzlich *PascalCasing*, d.h., ein Bezeichner beginnt grundsätzlich mit einem Großbuchstaben und jedes weitere Wort innerhalb des Bezeichners beginnt ebenfalls wieder mit einem Großbuchstaben. Ausnahmen gibt es lediglich für Abkürzungen, die nur aus zwei Buchstaben bestehen. Diese dürfen komplett in Großbuchstaben geschrieben sein (z.B. `UI` und `IO`). Alle anderen Abkürzungen werden entgegen ihrer normalen Schreibweise in Groß-/Kleinschreibung geschrieben (z.B. `Xml`, `Xsd` und `W3c`).

Es gibt weitergehende Regeln, die aber weit über den Rahmen dieses Buchs hinausführen würden.

Namensräume und Softwarekomponenten

Hierarchien

Ein Namensraum ist eine Gruppierung von Klassen, die in Assemblies (Softwarekomponenten) implementiert sind. Der Namensraum ist unabhängig von dem Namen der Assembly. Ein Namensraum kann in beliebig vielen Assemblies implementiert werden, ebenso wie jede Assembly Typen zu beliebig vielen verschiedenen Namensräumen beisteuern kann. In der nachstehenden Abbildung sind die vertikalen dreidimensionalen Kästen die Assemblies und die horizontalen flachen Kästen die Namensräume. Man sieht, dass `System.IO` in *System.dll* und *mscorlib.dll* implementiert ist, es also keine *System.IO.dll* gibt.

Die Gruppierung, also die Auswahl der Typen, die zu einem Namensraum gehören, sollte nach logischen oder funktionellen Prinzipien erfolgen. Im Gegensatz dazu sollte die Zusammenfassung von Typen zu einer Assembly gemäß den Bedürfnissen zur Verbreitung der Klassen (engl. Deployment) erfolgen.



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

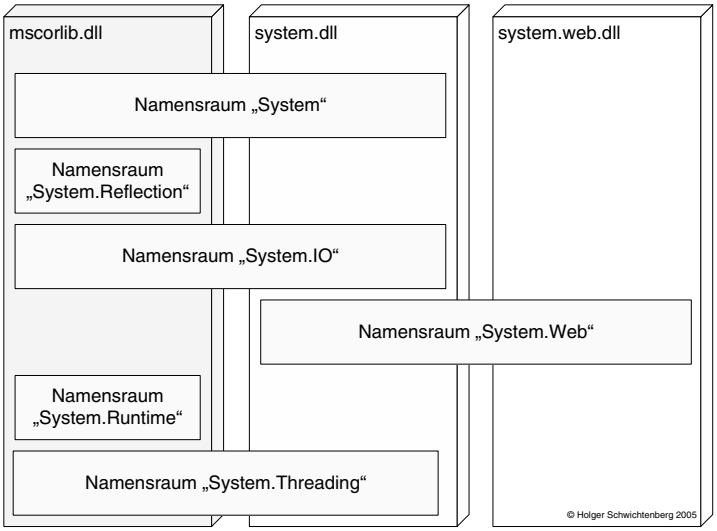


Abbildung B.5 *Der Namensraum ist unabhängig vom Namen der Assembly.*

Referenzierung Um die Klassen aus einem .NET-Namensraum nutzen zu können, muss die Assembly, die diesen Teil des Namensraums implementiert, in der Anwendung, welche die Klasse nutzen möchte, eingebunden (referenziert) werden. Man erstellt Referenzen auf Assemblies, nicht auf Namensräume. Die PowerShell referenziert automatisch alle wichtigen Assemblies der .NET-Klassenbibliothek. Möchte man zusätzliche Assemblies (z.B. auch von Drittanbietern) nutzen, so muss man diese einbinden.

Bestandteile einer .NET-Klasse

Eine .NET-Klasse ist ein Typ im Sinne der objektorientierten Programmierung und besteht daher aus der Vereinigung von Daten und Operationen. Eine .NET-Klasse besteht aus fünf Arten von Mitgliedern: Attribute, Methoden, Ereignisse, Konstruktoren und maximal ein Destruktor.

Attribute

Attribute

Attribute sind Datenmitglieder einer Klasse. Es gibt in .NET zwei Unterarten von Attributen: Felder (engl. Fields) und Eigenschaften (engl. Properties).

Field Fields sind Attribute, bei denen der Nutzer des Objekts direkt einen bestimmten Speicherplatz innerhalb des Objekts beschreibt. Er kann alle dem Datentyp des Felds entsprechenden Werte in das Feld schreiben, ohne dass eine weitere Prüfung erfolgt.

Property Im Gegensatz dazu wird bei einer Property bei jedem Schreib- und Lesezugriff Programmcode ausgeführt, bei dem z.B. Prüfungen durchgeführt werden können. So kann der Gültigkeitsbereich auf beliebige Werte beschränkt werden. Zu einer Property

gehört eine *Getter*-Methode (`Get`) und/oder eine *Setter*-Methode (`Set`). Wie und ob überhaupt die übergebenen Werte verarbeitet werden, ist in diesen Methoden zu implementieren. In der Regel gehört zu einem Property ein privates, also nur innerhalb des Objekts zugängliches Feld.

Indexer

Aus der Sicht des Nutzers der Klasse gibt es fast keinen Unterschied zwischen Fields und Properties. Eine Property kann jedoch Parameter besitzen und wird dann *Indexer* (*indiziertes Attribut*) genannt. Der Lesezugriff sieht dann aus wie der Aufruf einer Methode:

```
element = collection.item(0)
```

Beim Schreibzugriff fällt allerdings schon auf, dass es sich nicht um eine Methode handelt, denn einer solchen könnte kein Wert zugewiesen werden:

```
collection.item(2) = "Guten Tag"
```

Ein Indexer kann als ein Array von Attributen betrachtet werden.

- **Methoden**

Methode

Methoden sind in Klassen implementierte Unterprogramme. Eine Klasse enthält in der Regel eine Implementierung zu den Methoden. Eine Klasse kann aber auch einzelne *abstrakte* Methoden (auch *virtuelle Methoden* genannt) besitzen oder aber komplett abstrakt (auch *rein virtuelle Klasse* genannt) sein. Eine abstrakte Klasse kann nicht instanziiert werden.

Konstruktor

Ein Konstruktor ist eine Programmroutine, die bei der Instanziierung der Klasse aufgerufen wird. Konstruktoren können Parameter haben, die bei der Instanziierung anzugeben sind.

Destruktor

Auch ein Destruktor gehört zu den Programmroutinen. Jede Klasse kann einen Destruktor mit dem Namen `Finalize()` besitzen, der aufgerufen wird, bevor die automatische Speicherverwaltung die Instanz im Speicher freigibt.

- **Ereignisse**

Ereignisse

Nutzer eines Objekts können ihr Interesse an von der Klasse definierten Ereignissen bekunden, indem sie dem Objekt einen Zeiger auf ein Unterprogramm übergeben, welches das Objekt beim Eintritt bestimmter Bedingungen aufruft.

Instanzmitglieder vs. Klassenmitglieder

Mitglieder einer Klasse können den einzelnen Instanzen (Instanzmitglieder) und der Klasse selbst (Klassenmitglieder oder statische Mitglieder genannt) zugeordnet sein. Eine Klasse wird als statische Klasse bezeichnet, wenn sie nur statische Mitglieder besitzt. Statische Klassen und statische Mitglieder erfordern eine Sonderbehandlung in der PowerShell, die an gegebener Stelle erläutert werden wird.

Vererbung

.NET unterstützt (Einfach-)Vererbung zwischen Klassen, d.h., eine Klasse kann Mitglieder von maximal einer anderen Klasse übernehmen. Als Nutzer der PowerShell haben Sie zunächst wenig mit Vererbung zu tun, in der Dokumentation werden Sie aber auf Hinweise auf Vererbung stoßen.

Schnittstellen

.NET erlaubt die explizite Definition von Schnittstellen zur Entkopplung der Beschreibung der Mitglieder einer Klasse von der Implementierung dieser Mitglieder. Schnittstellen sind für den Nutzer der PowerShell nicht relevant. Wichtig ist nur, dass Sie wissen, dass man Schnittstellen nicht mit dem PowerShell-Commandlet `New-Object` instanziieren kann. Schnittstellen erkennt man an dem vorangestellten großen „I“, z.B. IDataAdapter.

C Commandlet-Referenz

In diesem Anhang dieses Buchs finden Sie eine Liste aller Commandlets aus folgenden Bibliotheken:

- Standardbibliotheken der PowerShell Version 2.0
- PowerShell-Module Windows Server 2008 R2 bzw. Windows 7 (mit RSAT)
- PowerShell Community Extensions Version 1.2
- PowerShellPack-Pack Stand 16.10.2009
- *www.IT-Visions.de* PowerShell Extensions Version 2.2
- PowerGUI Version 1.9



Zu der Liste beachten Sie bitte folgenden Hinweis:

Microsoft hat von PowerShell 1.0 zu PowerShell 2.0 die Implementierung einiger Befehle geändert. Clear-Host war in PowerShell 1.0 eine Funktion und ist in 2.0 ein Commandlet. man und md waren in PowerShell 1.0 Funktionen und sind zu Aliase. „man" ist alias für „Help" und „md" für die Funktion „mkdir". Die Liste nennt Commandlets und Funktionen, aber keine Aliase.

Anzahl der Commandlets pro Modul	
Module Name	**Anzahl der Commandlets**
ActiveDirectory	76
ADRMS	3
AppLocker	5
BestPractices	4
BitsTransfer	8
DesktopSearch	1
FailoverClusters	69
GroupPolicy	25
ITVisions_PowerShell_Extensions	32
Microsoft.PowerShell.Core	41
Microsoft.PowerShell.Diagnostics	4

***Tabelle C.1**: Übersicht über die Anzahl der Commandlets pro Modul*

Anzahl der Commandlets pro Modul	
Module Name	**Anzahl der Commandlets**
Microsoft.PowerShell.Host	2
Microsoft.PowerShell.Management	79
Microsoft.PowerShell.Security	10
Microsoft.PowerShell.Utility	87
Microsoft.Windows.ServerManager.Migration	5
Microsoft.WSMan.Management	13
NetworkLoadBalancingClusters	35
PowerGUI	2
PowerShell Core	11
PowerShellPack	52
PSCodeGen	3
PSDiagnostics	10
PSImageTools	10
PSRss	6
PSSystemTools	13
PSUserTools	4
ServerManager	3
TaskScheduler	10
TroubleshootingPack	2
WebAdministration	72
Windows.ServerBackup	30
WPK	716

***Tabelle C.1**: Übersicht über die Anzahl der Commandlets pro Modul (Forts.)*

CO_Name	CO_DescShort	ModuleName
Add-ADComputerService-Account	Fügt einem Active Directory-Computer mindestens ein Dienstkonto hinzu	ActiveDirectory
Add-ADDomainController-PasswordReplicationPolicy	Fügt der Zulassungs- oder Verweigerungsliste einer Kennwortreplikationsrichtlinie für schreibgeschützte Domänencontroller Benutzer, Computer und Gruppen hinzu.	ActiveDirectory
Add-ADFineGrained-PasswordPolicySubject	Wendet eine abgestimmte Kennwortrichtlinie auf mindestens einen Benutzer und mindestens eine Gruppe an	ActiveDirectory
Add-ADGroupMember	Fügt einer Active Directory-Gruppe mindestens ein Mitglied hinzu	ActiveDirectory
Add-ADPrincipalGroup-Membership	Fügt mindestens einer Active Directory-Gruppe ein Mitglied hinzu.	ActiveDirectory
Add-BitsFile	Fügt einem Übertragungsauftrag des intelligenten Hintergrundübertragungsdiensts (BITS) eine oder mehrere Dateien hinzu	BitsTransfer
Add-ChildControl	Fügt ein ChildControl-Element einem Eingabefeld hinzu	WPK
Add-ClusterDisk	Verfügbarmachen eines neuen Datenträgers zur Verwendung in einem Failovercluster. Der Datenträger (LUN) muss für alle Knoten im Failovercluster verfügbar gemacht werden und sollte für keine anderen Server verfügbar gemacht werden	FailoverClusters
Add-ClusterFileServerRole	Erstellen eines Clusterdateiservers (Ressourcengruppe, die mindestens einen Datenträger enthält, auf dem Sie freigegebene Ordner für Benutzer erstellen können)	FailoverClusters
Add-ClusterGeneric-ApplicationRole	Konfigurieren hoher Verfügbarkeit für eine Anwendung, die ursprünglich nicht für die Ausführung in einem Failovercluster konzipiert war	FailoverClusters
Add-ClusterGenericScript-Role	Konfigurieren einer Anwendung, die durch ein in Windows Script Host ausgeführtes Skript gesteuert wird, innerhalb eines Failoverclusters	FailoverClusters
Add-ClusterGeneric-ServiceRole	Konfigurieren hoher Verfügbarkeit für einen Dienst, der ursprünglich nicht für die Ausführung in einem Failovercluster konzipiert war	FailoverClusters
Add-ClusterGroup	Hinzufügen einer leeren Ressourcengruppe zur Failoverclusterkonfiguration in Vorbereitung auf das Hinzufügen von Clusterressourcen zur Gruppe	FailoverClusters

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen*

CO_Name	CO_DescShort	ModuleName
Add-ClusterNode	Hinzufügen eines Knotens (Server) zu einem Failovercluster. Vor dem Hinzufügen des neuen Knotens sollten für die vorhandenen Knoten und den vorgesehenen neuen Knoten Validierungstests ausgeführt werden.	FailoverClusters
Add-ClusterPrintServer-Role	Erstellen eines Clusterdruckerservers (einer Ressourcengruppe, die einen Drucker und einen Datenträger zum Speichern von Informationen zu Druckaufträgen und Druckertreibern umfasst)	FailoverClusters
Add-ClusterResource	Hinzufügen einer Ressource zu einem geclusterten Dienst oder einer Clusteranwendung (Ressourcengruppe) in einem Failovercluster	FailoverClusters
Add-ClusterResource-Dependency	Hinzufügen einer Ressource zur Liste der Ressourcen, von denen eine bestimmte Ressource abhängig ist (unter Verwendung von AND als Konnektor), innerhalb eines Failoverclusters. Vorhandene Abhängigkeiten verbleiben in der Liste.	FailoverClusters
Add-ClusterResourceType	Hinzufügen eines Ressourcentyps zu einem Failovercluster und Angeben von Informationen wie der Dynamic Link Library (DLL), die mit diesem Ressourcentyp verwendet werden soll	FailoverClusters
Add-ClusterServerRole	Hinzufügen einer Gruppe, die lediglich einen Clientzugriffspunkt sowie Speicher umfasst, zur Failoverclusterkonfiguration	FailoverClusters
Add-ClusterSharedVolume	Verfügbarmachen eines Volumes in freigegebenen Clustervolumes in einem Failovercluster	FailoverClusters
Add-ClusterVirtual-MachineRole	Erstellen eines virtuellen Clustercomputers, d.h. eines virtuellen Computers, für den ggf. ein Failover auf einen anderen Server im Failovercluster ausgeführt werden kann	FailoverClusters
Add-CodeGenerationRule	Hinzufügen einer neuen Regel für die Code-Generierungs-Engine	WPK
Add-Computer	Fügt den lokalen Computer einer Domäne oder Arbeitsgruppe hinzu.	Microsoft.Power-Shell.Management
Add-Content	Fügt den angegebenen Elementen Inhalt hinzu, z. B. Hinzufügen von Wörtern in einer Datei	Microsoft.Power-Shell.Management

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
Add-CropFilter	Erzeugt einen Filter für abgeschnittene Bilder	PSImageTools
Add-DirectoryEntry	Anlegen eines Eintrags in einem von ADSI unterstützen Verzeichnisdienst (z.B. LDAP, Active Directory, WinNT)	ITVisions_ PowerShell_ Extensions
Add-EventHandler	Fügt einen Eventhandler einem Objekt hinzu	WPK
Add-ForeachStatement	Fügt ein Foreach-Statement einem ISE hinzu	PowerShellPack
Add-GridColumn	Fügt eine Spalte einer bestehenden Tabelle hinzu	WPK
Add-GridRow	Fügt eine Zeile einer bestehenden Tabelle hinzu	WPK
Add-History	Fügt Einträge an den Sitzungsverlauf an.	Microsoft.Power-Shell.Core
Add-IfStatement	Fügt ein if-Statement einem ISE hinzu	PowerShellPack
Add-InlineHelp	Fügt eine Vorlage für die Inline-Hilfe der aktuellen Datei hinzu	PowerShellPack
Add-IseMenu	Eine Hilfe-Funktion, um Menüs dem ISE hinzuzufügen	PowerShellPack
Add-Member	Fügt der Instanz eines Windows Power-Shell-Objekts ein benutzerdefiniertes Element hinzu.	Microsoft.Power-Shell.Utility
Add-NlbClusterNode	Fügt dem NLB-Cluster einen neuen Knoten hinzu.	NetworkLoad-BalancingClusters
Add-NlbClusterNodeDip	Fügt einem NLB-Cluster eine dedizierte IP-Adresse hinzu.	NetworkLoad-BalancingClusters
Add-NlbClusterPortRule	Fügt einem NLB-Cluster eine neue Portregel hinzu.	NetworkLoad-BalancingClusters
Add-NlbClusterVip	Fügt einem NLB-Cluster eine virtuelle IP-Adresse hinzu	NetworkLoad-BalancingClusters
Add-OU	Anlegen einer Organisationseinheit	ITVisions_ PowerShell_ Extensions
Add-OverlayFilter	Erzeugt einen Filter für überlappende Bilder	PSImageTools
Add-Parameter	Fügt ein Parameter-Attribut der aktuellen Datei im ISE hinzu	PowerShellPack
Add-PInvoke	Erzeugt C#-Code, um eine C-Funktion anzusteuern	PowerShellPack
Add-PSSnapin	Fügt der aktuellen Sitzung ein oder mehrere Windows PowerShell-Snap-Ins hinzu	Microsoft.Power-Shell.Core

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
Add-RotateFlipFilter	Erzeugt einen Filter für rotierende Bilder	PSImageTools
Add-ScaleFilter	Erzeugt einen Filter, um Bilder zu vergrößern	PSImageTools
Add-SwitchStatement	Fügt ein Switch-Statement einer aktuellen Datei hinzu	PowerShellPack
Add-TaskAction	Fügt einer Aktion eine Aufgabendefinition hinzu	TaskScheduler
Add-TaskTrigger	Fügt einen Trigger einer bestehenden Aufgabe hinzu	TaskScheduler
Add-Type	Fügt einer Windows PowerShell-Sitzung einen Microsoft .NET Framework-Typ (eine Klasse) hinzu	Microsoft.PowerShell.Utility
Add-User	Anlegen eines Benutzers	ITVisions_PowerShell_Extensions
Add-WBBackupTarget	Fügt der Sicherungsrichtlinie (WBPolicy-Objekt) das WBBackupTarget-Objekt hinzu, das Sicherungsspeicherorte angibt.	Windows.ServerBackup
Add-WBBareMetalRecovery	Fügt der Sicherungsrichtlinie (WBPolicy-Objekt) die erforderlichen Elemente hinzu, sodass Sie später eine Bare-Metal-Recovery mit Sicherungen ausführen können, die mit dieser Richtlinie erstellt wurden.	Windows.ServerBackup
Add-WBFileSpec	Fügt der Sicherungsrichtlinie (WBPolicy-Objekt) das WBFileSpec-Objekt hinzu, das die in eine Sicherung einzuschließenden oder von dieser auszuschließenden Elemente angibt.	Windows.ServerBackup
Add-WBSystemState	Fügt der Sicherungsrichtlinie (WBPolicy-Objekt) die Elemente hinzu, die erforderlich sind, um mithilfe der mit dieser Richtlinie erstellten Sicherungen Systemstatuswiederherstellungen durchführen zu können.	Windows.ServerBackup
Add-WBVolume	Fügt der Sicherungsrichtlinie (WBPolicy-Objekt) die Liste der Quellvolumes hinzu.	Windows.ServerBackup
Add-WebConfiguration	Fügt einer IIS-Konfigurationsauflistung ein Auflistungselement hinzu.	WebAdministration
Add-WebConfigurationLock	Sperrt einen IIS-Konfigurationsabschnitt oder ein IIS-Konfigurationselement.	WebAdministration
Add-WebConfigurationProperty	Fügt einem IIS-Konfigurationsabschnitt eine Eigenschaft hinzu.	WebAdministration

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
Add-WindowsFeature	Mit dem Cmdlet "Add-WindowsFeature" können Sie angegebene Rollen, Rollendienste und Features auf Computern installieren, die unter Windows Server 2008 R2 ausgeführt werden.	ServerManager
Backup-GPO	Sichert eines oder alle Gruppenrichtlinienobjekte in einer Domäne.	GroupPolicy
Backup-WebConfiguration	Erstellt eine Sicherung einer IIS-Konfiguration.	WebAdministration
Block-ClusterAccess	Verhindern, dass angegebene Benutzer auf einen Failovercluster zugreifen	FailoverClusters
cd..	Wechsel zum übergeordneten Verzeichnis	PowerShell Core
Checkpoint-Computer	Erstellt einen Systemwiederherstellungspunkt auf dem lokalen Computer.	Microsoft.PowerShell.Management
Clear-ADAccount-Expiration	Löscht das Ablaufdatum für ein Active Directory-Konto.	ActiveDirectory
Clear-ClusterDisk-Reservation	Löschen der permanenten Reservierung auf einem Datenträger in einem Failovercluster	FailoverClusters
Clear-ClusterNode	Löschen der Clusterkonfiguration aus einem Knoten, der aus einem Failovercluster entfernt wurde	FailoverClusters
Clear-Content	Löscht den Inhalt eines Elements, beispielsweise den Text in einer Datei, ohne das Element selbst zu löschen.	Microsoft.PowerShell.Management
Clear-EventLog	Löscht alle Einträge aus angegebenen Ereignisprotokollen auf lokalen oder Remotecomputern.	Microsoft.PowerShell.Management
Clear-History	Löscht Einträge aus dem Befehlsverlauf.	Microsoft.PowerShell.Core
Clear-Host	Clear-Host	PowerShell Core
Clear-Item	Löscht den Inhalt eines Elements, jedoch nicht das Element selbst.	Microsoft.PowerShell.Management
Clear-ItemProperty	Löscht den Wert einer Eigenschaft, jedoch nicht die Eigenschaft selbst.	Microsoft.PowerShell.Management
Clear-Variable	Löscht den Wert einer Variablen.	Microsoft.PowerShell.Utility
Clear-WebConfiguration	Entfernt eine Konfigurationseinstellung vom angegebenen Konfigurationsspeicherort.	WebAdministration
Clear-WebRequestTracing-Settings	Löscht die Konfiguration für die Anforderungsablaufverfolgung von der angegebenen Website.	WebAdministration

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
Close-AllOpenedFiles	Schließt automatisch alle offenen gesicherten Dateien	PowerShellPack
Close-DBConnection	Schließt eine Datenbankverbindung	ITVisions_ PowerShell_ Extensions
Compare-Object	Vergleicht zwei Sätze von Objekten.	Microsoft.PowerShell.Utility
Complete-BitsTransfer	Schließt einen Übertragungsauftrag des intelligenten Hintergrundübertragungsdienst (BITS) ab.	BitsTransfer
Complete-Transaction	Führt einen Commit für die aktive Transaktion aus.	Microsoft.PowerShell.Management
Connect-ToTaskScheduler	Verbindet zum Steuerprogramm auf einem Computer	TaskScheduler
Connect-WSMan	Stellt auf einem Remotecomputer eine Verbindung mit dem WinRM-Dienst her.	Microsoft.WSMan.Management
ConvertFrom-Csv	Konvertiert Objekteigenschaften im CSV-Format (Comma-Separated Value) in CSV-Versionen der ursprünglichen Objekte.	Microsoft.PowerShell.Utility
ConvertFrom-SecureString	Konvertiert eine sichere Zeichenfolge in eine verschlüsselte Standardzeichenfolge.	Microsoft.PowerShell.Security
ConvertFrom-StringData	Konvertiert eine Zeichenfolge, die mindestens ein Schlüssel-Wert-Paar enthält, in eine Hashtabelle.	Microsoft.PowerShell.Utility
ConvertFrom-TypeToScriptCmdlet	Konvertiert .NET-Typen in Windows PowerShell Script Cmdlets entsprechend einer Anzahl von Regeln, die mit der Add-CodeGeneration-Regel hinzugefügt wurden.	WPK
Convert-Path	Konvertiert einen Pfad aus einem Windows PowerShell-Pfad in einen Windows PowerShell-Anbieterpfad.	Microsoft.PowerShell.Management
ConvertTo-Bitmap	Konvertiert ein Bild in eine Bitmap-Datei (.bmp)	PSImageTools
ConvertTo-Csv	Konvertiert Microsoft .NET Framework-Objekte in eine Reihe von CSV-Zeichenfolgen (Comma-Separated Value).	Microsoft.PowerShell.Utility
ConvertTo-DataTemplate	Konvertiert ein UIElement in eine Datenvorlage	WPK
ConvertTo-GridLength	Wandelt $length in Windows.GridLength-Objekte um.	WPK

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
ConvertTo-Html	Konvertiert Microsoft .NET Framework-Objekte in HTML-Inhalt, der in einem Webbrowser angezeigt werden kann.	Microsoft.PowerShell.Utility
ConvertTo-Jpeg	Konvertiert ein Bild in eine JPEG-Datei.	PSImageTools
ConvertTo-ParameterMetaData	Erzeugt aus Reflection-Informationen Metadaten für einen Parameter	WPK
ConvertTo-SecureString	Konvertiert verschlüsselte Standardzeichenfolgen in sichere Zeichenfolgen. Außerdem kann Klartext mit diesem Befehl in sichere Zeichenfolgen konvertiert werden. Wird mit ConvertFrom-SecureString und Read-Host verwendet.	Microsoft.PowerShell.Security
ConvertTo-ShortcutKeyTable	Konvertiert ein hierarchisches Menü in eine Shortcut-Tabelle.	PowerShellPack
ConvertTo-WebApplication	Konvertiert ein virtuelles IIS-Verzeichnis in eine IIS-Webanwendung.	WebAdministration
ConvertTo-Xaml	Typkonvertierung einer Zeichenkette nach XAML	WPK
ConvertTo-Xml	Erstellt eine XML-Darstellung eines Objekts.	Microsoft.PowerShell.Utility
Copy-Colored	Kopiert den aktuellen Text mit farbiger Kennzeichnung in die aktuelle Datei.	PowerShellPack
Copy-ColoredHTML	Kopiert den aktuellen Text in die aktuelle Datei als farbiges HTML..	PowerShellPack
Copy-DependencyProperty	Kopiert Abhängigkeitseigenschaften von einem Objekt zu einem anderen.	WPK
Copy-GPO	Kopiert ein Gruppenrichtlinienobjekt.	GroupPolicy
Copy-ImageIntoOrganizedFolder	Kopiert Bilder in einen Ordner, dessen Abhängigkeiten vorher bestimmt wurden.	PSImageTools
Copy-Item	Kopiert ein Element innerhalb eines Namespaces von einem Speicherort an einen anderen Speicherort.	Microsoft.PowerShell.Management
Copy-ItemProperty	Kopiert eine Eigenschaft und einen Wert von einem bestimmten Speicherort an einen anderen Speicherort.	Microsoft.PowerShell.Management
Copy-ToZip	Komprimiert Dateien in einer zip-Datei.	PowerShellPack
Debug-Process	Debuggt einen oder mehr Prozesse, die auf dem lokalen Computer ausgeführt werden.	Microsoft.PowerShell.Management
Disable-ADAccount	Deaktiviert ein Active Directory-Konto.	ActiveDirectory
Disable-ADOptionalFeature	Deaktiviert ein optionales Active Directory-Feature.	ActiveDirectory

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
Disable-ComputerRestore	Deaktiviert das Systemwiederherstellungsfeature auf dem angegebenen Dateisystemlaufwerk.	Microsoft.PowerShell.Management
Disable-NlbClusterPortRule	Deaktiviert eine Portregel auf einem NLB-Cluster oder einem bestimmten Host im Cluster.	NetworkLoadBalancingClusters
Disable-PSBreakpoint	Deaktiviert die Haltepunkte in der aktuellen Konsole.	Microsoft.PowerShell.Utility
Disable-PSRemoting	Deaktiviert PowerShell-Fernzugriff auf das System.	PowerShell Core
Disable-PSSessionConfiguration	Verweigert den Zugriff auf die Sitzungskonfigurationen auf dem lokalen Computer.	Microsoft.PowerShell.Core
Disable-PSTrace	Deaktiviert PowerShell-Tracing	PSDiagnostics
Disable-PSWSManCombinedTrace	Deaktiviert WS Management-Tracing	PSDiagnostics
Disable-WebGlobalModule	Deaktiviert ein IIS-Modul.	WebAdministration
Disable-WebRequestTracing	Deaktiviert die Anforderungsablaufverfolgung.	WebAdministration
Disable-WSManCredSSP	Deaktiviert die CredSSP (Credential Security Service Provider)-Authentifizierung auf einem Clientcomputer.	Microsoft.WSMan.Management
Disable-WSManTrace	Deaktiviert WS Management-Tracing	PSDiagnostics
Disconnect-WSMan	Trennt die Verbindung des Clients mit dem WinRM-Dienst auf einem Remotecomputer.	Microsoft.WSMan.Management
Enable-ADAccount	Aktiviert ein Active Directory-Konto.	ActiveDirectory
Enable-ADOptionalFeature	Aktiviert ein optionales Active Directory-Feature.	ActiveDirectory
Enable-ComputerRestore	Aktiviert das Systemwiederherstellungsfeature auf dem angegebenen Dateisystemlaufwerk.	Microsoft.PowerShell.Management
Enable-MultiTouch	Aktiviert Multitouch-Ereignisse in einem Fenster	WPK
Enable-NlbClusterPortRule	Aktiviert eine Portregel auf einem NLB-Cluster oder einem bestimmten Knoten im Cluster.	NetworkLoadBalancingClusters
Enable-PSBreakpoint	Aktiviert die Haltepunkte in der aktuellen Konsole.	Microsoft.PowerShell.Utility
Enable-PSRemoting	Konfiguriert den Computer zum Empfangen von Remotebefehlen.	Microsoft.PowerShell.Core

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
Enable-PSSession-Configuration	Aktiviert die Sitzungskonfigurationen auf dem lokalen Computer.	Microsoft.Power-Shell.Core
Enable-PSTrace	Aktiviert PowerShell-Tracing	PSDiagnostics
Enable-PSWSMan-CombinedTrace	Aktiviert WS Management-Tracing	PSDiagnostics
Enable-WebGlobalModule	Aktiviert ein IIS-Modul.	WebAdministration
Enable-WebRequest-Tracing	Aktiviert das Feature für die Anforderungs-ablaufverfolgung.	WebAdministration
Enable-WSManCredSSP	Aktiviert die CredSSP (Credential Security Service Provider)-Authentifizierung auf einem Clientcomputer.	Microsoft.WSMan. Management
Enable-WSManTrace	Aktiviert WS Management-Tracing	PSDiagnostics
Enter-PSSession	Startet eine interaktive Sitzung mit einem Remotecomputer.	Microsoft.Power-Shell.Core
Exit-PSSession	Beendet eine interaktive Sitzung mit einem Remotecomputer.	Microsoft.Power-Shell.Core
Export-Alias	Exportiert Informationen über gegenwärtig definierte Aliase in eine Datei.	Microsoft.Power-Shell.Utility
Export-Application	Exportiert ein WPK-Skript in ein ausführ-bares Skript	WPK
Export-Clixml	Erstellt eine XML-Darstellung von Objekten und speichert sie in einer Datei.	Microsoft.Power-Shell.Utility
Export-Console	Exportiert die Namen von Snap-Ins in der aktuellen Sitzung in eine Konsolendatei.	Microsoft.Power-Shell.Core
Export-Counter	Das Cmdlet "Export-Counter" akzeptiert PerformanceCounterSampleSet-Objekte und exportiert diese als Leistungsindikator-Protokolldateien.	Microsoft.Power-Shell.Diagnostics
Export-Csv	Konvertiert Microsoft .NET Framework-Objekte in eine Reihe von CSV-Zeichenfol-gen (Comma-Separated Value) und spei-chert die Zeichenfolgen in einer CSV-Datei.	Microsoft.Power-Shell.Utility
Export-FormatData	Speichert Formatierungsdaten der aktuel-len Sitzung in einer Formatierungsdatei.	Microsoft.Power-Shell.Utility
Export-FormatView	Exports a View Definition for a particular type	PowerShellPack
Export-ModuleMember	Gibt die Modulelemente an, die exportiert werden.	Microsoft.Power-Shell.Core
Export-PSSession	Importiert Befehle aus einer anderen Sit-zung und speichert sie in einem Windows PowerShell-Modul.	Microsoft.Power-Shell.Utility

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
Export-SmigServerSetting	Exportiert ausgewählte Windows-Features und Betriebssystemeinstellungen vom lokalen Computer und speichert sie in einem Migrationsspeicher.	Microsoft.Windows.ServerManager.Migration
ForEach-Object	Führt einen Vorgang für die einzelnen Eingabeobjekte einer Gruppe von Eingabeobjekten aus.	Microsoft.PowerShell.Core
Format-Custom	Formatiert die Ausgabe mithilfe einer benutzerdefinierten Ansicht.	Microsoft.PowerShell.Utility
Format-List	Formatiert die Ausgabe als Liste von Eigenschaften, in der jede Eigenschaft in einer neuen Zeile angezeigt wird.	Microsoft.PowerShell.Utility
Format-Table	Formatiert die Ausgabe als Tabelle.	Microsoft.PowerShell.Utility
Format-Wide	Formatiert Objekte als breite Tabelle, in der nur eine Eigenschaft jedes Objekts angezeigt wird.	Microsoft.PowerShell.Utility
Get-Acl	Ruft die Sicherheitsbeschreibung für eine Ressource ab, z. B. für eine Datei oder einen Registrierungsschlüssel.	Microsoft.PowerShell.Security
Get-ADAccount-AuthorizationGroup	Ruft die Tokengruppeninformationen für Konten ab.	ActiveDirectory
Get-ADAccountResultant-PasswordReplicationPolicy	Ruft die resultierende Kennwortreplikationsrichtlinie für ein Active Directory-Konto ab.	ActiveDirectory
Get-ADComputer	Ruft mindestens einen Active Directory-Computer ab.	ActiveDirectory
Get-ADComputerService-Account	Ruft die von einem Computer gehosteten Dienstkonten ab.	ActiveDirectory
Get-ADDefaultDomain-PasswordPolicy	Ruft die Standardkennwortrichtlinie für eine Active Directory-Domäne ab.	ActiveDirectory
Get-ADDomain	Ruft eine Active Directory-Domäne ab.	ActiveDirectory
Get-ADDomainController	Ruft auf der Grundlage auffindbarer Dienstkriterien, Suchparameter oder angegebener Domänencontroller-ID, z. B. den NetBIOS-Namen, mindestens einen Active Directory-Domänencontroller ab.	ActiveDirectory
Get-ADDomainController-PasswordReplicationPolicy	Ruft die Mitglieder der Zulassungs- oder Verweigerungsliste einer Kennwortreplikationsrichtlinie für einen schreibgeschützten Domänencontroller ab.	ActiveDirectory

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
Get-ADDomainController-PasswordReplication-PolicyUsage	Ruft die Active Directory-Konten ab, die von einem schreibgeschützten Domänencontroller authentifiziert werden oder die in der offengelegten Liste des Domänencontrollers aufgeführt werden.	ActiveDirectory
Get-ADFineGrainedPasswordPolicy	Ruft mindestens eine abgestimmte Active Directory-Kennwortrichtlinie ab.	ActiveDirectory
Get-ADFineGrainedPasswordPolicySubject	Ruft die Benutzer und die Gruppen ab, auf die eine abgestimmte Kennwortrichtlinie angewendet wird.	ActiveDirectory
Get-ADForest	Ruft eine Active Directory-Gesamtstruktur ab.	ActiveDirectory
Get-ADGroup	Ruft mindestens eine Active Directory-Gruppe ab.	ActiveDirectory
Get-ADGroupMember	Ruft die Mitglieder einer Active Directory-Gruppe ab.	ActiveDirectory
Get-ADObject	Ruft mindestens ein Active Directory-Objekt ab.	ActiveDirectory
Get-ADOptionalFeature	Ruft mindestens ein optionales Active Directory-Feature ab.	ActiveDirectory
Get-ADOrganizationalUnit	Ruft mindestens eine Active Directory-Organisationseinheit ab.	ActiveDirectory
Get-ADPrincipalGroup-Membership	Ruft die Active Directory-Gruppen ab, die einen angegebenen Benutzer, Computer, eine angegebene Gruppe oder ein angegebenes Dienstkonto enthalten.	ActiveDirectory
Get-ADRootDSE	Ruft den Stamm einer Verzeichnisserver-Informationsstruktur ab.	ActiveDirectory
Get-ADServiceAccount	Ruft mindestens ein Active Directory-Dienstkonto ab.	ActiveDirectory
Get-ADUser	Ruft mindestens einen Active Directory-Benutzer ab.	ActiveDirectory
Get-ADUserResultant-PasswordPolicy	Ruft die resultierende Kennwortrichtlinie für einen Benutzer ab.	ActiveDirectory
Get-Alias	Ruft die Aliase für die aktuelle Sitzung ab.	Microsoft.Power-Shell.Utility
Get-ApplicationCommand	Holt das ApplicationCommand	WPK
Get-AppLockerFile-Information	Ruft die AppLocker-Dateiinformationen aus einer Liste von Dateien oder einem Ereignisprotokoll ab.	AppLocker
Get-AppLockerPolicy	Ruft die lokale, die aktuell gültige oder die AppLocker-Richtlinie der Domäne ab.	AppLocker

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
Get-Article	Ruft Artikel von einem RSS Feed ab.	PSRss
Get-Authenticode-Signature	Ruft Informationen über die Authenticode-Signatur in einer Datei ab.	Microsoft.Power-Shell.Security
Get-BIOS	Liefert BIOS-Informationen	ITVisions_ PowerShell_ Extensions
Get-BitsTransfer	Ruft das zugeordnete BitsJob-Objekt eines vorhandenen Übertragungsauftrags des intelligenten Hintergrundübertragungs-diensts (BITS) ab.	BitsTransfer
Get-BootStatus	Ruft Informationen ab, ob das System normal startet.	PSSystemTools
Get-BpaModel	Mit dem Cmdlet "Get-BPAModel" können Sie die Liste der auf einem Computer instal-lierten Rollen, die von Best Practices Analy-zer (BPA) unterstützt werden, abrufen und anzeigen.	BestPractices
Get-BpaResult	Mit dem Cmdlet "Get-BPAResult" können Sie die Ergebnisse des aktuellen Best Practices Analyzer (BPA)-Scans für ein bestimmtes Modell abrufen und anzeigen.	BestPractices
Get-CDRomdrive	Liefert die verfügbaren CD-ROM-Laufwerke	ITVisions_ PowerShell_ Extensions
Get-ChildControl	Liefert ein Kind-UIElement aus dem visuel-len Baum	WPK
Get-ChildItem	Ruft die Elemente und untergeordneten Elemente an einem oder mehreren ange-gebenen Speicherorten ab.	Microsoft.Power-Shell.Management
Get-Cluster	Abrufen von Informationen über mindes-tens einen Failovercluster in einer bestimm-ten Domäne	FailoverClusters
Get-ClusterAccess	Abrufen von Informationen zu Berechti-gungen, die den Zugriff auf einen Failover-cluster steuern	FailoverClusters
Get-ClusterAvailableDisk	Abrufen von Informationen über die Daten-träger, die Failover-Clusterunterstützung bieten und für alle Knoten sichtbar sind, jedoch noch nicht der Clusterdatenträger-gruppe angehören	FailoverClusters

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
Get-ClusterGroup	Abrufen von Informationen über eine(n) oder mehrere geclusterte Dienste oder Clusteranwendungen (Ressourcengruppen) in einem Failovercluster	FailoverClusters
Get-ClusterLog	Erstellen einer Protokolldatei für alle (oder einen bestimmten) Knoten in einem Failovercluster	FailoverClusters
Get-ClusterNetwork	Abrufen von Informationen über mindestens ein Netzwerk in einem Failovercluster	FailoverClusters
Get-Cluster-NetworkInterface	Abrufen von Informationen über mindestens eine Netzwerkkarte in einem Failovercluster	FailoverClusters
Get-ClusterNode	Abrufen von Informationen über mindestens einen Knoten (Server) in einem Failovercluster	FailoverClusters
Get-ClusterOwnerNode	Abrufen von Informationen darüber, welche Knoten eine bestimmte Ressource in einem Failovercluster besitzen können. Abrufen von Informationen über die bevorzugte Reihenfolge unter Besitzerknoten für einen geclusterten Dienst bzw. eine Clusteranwendung (Ressourcengruppe)	FailoverClusters
Get-ClusterParameter	Abrufen ausführlicher Informationen über ein Objekt in einem Failovercluster, z. B. eine Clusterressource Mit diesem Cmdlet können Sie private Eigenschaften für ein Clusterobjekt verwalten. Wenn Sie allgemeine Eigenschaften für ein Clusterobjekt abrufen und festlegen möchten, rufen Sie mit dem entsprechenden Get-Cluster*-Cmdlet das betreffende Clusterobjekt ab, und legen Sie dann die betreffende Eigenschaft für das Clusterobjekt direkt fest.	FailoverClusters
Get-ClusterQuorum	Abrufen von Informationen über die Quorumkonfiguration eines Failoverclusters	FailoverClusters
Get-ClusterResource	Abrufen von Informationen über mindestens eine Ressource in einem Failovercluster	FailoverClusters
Get-ClusterResource-Dependency	Abrufen von Informationen über die Abhängigkeiten, die zwischen Clusterressourcen in einem Failovercluster konfiguriert wurden	FailoverClusters

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
Get-ClusterResource-DependencyReport	Generieren eines Berichts, in dem die Abhängigkeiten von Ressourcen in einem Failovercluster aufgeführt sind	FailoverClusters
Get-ClusterResourceType	Abrufen von Informationen über mindestens einen Ressourcentyp in einem Failovercluster	FailoverClusters
Get-ClusterSharedVolume	Abrufen von Informationen zu freigegebenen Clustervolumes in einem Failovercluster	FailoverClusters
Get-Command	Ruft grundlegende Informationen zu Cmdlets und anderen Elementen von Windows PowerShell-Befehlen ab.	Microsoft.PowerShell.Core
Get-CommandWithParameterType	Ruft Kommandos ab, die einen bestimmten Parameter-Typ akzeptieren.	PowerShellPack
Get-ComponentCommand	Ruft WPF-Kommandos für ComponentCommand ab.	WPK
Get-ComputerInfo	Liefert Informationen über den Computer	ITVisions_PowerShell_Extensions
Get-Computername	Liefert den Namen des Computers	ITVisions_PowerShell_Extensions
Get-ComputerRestorePoint	Ruft die Wiederherstellungspunkte auf dem lokalen Computer ab.	Microsoft.PowerShell.Management
Get-Content	Ruft den Inhalt des Elements am angegebenen Speicherort ab.	Microsoft.PowerShell.Management
Get-Counter	Ruft Leistungsindikatordaten von lokalen Computern und Remotecomputern ab.	Microsoft.PowerShell.Diagnostics
Get-Credential	Ruft ein Objekt mit Anmeldeinformationen entsprechend einem Benutzernamen und Kennwort ab.	Microsoft.PowerShell.Security
Get-Culture	Ruft die aktuelle Kultureinstellung des Betriebssystems ab.	Microsoft.PowerShell.Utility
Get-CurrentOpenedFileToken	Ruft die PowerShell Parser Tokens für die aktuelle Datei ab.	PowerShellPack
Get-CurrentToken	Holt das aktuelle Token innerhalb einer Datei.	PowerShellPack
Get-CurrentUser	Liefert den aktuellen angemeldetn Benutzer.	PSUserTools
Get-Date	Ruft das aktuelle Datum und die aktuelle Uhrzeit ab.	Microsoft.PowerShell.Utility

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
Get-DbConnection	Erzeugt eine Verbindung zu einer Datenbank	ITVisions_ PowerShell_ Extensions
Get-DbRow	Liefert eine Zeile au seiner Datenbanktabelle	ITVisions_ PowerShell_ Extensions
Get-DbTable	Liefert eine komplette Datenbanktabelle	ITVisions_ PowerShell_ Extensions
Get-DependencyProperty	Ruft die Abhängigkeitseigenschaften ab, die einem Steuerelement zugeordnet worden sind.	WPK
Get-DirectoryChildren	Liefert untergeordnete Einträge in einem Ast im Verzeichnisbaum.	ITVisions_ PowerShell_ Extensions
Get-DirectoryEntry	Liefert einen Verzeichnisdiensteintrag in einem von ADSI unterstützen Verzeichnisdienst (z.B. LDAP, Active Directory, WinNT)	ITVisions_ PowerShell_ Extensions
Get-DirectoryValue	Liefert einen einzelnen Wert aus einem Verzeichnisdiensteintrag in einem von ADSI unterstützen Verzeichnisdienst (z.B. LDAP, Active Directory, WinNT)	ITVisions_ PowerShell_ Extensions
Get-Disk	Liefert die verfügbaren Laufwerke	ITVisions_ PowerShell_ Extensions
Get-DisplaySetting	Ruft die Bildschirmbreite und -höhe ab.	PSSystemTools
Get-DuplicateFile	Findet Dateien mit gleichem Inhalt	PowerShellPack
Get-Event	Ruft die Ereignisse in der Ereigniswarteschlange ab.	Microsoft.PowerShell.Utility
Get-EventLog	Ruft die Ereignisse in einem Ereignisprotokoll oder eine Liste der Ereignisprotokolle auf dem lokalen oder Remotecomputern ab.	Microsoft.PowerShell.Management
Get-EventSubscriber	Ruft die Ereignisabonnenten in der aktuellen Sitzung ab.	Microsoft.PowerShell.Utility
Get-Everyone	Rufte alle Anwender ab.	PSUserTools
Get-ExecutionPolicy	Ruft die Ausführungsrichtlinien für die aktuelle Sitzung ab.	Microsoft.PowerShell.Security
Get-Feed	Ruft RSS-Feed ab.	PSRss
Get-Font	Ruft die aktuell im System geladenen Fonts ab.	PSSystemTools
Get-FormatData	Ruft die Formatierungsdaten in der aktuellen Sitzung ab.	Microsoft.PowerShell.Utility

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
Get-FreeDiskSpace	Liefert den freien Festplattenplatz.	PowerShellPack
Get-FunctionFromFile	Entnimmt Funktionsdeklarationen aus einer Datei.	PowerShellPack
Get-GPInheritance	Ruft Informationen zur Gruppenrichtlinienvererbung für eine angegebene Domäne oder Organisationseinheit ab.	GroupPolicy
Get-GPO	Ruft eines oder alle Gruppenrichtlinienobjekte in einer Domäne ab.	GroupPolicy
Get-GPOReport	Generiert für ein angegebenes Gruppenrichtlinienobjekt oder für alle Gruppenrichtlinienobjekte in einer Domäne einen Bericht im XML- oder HTML-Format.	GroupPolicy
Get-GPPermissions	Ruft die Berechtigungsstufe für einen oder mehrere Sicherheitsprinzipale in einem angegebenen Gruppenrichtlinienobjekt ab.	GroupPolicy
Get-GPPrefRegistryValue	Ruft eines oder mehrere Registrierungseinstellungselemente unter der Computerkonfiguration oder der Benutzerkonfiguration in einem Gruppenrichtlinienobjekt ab.	GroupPolicy
Get-GPRegistryValue	Ruft eine oder mehrere registrierungsbasierte Richtlinieneinstellungen aus der Computerkonfiguration oder der Benutzerkonfiguration in einem Gruppenrichtlinienobjekt ab.	GroupPolicy
Get-GPResultantSetOfPolicy	Gibt die Richtlinienergebnissatz-Informationen für einen Benutzer, einen Computer oder beides in einer Datei aus.	GroupPolicy
Get-GPStarterGPO	Ruft ein Starter-Gruppenrichtlinienobjekt oder alle Starter-Gruppenrichtlinienobjekte in einer Domäne ab.	GroupPolicy
Get-HashtableAsObject	Wandelt ein Hashtable in ein PowerShell-Objekt um.	WPK
Get-Help	Zeigt Informationen zu Windows PowerShell-Befehlen und -Konzepten an.	Microsoft.PowerShell.Core
Get-History	Ruft eine Liste der Befehle ab, die während der aktuellen Sitzung eingegeben wurden.	Microsoft.PowerShell.Core
Get-Host	Ruft ein Objekt ab, das das aktuelle Hostprogramm darstellt. Und zeigt standardmäßig Informationen über die Windows PowerShell-Version und die Regionaleinstellungen an.	Microsoft.PowerShell.Utility

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
Get-HotFix	Ruft die Hotfixes ab, die auf dem lokalen und auf Remotecomputern angewendet wurden.	Microsoft.PowerShell.Management
Get-Image	Ruft Bildinformationen ab.	PSImageTools
Get-ImageProperty	Ruft die erweiterten Eigenschaften von Bildern ab.	PSImageTools
Get-Item	Ruft das Element am angegebenen Speicherort ab.	Microsoft.PowerShell.Management
Get-ItemProperty	Ruft die Eigenschaften eines angegebenen Elements ab.	Microsoft.PowerShell.Management
Get-ITVisions	Ermöglicht eine Abfrage, ob eine neue Version der ITVisions_PowerShell_Extensions existiert.	ITVisions_PowerShell_Extensions
Get-Job	Ruft Windows PowerShell-Hintergrundaufträge ab, die in der aktuellen Sitzung ausgeführt werden.	Microsoft.PowerShell.Core
Get-Keyboard	Liefert die verfügbaren Tastaturen	ITVisions_PowerShell_Extensions
Get-Location	Ruft Informationen über den aktuellen Arbeitsspeicherort ab.	Microsoft.PowerShell.Management
Get-LogicalDiskInventory	Listet die Laufwerke aus.	PSSystemTools
Get-MediaCommand	Liefert ein WPF-MediaCommand.	WPK
Get-Member	Ruft die Eigenschaften und Methoden von Objekten ab.	Microsoft.PowerShell.Utility
Get-MemoryDevice	Liefert das verfügbare RAM.	ITVisions_PowerShell_Extensions
Get-Metadata	Liefert Informationen über den Inhalt der Pipeline.	ITVisions_PowerShell_Extensions
Get-Module	Ruft die Module ab, die in die aktuelle Sitzung importiert wurden oder importiert werden können.	Microsoft.PowerShell.Core
Get-MultiTouchMaximum	Ruft die Anzahl von Fingern ab, die mit einem Multitouch-Gerät verwendet werden können.	PSSystemTools
Get-NavigationCommand	Liefert ein WPF NavigationCommand.	WPK
Get-Networkadapter	Liefert die verfügbaren Netzwerkkarten.	ITVisions_PowerShell_Extensions

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
Get-NlbCluster	Ruft Informationen über das NLB-Clusterobjekt ab, das vom Aufrufer abgefragt wurde.	NetworkLoad-BalancingClusters
Get-NlbClusterDriverInfo	Ruft Informationen über den NLB-Treiber auf dem lokalen Computer ab.	NetworkLoad-BalancingClusters
Get-NlbClusterNode	Ruft Informationen über ein Knotenobjekt ab, das vom Aufrufer abgefragt wird. Ruft Informationen über das NLB-Clusterobjekt ab, das vom Aufrufer abgefragt wurde.	NetworkLoad-BalancingClusters
Get-NlbClusterNodeDip	Ruft die dedizierte IP-Adresse ab, die vom Aufrufer abgefragt wird.	NetworkLoad-BalancingClusters
Get-NlbClusterNode-NetworkInterface	Ruft Informationen über Schnittstellen auf einem Host ab, einschließlich Informationen über den NLB-Treiber.	NetworkLoad-BalancingClusters
Get-NlbClusterPortRule	Ruft die vom Aufrufer abgefragten Portregelobjekte ab.	NetworkLoad-BalancingClusters
Get-NlbClusterVip	Ruft virtuelle IP-Adressen ab, die vom Aufrufer abgefragt werden.	NetworkLoad-BalancingClusters
Get-OSVersion	Ruft die Betriebssystemversion ab	PSSystemTools
Get-PfxCertificate	Mit diesem Befehl werden Informationen über PFX-Zertifikatsdateien auf dem Computer abgerufen.	Microsoft.Power-Shell.Security
Get-PipelineInfo	Liefert Informationen über den Inhalt der Pipeline.	ITVisions_ PowerShell_ Extensions
Get-PointingDevice	Liefert die verfügbaren Eingabegeräte.	ITVisions_ PowerShell_ Extensions
Get-PowerShellData-Source	Liefert eine PowerShell-Datenquelle.	WPK
Get-Process	Ruft die Prozesse ab, die auf dem lokalen Computer oder einem Remotecomputer ausgeführt werden.	Microsoft.Power-Shell.Management
Get-Processor		PSSystemTools
Get-ProgID	Liefert alle COM-Programmatic Identifier.	PowerShellPack
Get-PSBreakpoint	Ruft die Haltepunkte ab, die in der aktuellen Sitzung festgelegt sind.	Microsoft.Power-Shell.Utility
Get-PSCallStack	Zeigt die aktuelle Aufrufliste an.	Microsoft.Power-Shell.Utility
Get-PSDrive	Ruft die Windows PowerShell-Laufwerke in der aktuellen Sitzung ab.	Microsoft.Power-Shell.Management

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
Get-PSProvider	Ruft Informationen über den angegebenen Windows PowerShell-Anbieter ab.	Microsoft.PowerShell.Management
Get-PSSession	Ruft die Windows PowerShell-Sitzungen (PSSessions) in der aktuellen Sitzung ab.	Microsoft.PowerShell.Core
Get-PSSessionConfiguration	Ruft die registrierten Sitzungskonfigurationen auf dem Computer ab.	Microsoft.PowerShell.Core
Get-PSSnapin	Ruft die Windows PowerShell-Snap-Ins auf dem Computer ab.	Microsoft.PowerShell.Core
Get-Random	Ruft eine Zufallszahl ab oder wählt Objekte nach dem Zufallsprinzip aus einer Auflistung aus.	Microsoft.PowerShell.Utility
Get-Resource	Findet eine WPF-Ressource.	WPK
Get-RunningTask	Ruft die Aufgaben ab, die gerade auf dem System laufen.	TaskScheduler
Get-ScheduledTask	Liefert die Liste der geplanten Vorgänge.	TaskScheduler
Get-Service	Ruft die Dienste auf einem lokalen Computer oder einem Remotecomputer ab.	Microsoft.PowerShell.Management
Get-SHA1	Computes the SHA1 hash of the specified file.	PowerShellPack
Get-SmigServerFeature	Ruft den Satz aller Windows-Features ab, die vom lokalen Server oder aus einem Migrationsspeicher migriert werden können.	Microsoft.Windows.ServerManager.Migration
Get-SoundDevice	Liefert die verfügbaren Audioausgabegeräte.	ITVisions_PowerShell_Extensions
Get-SystemFont	Liefert die Schriftarten für Systemelemente.	PSSystemTools
Get-Tapedrive	Liefert die verfügbaren Bandlaufwerke.	ITVisions_PowerShell_Extensions
Get-TokenFromFile	Holt ein Token aus einer Datei.	PowerShellPack
Get-TraceSource	Ruft die Windows PowerShell-Komponenten ab, die für die Ablaufverfolgung instrumentiert werden.	Microsoft.PowerShell.Utility
Get-Transaction	Ruft die aktuelle (aktive) Transaktion ab.	Microsoft.PowerShell.Management
Get-TroubleshootingPack	Ruft Informationen zu einem Problembehandlungspaket ab und kann eine Antwortdatei generieren.	TroubleshootingPack
Get-Type	Liefert die Liste der aktuell geladenen Typen oder liefert Informationen über einen Typen.	PowerShellPack

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
Get-UICulture	Ruft die aktuellen Einstellungen für die Benutzeroberflächenkultur aus dem Betriebssystem ab.	Microsoft.PowerShell.Utility
Get-Unique	Gibt die eindeutigen Elemente aus einer sortierten Liste zurück.	Microsoft.PowerShell.Utility
Get-USB	Liefert die verfügbaren USB-Geräte	PSSystemTools
Get-USBController	Liefert die verfügbaren USB-Geräte.	ITVisions_PowerShell_Extensions
Get-Variable	Ruft die Variablen in der aktuellen Konsole ab.	Microsoft.PowerShell.Utility
Get-Videocontroller	Zeigt die Videokarten an.	ITVisions_PowerShell_Extensions
Get-WBBackupSet	Ruft die Liste der Sicherungen (WBBackupSet-Objekte) ab, die für einen Server erstellt und an dem von Ihnen angegebenen Speicherort gespeichert wurden.	Windows.ServerBackup
Get-WBBackupTarget	Ruft die Speicherorte zum Speichern von Sicherungen ab, die Sie als Teil der Sicherungsrichtlinie (WBPolicy-Objekt) angegeben haben.	Windows.ServerBackup
Get-WBBareMetalRecovery	Ruft den Wert ab, der angibt, ob der Sicherungsrichtlinie (WBPolicy-Objekt) die Funktion zum Ausführen von Bare-Metal-Recoverys aus Sicherungen hinzugefügt wurde.	Windows.ServerBackup
Get-WBDisk	Ruft die Liste der internen und externen Datenträger ab, die auf dem lokalen Computer online sind.	Windows.ServerBackup
Get-WBFileSpec	Ruft die Liste der WBFileSpec-Objekte ab, die der angegebenen Sicherungsrichtlinie (WBPolicy-Objekt) zugeordnet sind.	Windows.ServerBackup
Get-WBJob	Ruft den derzeit ausgeführten Sicherungsvorgang (WBJob-Objekt) ab.	Windows.ServerBackup
Get-WBPolicy	Ruft die aktuelle, für den Computer festgelegte Sicherungsrichtlinie ab.	Windows.ServerBackup
Get-WBSchedule	Ruft den aktuellen Zeitplan für Sicherungen in der Sicherungsrichtlinie (WBPolicy-Objekt) ab.	Windows.ServerBackup
Get-WBSummary	Ruft den Verlauf der ausgeführten Sicherungsvorgänge ab.	Windows.ServerBackup

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
Get-WBSystemState	Ruft einen booleschen Wert ab, der angibt, ob die Funktion zum Ausführen von Systemstatuswiederherstellungen mit den Sicherungen der Sicherungsrichtlinie (WBPolicy-Objekt) hinzugefügt wurde.	Windows.Server-Backup
Get-WBVolume	Ruft die Liste der Quellvolumes für die in der Sicherungsrichtlinie (WBPolicy-Objekt) angegebene Sicherung ab.	Windows.Server-Backup
Get-WBVssBackupOptions	Ruft eine Einstellung ab, die angibt, ob es sich bei den mit der Sicherungsrichtlinie (WBPolicy-Objekt) erstellten Sicherungen um VSS-Kopiesicherungen (Volumeschattenkopie-Dienst) oder um vollständige VSS-Sicherungen handeln muss.	Windows.Server-Backup
Get-WebAppDomain	Ruft die Anwendungsdomänen ab, in denen der angegebene IIS-Arbeitsprozess ausgeführt wird.	WebAdministration
Get-WebApplication	Ruft die Webanwendungen ab, die einer bestimmten Site zugeordnet sind oder den angegebenen Namen besitzen.	WebAdministration
Get-WebAppPoolState	Ruft den Laufzeitstatus eines IIS-Anwendungspools ab.	WebAdministration
Get-WebBinding	Ruft die Bindungen der angegebenen IIS-Site ab.	WebAdministration
Get-WebConfigFile	Ruft den Dateisystempfad der Datei "web.config" ab.	WebAdministration
Get-WebConfiguration	Ruft ein IIS-Konfigurationselement am angegebenen Pfad ab.	WebAdministration
Get-WebConfiguration-Backup	Ruft eine Liste verfügbarer IIS-Konfigurationssicherungen ab.	WebAdministration
Get-WebConfigurationLocation	Ruft den Speicherort einer angegebenen Konfigurationseinstellung ab.	WebAdministration
Get-WebConfiguration-Lock	Ruft den Sperrstatus des angegebenen IIS-Konfigurationsspeicherorts ab.	WebAdministration
Get-WebConfiguration-Property	Ruft eine IIS-Konfigurationseigenschaft am angegebenen Pfad ab.	WebAdministration
Get-WebFilePath	Ruft den physischen Pfad zum angegebenen IIS-Modulspeicherort ab.	WebAdministration
Get-WebGlobalModule	Ruft in IIS konfigurierte Module ab.	WebAdministration
Get-WebHandler	Ruft IIS-Anforderungshandler ab.	WebAdministration
Get-WebItemState	Ruft den Laufzeitstatus einer Site oder eines Anwendungspools ab.	WebAdministration

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
Get-WebManagedModule	Ruft die verwalteten Module ab, die für eine bestimmte Anwendung konfiguriert sind.	WebAdministration
Get-WebRequest	Zeigt IIS-Anforderungen an, die gerade ausgeführt werden.	WebAdministration
Get-Website	Ruft eine IIS-Website ab.	WebAdministration
Get-WebsiteState	Ruft den Status einer IIS-Website ab.	WebAdministration
Get-WebURL	Gibt eine einfache Webanforderung für den angegebenen IIS-Modulspeicherort aus.	WebAdministration
Get-WebVirtualDirectory	Ruft ein virtuelles IIS-Verzeichnis ab.	WebAdministration
Get-WindowsEdition	Zeigt die Betriebssystemversion an.	PSSystemTools
Get-WindowsFeature	Mit dem Cmdlet "Get-WindowsFeature" können Sie Informationen zu Rollen, Rollendiensten und Features abrufen, die auf Computern verfügbar sind, die unter Windows Server 2008 R2 ausgeführt werden.	ServerManager
Get-WinEvent	Ruft Ereignisse aus Ereignisprotokollen und Ereignisablaufverfolgungs-Protokolldateien auf lokalen Computern und Remotecomputern ab.	Microsoft.PowerShell.Diagnostics
Get-WmiObject	Ruft Instanzen von WMI-Klassen (Windows Management Instrumentation, Windows-Verwaltungsinstrumentation) oder Informationen über die verfügbaren Klassen ab.	Microsoft.PowerShell.Management
Get-WSManCredSSP	Ruft die Credential Security Service Provider-bezogene Konfiguration für den Client ab.	Microsoft.WSMan.Management
Get-WSManInstance	Zeigt Verwaltungsinformationen für eine von einer Ressourcen-URI angegebene Ressourceninstanz an.	Microsoft.WSMan.Management
Grant-ClusterAccess	Gewähren entweder von Vollzugriff oder schreibgeschütztem Zugriff auf einen Failovercluster	FailoverClusters
Group-Object	Gruppiert Objekte, die für die angegebenen Eigenschaften denselben Wert enthalten.	Microsoft.PowerShell.Utility
help	Zeigt Informationen zu Windows PowerShell-Befehlen und -Konzepten an.	PowerShell Core
Hide-UIElement	Verbirgt ein UI-Element	WPK
Import-Alias	Importiert eine Aliasliste aus einer Datei.	Microsoft.PowerShell.Utility

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
Import-Clixml	Importiert eine CLIXML-Datei und erstellt entsprechende Objekte in Windows PowerShell.	Microsoft.Power-Shell.Utility
Import-Counter	Importiert Leistungsindikator-Protokoll-dateien (".blg", ".csv", ".tsv") und erstellt die Objekte, die die einzelnen Leistungsindika-torstichproben im Protokoll darstellen.	Microsoft.Power-Shell.Diagnostics
Import-Csv	Konvertiert Objekteigenschaften in einer CSV-Datei (Comma-Separated Value) in CSV-Versionen der ursprünglichen Objekte.	Microsoft.Power-Shell.Utility
Import-GPO	Importiert die Gruppenrichtlinieneinstel-lungen aus einem gesicherten Gruppen-richtlinienobjekt in ein angegebenes Gruppenrichtlinienobjekt.	GroupPolicy
Import-IniFile	Importiert die Einstellungen aus einer INI-Datei	PSSystemTools
Import-LocalizedData	Importiert sprachspezifische Daten auf Grundlage der für das Betriebssystem aus-gewählten Benutzeroberflächenkultur in Skripts und Funktionen.	Microsoft.Power-Shell.Utility
Import-Module	Fügt der aktuellen Sitzung Module hinzu.	Microsoft.Power-Shell.Core
Import-PSSession	Importiert Befehle aus einer anderen Sitzung in die aktuelle Sitzung.	Microsoft.Power-Shell.Utility
Import-SmigServerSetting	Importiert ausgewählte Windows-Features und Betriebssystemeinstellungen aus einem Migrationsspeicher auf den lokalen Computer.	Microsoft.Windows. ServerManager. Migration
ImportSystemModules	Importiert die Systemmodule.	PowerShell Core
Install-ADRMS	Installiert und konfiguriert AD RMS auf diesem Computer	ADRMS
Install-ADServiceAccount	Installiert ein Active Directory-Dienstkonto auf einem Computer.	ActiveDirectory
Invoke-BpaModel	Mit dem Cmdlet "Invoke-BPAModel" können Sie einen Best Practices Analyzer (BPA)-Scan für ein bestimmtes, auf dem Computer installiertes Modell starten.	BestPractices
Invoke-Command	Führt Befehle auf lokalen Computern und auf Remotecomputern aus.	Microsoft.Power-Shell.Core
Invoke-DbCommand	Führt einen SQL-Befehl auf einer Datenbank aus	ITVisions_ PowerShell_ Extensions

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
Invoke-Expression	Führt Befehle oder Ausdrücke auf dem lokalen Computer aus.	Microsoft.PowerShell.Utility
Invoke-History	Führt Befehle aus dem Sitzungsverlauf aus.	Microsoft.PowerShell.Core
Invoke-Item	Führt die Standardaktion für das angegebene Element aus.	Microsoft.PowerShell.Management
Invoke-ScalarDbCommand	Führt einen SQL-Befehl auf einer Datenbank aus, der einen einzelnen Wert zurückliefert	ITVisions_PowerShell_Extensions
Invoke-TroubleshootingPack	Führt ein Problembehandlungspaket zum Beheben eines Problems im interaktiven oder im unbeaufsichtigten Modus aus und generiert anschließend einen Ergebnisbericht.	TroubleshootingPack
Invoke-WmiMethod	Ruft Methoden der Windows-Verwaltungsinstrumentation (Windows Management Instrumentation, WMI) auf.	Microsoft.PowerShell.Management
Invoke-WSManAction	Ruft eine Aktion für das vom Ressourcen-URI und in den Selektoren angegebene Objekt auf.	Microsoft.WSMan.Management
Join-Path	Kombiniert einen Pfad und einen untergeordneten Pfad zu einem Pfad. Der Anbieter stellt die Pfadtrennzeichen bereit.	Microsoft.PowerShell.Management
Limit-EventLog	Legt die Ereignisprotokolleigenschaften fest, die die Größe des Ereignisprotokolls und das Alter seiner Einträge einschränken.	Microsoft.PowerShell.Management
Measure-Command	Misst die Zeit, die zum Ausführen von Skriptblöcken und Cmdlets benötigt wird.	Microsoft.PowerShell.Utility
Measure-Object	Berechnet die numerischen Eigenschaften von Objekten sowie die Zeichen, Wörter und Zeilen in Zeichenfolgenobjekten, z.B. Textdateien.	Microsoft.PowerShell.Utility
mkdir	Erstellt ein neues Element.	PowerShell Core
more	Ermöglicht die seitenweise Ausgabe.	PowerShell Core
Mount-SpecialFolder	Erzeugt PowerShell-Laufwerke für alle Sonderordner.	PowerShellPack
Move-ADDirectoryServer	Verschiebt einen Verzeichnisserver in Active Directory an einen neuen Standort.	ActiveDirectory
Move-ADDirectoryServerOperationMasterRole	Verschiebt Betriebsmasterrollen auf einen Active Directory-Verzeichnisserver.	ActiveDirectory

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
Move-ADObject	Verschiebt ein Active Directory-Objekt oder einen Container mit Active Directory-Objekten in einen anderen Container oder eine andere Domäne.	ActiveDirectory
Move-ClusterGroup	Verschieben eines geclusterten Diensts oder einer Clusteranwendung (Ressourcengruppe) zwischen Knoten in einem Failovercluster	FailoverClusters
Move-ClusterResource	Verschieben einer Clusterressource von einem geclusterten Dienst oder einer Clusteranwendung innerhalb eines Failoverclusters in einen anderen geclusterten Dienst oder eine andere Clusteranwendung	FailoverClusters
Move-ClusterShared-Volume	Verschieben eines freigegebenen Clustervolumes auf einen anderen Besitzerknoten in einem Failovercluster	FailoverClusters
Move-ClusterVirtual-MachineRole	Ändern des Besitzerknotens eines virtuellen Clustercomputers in einen anderen Knoten	FailoverClusters
Move-Control	Bewegt ein Steuerelement.	WPK
Move-Item	Verschiebt ein Element von einem Speicherort an einen anderen Speicherort.	Microsoft.PowerShell.Management
Move-ItemProperty	Verschiebt eine Eigenschaft von einem Speicherort an einen anderen Speicherort.	Microsoft.PowerShell.Management
Move-ToLastGroup	Springt zum nächsten grouping token ({}) im aktuellen Skript.	PowerShellPack
Move-ToLastPowerShell-Tab	Springt zum letzten PowerShell-Tabulator	PowerShellPack
Move-ToNextGroup	Springt zum nächsten grouping token ({}) im aktuellen Skript	PowerShellPack
Move-ToNextPowerShell-Tab	Sprint zum nächsten PowerShell-Tabulator	PowerShellPack
New-AccessKeyPressed-EventArgs	Erzeugt ein neues System.Windows.Input.AccessKeyPressedEventArgs-Objekt	WPK
New-AccessText	Erzeugt ein neues System.Windows.Controls.AccessText-Objekt	WPK
New-ADComputer	Erstellt einen neuen Active Directory-Computer-Objekt.	ActiveDirectory
New-ADFineGrained-PasswordPolicy	Erstellt eine neue abgestimmte Active Directory-Kennwortrichtlinie.	ActiveDirectory
New-ADGroup	Erstellt eine Active Directory-Gruppe.	ActiveDirectory
New-ADObject	Erstellt ein Active Directory-Objekt.	ActiveDirectory

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
New-ADOrganizational-Unit	Erstellt eine neue Active Directory-Organisationseinheit.	ActiveDirectory
New-AdornedElement-Placeholder	Erzeugt ein neues System.Windows.Controls.AdornedElementPlaceholder-Objekt	WPK
New-AdornerDecorator	Erzeugt ein neues System.Windows.Documents.AdornerDecorator-Objekt	WPK
New-ADServiceAccount	Erstellt ein neues Active Directory-Dienstkonto.	ActiveDirectory
New-ADUser	Erstellt einen neuen Active Directory-Benutzer.	ActiveDirectory
New-Alias	Erstellt einen neuen Alias.	Microsoft.PowerShell.Utility
New-AlternationConverter	Erzeugt ein neues System.Windows.Controls.AlternationConverter-Objekt	WPK
New-AmbientAttribute	Erzeugt ein neues System.Windows.Markup.AmbientAttribute-Objekt	WPK
New-AmbientLight	Erzeugt ein neues System.Windows.Media.Media3D.AmbientLight-Objekt	WPK
New-Annotation	Erzeugt ein neues System.Windows.Annotations.Annotation-Objekt	WPK
New-AnnotationResource	Erzeugt ein neues System.Windows.Annotations.AnnotationResource-Objekt	WPK
New-Application	Erzeugt ein neues System.Windows.Application-Objekt	WPK
New-AppLockerPolicy	Erstellt eine neue AppLocker-Richtlinie aus einer Liste mit Dateiinformationen und anderen Optionen für die Regelerstellung.	AppLocker
New-ArcSegment	Erzeugt ein neues System.Windows.Media.ArcSegment-Objekt	WPK
New-ArrayExtension	Erzeugt ein neues System.Windows.Markup.ArrayExtension-Objekt	WPK
New-AttachedProperty-BrowsableForChildren-Attribute	Erzeugt ein neues System.Windows.AttachedPropertyBrowsableForChildren-Attribute-Objekt	WPK
New-AxisAngleRotation3D	Erzeugt ein neues System.Windows.Media.Media3D.AxisAngleRotation3D-Objekt	WPK
New-BamlLocalizable-Resource	Erzeugt ein neues System.Windows.Markup.Localizer.BamlLocalizableResource-Objekt	WPK

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
New-BamlLocalization-Dictionary	Erzeugt ein neues System.Windows.Markup.Localizer.BamlLocalization-Dictionary-Objekt	WPK
New-BeginStoryboard	Erzeugt ein neues System.Windows.Media.Animation.BeginStoryboard-Objekt	WPK
New-BevelBitmapEffect	Erzeugt ein neues System.Windows.Media.Effects.BevelBitmapEffect-Objekt	WPK
New-BezierSegment	Erzeugt ein neues System.Windows.Media.BezierSegment-Objekt	WPK
New-Binding	Erzeugt ein neues System.Windows.Data.Binding-Objekt	WPK
New-BindingGroup	Erzeugt ein neues System.Windows.Data.BindingGroup-Objekt	WPK
New-BindingNavigator	Erzeugt ein neues System.Windows.Forms.BindingNavigator-Objekt	WPK
New-BitmapEffect-Collection	Erzeugt ein neues System.Windows.Media.Effects.BitmapEffectCollection-Objekt	WPK
New-BitmapEffectGroup	Erzeugt ein neues System.Windows.Media.Effects.BitmapEffectGroup-Objekt	WPK
New-BitmapEffectInput	Erzeugt ein neues System.Windows.Media.Effects.BitmapEffectInput-Objekt	WPK
New-BitmapImage	Erzeugt ein neues System.Windows.Media.Imaging.BitmapImage-Objekt	WPK
New-BlockElement	Erzeugt ein neues System.Windows.Documents.DocumentStructures.Block-Element-Objekt	WPK
New-BlockUIContainer	Erzeugt ein neues System.Windows.Documents.BlockUIContainer-Objekt	WPK
New-BlurBitmapEffect	Erzeugt ein neues System.Windows.Media.Effects.BlurBitmapEffect-Objekt	WPK
New-BlurEffect	Erzeugt ein neues System.Windows.Media.Effects.BlurEffect-Objekt	WPK
New-BmpBitmapEncoder	Erzeugt ein neues System.Windows.Media.Imaging.BmpBitmapEncoder-Objekt	WPK
New-Bold	Erzeugt ein neues System.Windows.Documents.Bold-Objekt	WPK
New-BooleanAnimation-UsingKeyFrames	Erzeugt ein neues System.Windows.Media.Animation.BooleanAnimationUsing-KeyFrames-Objekt	WPK

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
New-BooleanKeyFrame-Collection	Erzeugt ein neues System.Windows.Media.Animation.BooleanKeyFrame-Collection-Objekt	WPK
New-BooleanToVisibility-Converter	Erzeugt ein neues System.Windows.Controls.BooleanToVisibilityConverter-Objekt	WPK
New-BoolIListConverter	Erzeugt ein neues System.Windows.Media.Converters.BoolIListConverter-Objekt	WPK
New-Border	Erzeugt ein neues System.Windows.Controls.Border-Objekt	WPK
New-BorderGapMask-Converter	Erzeugt ein neues System.Windows.Controls.BorderGapMaskConverter-Objekt	WPK
New-BrushConverter	Erzeugt ein neues System.Windows.Media.BrushConverter-Objekt	WPK
New-BrushValueSerializer	Erzeugt ein neues System.Windows.Media.Converters.BrushValueSerializer-Objekt	WPK
New-BulletDecorator	Erzeugt ein neues System.Windows.Controls.Primitives.BulletDecorator-Objekt	WPK
New-Button	Erzeugt ein neues System.Windows.Controls.Button-Objekt	WPK
New-ByteAnimation	Erzeugt ein neues System.Windows.Media.Animation.ByteAnimation-Objekt	WPK
New-ByteAnimationUsing-KeyFrames	Erzeugt ein neues System.Windows.Media.Animation.ByteAnimationUsingKey-Frames-Objekt	WPK
New-ByteKeyFrame-Collection	Erzeugt ein neues System.Windows.Media.Animation.ByteKeyFrameCollection-Objekt	WPK
New-Canvas	Erzeugt ein neues System.Windows.Controls.Canvas-Objekt	WPK
New-CharacterMetrics	Erzeugt ein neues System.Windows.Media.CharacterMetrics-Objekt	WPK
New-CharAnimation-UsingKeyFrames	Erzeugt ein neues System.Windows.Media.Animation.CharAnimationUsing-KeyFrames-Objekt	WPK
New-CharIListConverter	Erzeugt ein neues System.Windows.Media.Converters.CharIListConverter-Objekt	WPK
New-CharKeyFrame-Collection	Erzeugt ein neues System.Windows.Media.Animation.CharKeyFrameCollection-Objekt	WPK

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
New-CheckBox	Erzeugt ein neues System.Windows.Controls.CheckBox-Objekt	WPK
New-CheckedListBox	Erzeugt ein neues System.Windows.Forms.CheckedListBox-Objekt	WPK
New-Cluster	Erstellen eines neuen Failoverclusters. Bevor Sie einen Cluster erstellen können, müssen Sie die Hardware (Server, Netzwerke und Speicher) verbinden und Validierungstests ausführen.	FailoverClusters
New-CollectionContainer	Erzeugt ein neues System.Windows.Data.CollectionContainer-Objekt	WPK
New-CollectionViewSource	Erzeugt ein neues System.Windows.Data.CollectionViewSource-Objekt	WPK
New-ColorAnimation	Erzeugt ein neues System.Windows.Media.Animation.ColorAnimation-Objekt	WPK
New-ColorAnimationUsingKeyFrames	Erzeugt ein neues System.Windows.Media.Animation.ColorAnimationUsingKeyFrames-Objekt	WPK
New-ColorConvertedBitmap	Erzeugt ein neues System.Windows.Media.Imaging.ColorConvertedBitmap-Objekt	WPK
New-ColorConvertedBitmapExtension	Erzeugt ein neues System.Windows.ColorConvertedBitmapExtension-Objekt	WPK
New-ColorConverter	Erzeugt ein neues System.Windows.Media.ColorConverter-Objekt	WPK
New-ColorDialog	Erzeugt ein neues System.Windows.Forms.ColorDialog-Objekt	WPK
New-ColorKeyFrameCollection	Erzeugt ein neues System.Windows.Media.Animation.ColorKeyFrameCollection-Objekt	WPK
New-ColumnDefinition	Erzeugt ein neues System.Windows.Controls.ColumnDefinition-Objekt	WPK
New-CombinedGeometry	Erzeugt ein neues System.Windows.Media.CombinedGeometry-Objekt	WPK
New-ComboBox	Erzeugt ein neues System.Windows.Controls.ComboBox-Objekt	WPK
New-ComboBoxItem	Erzeugt ein neues System.Windows.Controls.ComboBoxItem-Objekt	WPK
New-CommandBinding	Erzeugt ein neues System.Windows.Input.CommandBinding-Objekt	WPK
New-CommandBindingCollection	Erzeugt ein neues System.Windows.Input.CommandBindingCollection-Objekt	WPK

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
New-CommandConverter	Erzeugt ein neues System.Windows.Input.CommandConverter-Objekt	WPK
New-Component-ResourceKey	Erzeugt ein neues System.Windows.ComponentResourceKey-Objekt	WPK
New-Component-ResourceKeyConverter	Erzeugt ein neues System.Windows.Markup.ComponentResourceKeyConverter-Objekt	WPK
New-CompositeCollection	Erzeugt ein neues System.Windows.Data.CompositeCollection-Objekt	WPK
New-Condition	Erzeugt ein neues System.Windows.Condition	WPK
New-ConditionCollection	Erzeugt ein neues System.Windows.ConditionCollection-Objekt	WPK
New-ContainerControl	Erzeugt ein neues System.Windows.Forms.ContainerControl-Objekt	WPK
New-ContainerUIElement3D	Erzeugt ein neues System.Windows.Media.Media3D.ContainerUIElement3D-Objekt	WPK
New-ContainerVisual	Erzeugt ein neues System.Windows.Media.ContainerVisual-Objekt	WPK
New-ContentControl	Erzeugt ein neues System.Windows.Controls.ContentControl-Objekt	WPK
New-ContentElement	Erzeugt ein neues System.Windows.ContentElement-Objekt	WPK
New-ContentLocator	Erzeugt ein neues System.Windows.Annotations.ContentLocator-Objekt	WPK
New-ContentLocator-Group	Erzeugt ein neues System.Windows.Annotations.ContentLocatorGroup-Objekt	WPK
New-ContentPresenter	Erzeugt ein neues System.Windows.Controls.ContentPresenter-Objekt	WPK
New-ContentProperty-Attribute	Erzeugt ein neues System.Windows.Markup.ContentPropertyAttribute-Objekt	WPK
New-ContentTypes	Erzeugt ein neues System.Windows.Resources.ContentTypes-Objekt	WPK
New-ContextMenu	Erzeugt ein neues System.Windows.Controls.ContextMenu-Objekt	WPK
New-ContextMenuStrip	Erzeugt ein neues System.Windows.Forms.ContextMenuStrip-Objekt	WPK
New-Control	Erzeugt ein neues System.Windows.Controls.Control-Objekt	WPK

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
New-ControlTemplate	Erzeugt ein neues System.Windows.Controls.ControlTemplate-Objekt	WPK
New-CornerRadius-Converter	Erzeugt ein neues System.Windows.CornerRadiusConverter-Objekt	WPK
New-CroppedBitmap	Erzeugt ein neues System.Windows.Media.Imaging.CroppedBitmap-Objekt	WPK
New-CultureInfoIetf-LanguageTagConverter	Erzeugt ein neues System.Windows.CultureInfoIetfLanguageTagConverter-Objekt	WPK
New-CurrentChanging-EventArgs	Erzeugt ein neues System.ComponentModel.CurrentChangingEventArgs-Objekt	WPK
New-CursorConverter	Erzeugt ein neues System.Windows.Input.CursorConverter-Objekt	WPK
New-D3DImage	Erzeugt ein neues System.Windows.Interop.D3DImage-Objekt	WPK
New-DashStyle	Erzeugt ein neues System.Windows.Media.DashStyle-Objekt	WPK
New-DataErrorValidation-Rule	Erzeugt ein neues System.Windows.Controls.DataErrorValidationRule-Objekt	WPK
New-DataGrid	Erzeugt ein neues System.Windows.Forms.DataGrid-Objekt	WPK
New-DataGridTextBox	Erzeugt ein neues System.Windows.Forms.DataGridTextBox-Objekt	WPK
New-DataGridView	Erzeugt ein neues System.Windows.Forms.DataGridView-Objekt	WPK
New-DataGridView-ComboBoxEditingControl	Erzeugt ein neues System.Windows.Forms.DataGridViewComboBoxEditingControl-Objekt	WPK
New-DataGridViewText-BoxEditingControl	Erzeugt ein neues System.Windows.Forms.DataGridViewTextBoxEditingControl-Objekt	WPK
New-DataObject	Erzeugt ein neues System.Windows.DataObject-Objekt	WPK
New-DataTemplate	Erzeugt ein neues System.Windows.DataTemplate-Objekt	WPK
New-DataTemplateKey	Erzeugt ein neues System.Windows.DataTemplateKey-Objekt	WPK
New-DataTemplate-Selector	Erzeugt ein neues System.Windows.Controls.DataTemplateSelector-Objekt	WPK
New-DataTrigger	Erzeugt ein neues System.Windows.DataTrigger-Objekt	WPK
New-DateTimePicker	Erzeugt ein neues System.Windows.Forms.DateTimePicker-Objekt	WPK

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
New-DateTimeValue-Serializer	Erzeugt ein neues System.Windows.Markup.DateTimeValueSerializer-Objekt	WPK
New-DecimalAnimation	Erzeugt ein neues System.Windows.Media.Animation.DecimalAnimation-Objekt	WPK
New-DecimalAnimation-UsingKeyFrames	Erzeugt ein neues System.Windows.Media.Animation.DecimalAnimationUsingKey-Frames-Objekt	WPK
New-DecimalKeyFrame-Collection	Erzeugt ein neues System.Windows.Media.Animation.DecimalKeyFrameCollection-Objekt	WPK
New-Decorator	Erzeugt ein neues System.Windows.Controls.Decorator-Objekt	WPK
New-DependencyObject	Erzeugt ein neues System.Windows.DependencyObject-Objekt	WPK
New-Dependency-PropertyConverter	Erzeugt ein neues System.Windows.Markup.DependencyPropertyConverter-Objekt	WPK
New-DialogResult-Converter	Erzeugt ein neues System.Windows.DialogResultConverter-Objekt	WPK
New-DiffuseMaterial	Erzeugt ein neues System.Windows.Media.Media3D.DiffuseMaterial-Objekt	WPK
New-DirectionalLight	Erzeugt ein neues System.Windows.Media.Media3D.DirectionalLight-Objekt	WPK
New-DisableDpi-AwarenessAttribute	Erzeugt ein neues System.Windows.Media.DisableDpiAwarenessAttribute-Objekt	WPK
New-DiscreteBooleanKey-Frame	Erzeugt ein neues System.Windows.Media.Animation.DiscreteBooleanKeyFrame-Objekt	WPK
New-DiscreteByteKey-Frame	Erzeugt ein neues System.Windows.Media.Animation.DiscreteByteKeyFrame-Objekt	WPK
New-DiscreteCharKey-Frame	Erzeugt ein neues System.Windows.Media.Animation.DiscreteCharKeyFrame-Objekt	WPK
New-DiscreteColorKey-Frame	Erzeugt ein neues System.Windows.Media.Animation.DiscreteColorKeyFrame-Objekt	WPK
New-DiscreteDecimalKey-Frame	Erzeugt ein neues System.Windows.Media.Animation.DiscreteDecimalKeyFrame-Objekt	WPK
New-DiscreteDoubleKey-Frame	Erzeugt ein neues System.Windows.Media.Animation.DiscreteDoubleKeyFrame-Objekt	WPK
New-DiscreteInt-16KeyFrame	Erzeugt ein neues System.Windows.Media.Animation.DiscreteInt16KeyFrame-Objekt	WPK

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
New-DiscreteInt-32KeyFrame	Erzeugt ein neues System.Windows.Media.Animation.DiscreteInt32KeyFrame-Objekt	WPK
New-DiscreteInt64-KeyFrame	Erzeugt ein neues System.Windows.Media.Animation.DiscreteInt64KeyFrame-Objekt	WPK
New-DiscreteMatrixKey-Frame	Erzeugt ein neues System.Windows.Media.Animation.DiscreteMatrixKeyFrame-Objekt	WPK
New-DiscreteObjectKey-Frame	Erzeugt ein neues System.Windows.Media.Animation.DiscreteObjectKeyFrame-Objekt	WPK
New-DiscretePoint-3DKeyFrame	Erzeugt ein neues System.Windows.Media.Animation.DiscretePoint3DKeyFrame-Objekt	WPK
New-DiscretePointKey-Frame	Erzeugt ein neues System.Windows.Media.Animation.DiscretePointKeyFrame-Objekt	WPK
New-DiscreteQuaternion-KeyFrame	Erzeugt ein neues System.Windows.Media.Animation.DiscreteQuaternionKeyFrame-Objekt	WPK
New-DiscreteRectKey-Frame	Erzeugt ein neues System.Windows.Media.Animation.DiscreteRectKeyFrame-Objekt	WPK
New-DiscreteRotation-3DKeyFrame	Erzeugt ein neues System.Windows.Media.Animation.DiscreteRotation3DKeyFrame-Objekt	WPK
New-DiscreteSingleKey-Frame	Erzeugt ein neues System.Windows.Media.Animation.DiscreteSingleKeyFrame-Objekt	WPK
New-DiscreteSizeKey-Frame	Erzeugt ein neues System.Windows.Media.Animation.DiscreteSizeKeyFrame-Objekt	WPK
New-DiscreteStringKey-Frame	Erzeugt ein neues System.Windows.Media.Animation.DiscreteStringKeyFrame-Objekt	WPK
New-DiscreteThickness-KeyFrame	Erzeugt ein neues System.Windows.Media.Animation.DiscreteThicknessKeyFrame-Objekt	WPK
New-DiscreteVector-3DKeyFrame	Erzeugt ein neues System.Windows.Media.Animation.DiscreteVector3DKeyFrame-Objekt	WPK
New-DiscreteVectorKey-Frame	Erzeugt ein neues System.Windows.Media.Animation.DiscreteVectorKeyFrame-Objekt	WPK
New-DispatcherFrame	Erzeugt ein neues System.Windows.Threading.DispatcherFrame-Objekt	WPK
New-Dispatcher-SynchronizationContext	Erzeugt ein neues System.Windows.Threading.DispatcherSynchronization-Context-Objekt	WPK
New-DispatcherTimer	Erzeugt ein neues System.Windows.Threading.DispatcherTimer-Objekt	WPK

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
New-DockPanel	Erzeugt ein neues System.Windows.Controls.DockPanel-Objekt	WPK
New-DocObjHost	Erzeugt ein neues System.Windows.Interop.DocObjHost-Objekt	WPK
New-DocumentPageView	Erzeugt ein neues System.Windows.Controls.Primitives.DocumentPageView-Objekt	WPK
New-DocumentReference	Erzeugt ein neues System.Windows.Documents.DocumentReference-Objekt	WPK
New-DocumentViewer	Erzeugt ein neues System.Windows.Controls.DocumentViewer-Objekt	WPK
New-DomainUpDown	Erzeugt ein neues System.Windows.Forms.DomainUpDown-Objekt	WPK
New-DoubleAnimation	Erzeugt ein neues System.Windows.Media.Animation.DoubleAnimation-Objekt	WPK
New-DoubleAnimationUsingKeyFrames	Erzeugt ein neues System.Windows.Media.Animation.DoubleAnimationUsingKeyFrames-Objekt	WPK
New-DoubleAnimationUsingPath	Erzeugt ein neues System.Windows.Media.Animation.DoubleAnimationUsingPath-Objekt	WPK
New-DoubleCollection	Erzeugt ein neues System.Windows.Media.DoubleCollection-Objekt	WPK
New-DoubleCollectionConverter	Erzeugt ein neues System.Windows.Media.DoubleCollectionConverter-Objekt	WPK
New-DoubleCollectionValueSerializer	Erzeugt ein neues System.Windows.Media.Converters.DoubleCollectionValueSerializer-Objekt	WPK
New-DoubleIListConverter	Erzeugt ein neues System.Windows.Media.Converters.DoubleIListConverter-Objekt	WPK
New-DoubleKeyFrameCollection	Erzeugt ein neues System.Windows.Media.Animation.DoubleKeyFrameCollection-Objekt	WPK
New-DrawingAttributes	Erzeugt ein neues System.Windows.Ink.DrawingAttributes-Objekt	WPK
New-DrawingBrush	Erzeugt ein neues System.Windows.Media.DrawingBrush-Objekt	WPK
New-DrawingCollection	Erzeugt ein neues System.Windows.Media.DrawingCollection-Objekt	WPK
New-DrawingGroup	Erzeugt ein neues System.Windows.Media.DrawingGroup-Objekt	WPK

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
New-DrawingImage	Erzeugt ein neues System.Windows.Media.DrawingImage-Objekt	WPK
New-DrawingVisual	Erzeugt ein neues System.Windows.Media.DrawingVisual-Objekt	WPK
New-DropShadow-BitmapEffect	Erzeugt ein neues System.Windows.Media.Effects.DropShadowBitmapEffect-Objekt	WPK
New-DropShadowEffect	Erzeugt ein neues System.Windows.Media.Effects.DropShadowEffect-Objekt	WPK
New-DurationConverter	Erzeugt ein neues System.Windows.DurationConverter-Objekt	WPK
New-DynamicRenderer	Erzeugt ein neues System.Windows.Input.StylusPlugIns.DynamicRenderer-Objekt	WPK
New-DynamicResource-Extension	Erzeugt ein neues System.Windows.DynamicResourceExtension-Objekt	WPK
New-DynamicResource-ExtensionConverter	Erzeugt ein neues System.Windows.DynamicResourceExtensionConverter-Objekt	WPK
New-ElementHost	Erzeugt ein neues System.Windows.Forms.Integration.ElementHost-Objekt	WPK
New-ElementLocalizability	Erzeugt ein neues System.Windows.Markup.Localizer.ElementLocalizability-Objekt	WPK
New-Ellipse	Erzeugt ein neues System.Windows.Shapes.Ellipse-Objekt	WPK
New-EllipseGeometry	Erzeugt ein neues System.Windows.Media.EllipseGeometry-Objekt	WPK
New-EmbossBitmapEffect	Erzeugt ein neues System.Windows.Media.Effects.EmbossBitmapEffect-Objekt	WPK
New-EmissiveMaterial	Erzeugt ein neues System.Windows.Media.Media3D.EmissiveMaterial-Objekt	WPK
New-Enum	Erzeugt ein neues enumerated type-Objekt	PSCodeGen
New-Event	Erstellt ein neues Ereignis.	Microsoft.PowerShell.Utility
New-EventLog	Erstellt ein neues Ereignisprotokoll und eine neue Ereignisquelle auf einem lokalen oder einem Remotecomputer.	Microsoft.PowerShell.Management
New-EventPrivateKey	Erzeugt ein neues System.Windows.EventPrivateKey-Objekt	WPK
New-EventSetter	Erzeugt ein neues System.Windows.EventSetter-Objekt	WPK
New-EventTrigger	Erzeugt ein neues System.Windows.EventTrigger-Objekt	WPK

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
New-ExceptionValidation-Rule	Erzeugt ein neues System.Windows. Controls.ExceptionValidationRule-Objekt	WPK
New-Expander	Erzeugt ein neues System.Windows. Controls.Expander-Objekt	WPK
New-ExpressionConverter	Erzeugt ein neues System.Windows. ExpressionConverter-Objekt	WPK
New-FamilyTypeface	Erzeugt ein neues System.Windows.Media. FamilyTypeface-Objekt	WPK
New-Feed	Abonniert einen RSS Feed.	PSRss
New-Figure	Erzeugt ein neues System.Windows. Documents.Figure-Objekt	WPK
New-FigureLength-Converter	Erzeugt ein neues System.Windows.Figure-LengthConverter-Objekt	WPK
New-FigureStructure	Erzeugt ein neues System.Windows. Documents.DocumentStructures.Figure-Structure-Objekt	WPK
New-FileFormatException	Erzeugt ein neues System.IO.FileFormat-Exception-Objekt	WPK
New-FixedDocument	Erzeugt ein neues System.Windows. Documents.FixedDocument-Objekt	WPK
New-FixedDocument-Sequence	Erzeugt ein neues System.Windows.Docu-ments.FixedDocumentSequence-Objekt	WPK
New-FixedPage	Erzeugt ein neues System.Windows. Documents.FixedPage-Objekt	WPK
New-Floater	Erzeugt ein neues System.Windows. Documents.Floater-Objekt	WPK
New-FlowDocument	Erzeugt ein neues System.Windows. Documents.FlowDocument-Objekt	WPK
New-FlowDocumentPage-Viewer	Erzeugt ein neues System.Windows. Controls.FlowDocumentPageViewer-Objekt	WPK
New-FlowDocument-Reader	Erzeugt ein neues System.Windows. Controls.FlowDocumentReader-Objekt	WPK
New-FlowDocument-ScrollViewer	Erzeugt ein neues System.Windows. Controls.FlowDocumentScrollViewer-Objekt	WPK
New-FlowLayoutPanel	Erzeugt ein neues System.Windows.Forms. FlowLayoutPanel-Objekt	WPK
New-FolderBrowserDialog	Erzeugt ein neues System.Windows.Forms. FolderBrowserDialog-Objekt	WPK
New-FontDialog	Erzeugt ein neues System.Windows.Forms. FontDialog-Objekt	WPK

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
New-FontEmbedding-Manager	Erzeugt ein neues System.Windows.Media.FontEmbeddingManager-Objekt	WPK
New-FontFamily	Erzeugt ein neues System.Windows.Media.FontFamily-Objekt	WPK
New-FontFamilyConverter	Erzeugt ein neues System.Windows.Media.FontFamilyConverter-Objekt	WPK
New-FontFamilyMap	Erzeugt ein neues System.Windows.Media.FontFamilyMap-Objekt	WPK
New-FontFamilyValue-Serializer	Erzeugt ein neues System.Windows.Media.FontFamilyValueSerializer-Objekt	WPK
New-FontSizeConverter	Erzeugt ein neues System.Windows.FontSizeConverter-Objekt	WPK
New-FontStretchConverter	Erzeugt ein neues System.Windows.FontStretchConverter-Objekt	WPK
New-FontStyleConverter	Erzeugt ein neues System.Windows.FontStyleConverter-Objekt	WPK
New-FontWeight-Converter	Erzeugt ein neues System.Windows.FontWeightConverter-Objekt	WPK
New-Form	Erzeugt ein neues System.Windows.Forms.Form-Objekt	WPK
New-FormatConverted-Bitmap	Erzeugt ein neues System.Windows.Media.Imaging.FormatConvertedBitmap-Objekt	WPK
New-Frame	Erzeugt ein neues System.Windows.Controls.Frame-Objekt	WPK
New-FrameworkContent-Element	Erzeugt ein neues System.Windows.FrameworkContentElement-Objekt	WPK
New-FrameworkElement	Erzeugt ein neues System.Windows.FrameworkElement-Objekt	WPK
New-FrameworkElement-Factory	Erzeugt ein neues System.Windows.FrameworkElementFactory-Objekt	WPK
New-FrameworkProperty-Metadata	Erzeugt ein neues System.Windows.FrameworkPropertyMetadata-Objekt	WPK
New-GeneralTransform-3DCollection	Erzeugt ein neues System.Windows.Media.Media3D.GeneralTransform3DCollection-Objekt	WPK
New-GeneralTransform-3DGroup	Erzeugt ein neues System.Windows.Media.Media3D.GeneralTransform3DGroup-Objekt	WPK
New-GeneralTransform-Collection	Erzeugt ein neues System.Windows.Media.GeneralTransformCollection-Objekt	WPK

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
New-GeneralTransform-Group	Erzeugt ein neues System.Windows.Media.GeneralTransformGroup-Objekt	WPK
New-GeometryCollection	Erzeugt ein neues System.Windows.Media.GeometryCollection-Objekt	WPK
New-GeometryConverter	Erzeugt ein neues System.Windows.Media.GeometryConverter-Objekt	WPK
New-GeometryDrawing	Erzeugt ein neues System.Windows.Media.GeometryDrawing-Objekt	WPK
New-GeometryGroup	Erzeugt ein neues System.Windows.Media.GeometryGroup-Objekt	WPK
New-GeometryModel3D	Erzeugt ein neues System.Windows.Media.Media3D.GeometryModel3D-Objekt	WPK
New-GeometryValue-Serializer	Erzeugt ein neues System.Windows.Media.Converters.GeometryValueSerializer-Objekt	WPK
New-GestureRecognizer	Erzeugt ein neues System.Windows.Ink.GestureRecognizer-Objekt	WPK
New-GifBitmapEncoder	Erzeugt ein neues System.Windows.Media.Imaging.GifBitmapEncoder-Objekt	WPK
New-GlyphRun	Erzeugt ein neues System.Windows.Media.GlyphRun-Objekt	WPK
New-GlyphRunDrawing	Erzeugt ein neues System.Windows.Media.GlyphRunDrawing-Objekt	WPK
New-Glyphs	Erzeugt ein neues System.Windows.Documents.Glyphs-Objekt	WPK
New-GlyphTypeface	Erzeugt ein neues System.Windows.Media.GlyphTypeface-Objekt	WPK
New-GPLink	Verknüpft ein Gruppenrichtlinienobjekt mit einer Site, Domäne oder Organisationseinheit.	GroupPolicy
New-GPO	Erstellt ein neues Gruppenrichtlinienobjekt.	GroupPolicy
New-GPStarterGPO	Erstellt ein neues Starter-Gruppenrichtlinienobjekt.	GroupPolicy
New-GradientStop	Erzeugt ein neues System.Windows.Media.GradientStop-Objekt	WPK
New-GradientStop-Collection	Erzeugt ein neues System.Windows.Media.GradientStopCollection-Objekt	WPK
New-Grid	Erzeugt ein neues System.Windows.Controls.Grid-Objekt	WPK
New-GridLengthConverter	Erzeugt ein neues System.Windows.GridLengthConverter-Objekt	WPK

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
New-GridSplitter	Erzeugt ein neues System.Windows.Controls.GridSplitter-Objekt	WPK
New-GridView	Erzeugt ein neues System.Windows.Controls.GridView-Objekt	WPK
New-GridViewColumn	Erzeugt ein neues System.Windows.Controls.GridViewColumn-Objekt	WPK
New-GridViewColumnCollection	Erzeugt ein neues System.Windows.Controls.GridViewColumnCollection-Objekt	WPK
New-GridViewColumnHeader	Erzeugt ein neues System.Windows.Controls.GridViewColumnHeader-Objekt	WPK
New-GridViewHeaderRowPresenter	Erzeugt ein neues System.Windows.Controls.GridViewHeaderRowPresenter-Objekt	WPK
New-GridViewRowPresenter	Erzeugt ein neues System.Windows.Controls.GridViewRowPresenter-Objekt	WPK
New-GroupBox	Erzeugt ein neues System.Windows.Controls.GroupBox-Objekt	WPK
New-GroupItem	Erzeugt ein neues System.Windows.Controls.GroupItem-Objekt	WPK
New-GroupStyle	Erzeugt ein neues System.Windows.Controls.GroupStyle-Objekt	WPK
New-GuidelineSet	Erzeugt ein neues System.Windows.Media.GuidelineSet-Objekt	WPK
New-HeaderedContentControl	Erzeugt ein neues System.Windows.Controls.HeaderedContentControl-Objekt	WPK
New-HeaderedItemsControl	Erzeugt ein neues System.Windows.Controls.HeaderedItemsControl-Objekt	WPK
New-HierarchicalDataTemplate	Erzeugt ein neues System.Windows.HierarchicalDataTemplate-Objekt	WPK
New-HostVisual	Erzeugt ein neues System.Windows.Media.HostVisual-Objekt	WPK
New-HScrollBar	Erzeugt ein neues System.Windows.Forms.HScrollBar-Objekt	WPK
New-Hyperlink	Erzeugt ein neues System.Windows.Documents.Hyperlink-Objekt	WPK
New-Image	Erzeugt ein neues System.Windows.Controls.Image-Objekt	WPK
New-ImageBrush	Erzeugt ein neues System.Windows.Media.ImageBrush-Objekt	WPK
New-ImageDrawing	Erzeugt ein neues System.Windows.Media.ImageDrawing-Objekt	WPK

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
New-ImageSource-Converter	Erzeugt ein neues System.Windows.Media.ImageSourceConverter-Objekt	WPK
New-ImageSourceValue-Serializer	Erzeugt ein neues System.Windows.Media.ImageSourceValueSerializer-Objekt	WPK
New-InkCanvas	Erzeugt ein neues System.Windows.Controls.InkCanvas-Objekt	WPK
New-InkPresenter	Erzeugt ein neues System.Windows.Controls.InkPresenter-Objekt	WPK
New-InlineUIContainer	Erzeugt ein neues System.Windows.Documents.InlineUIContainer-Objekt	WPK
New-InputBinding-Collection	Erzeugt ein neues System.Windows.Input.InputBindingCollection-Objekt	WPK
New-InputGesture-Collection	Erzeugt ein neues System.Windows.Input.InputGestureCollection-Objekt	WPK
New-InputScope	Erzeugt ein neues System.Windows.Input.InputScope-Objekt	WPK
New-InputScopeConverter	Erzeugt ein neues System.Windows.Input.InputScopeConverter-Objekt	WPK
New-InputScopeName	Erzeugt ein neues System.Windows.Input.InputScopeName-Objekt	WPK
New-InputScopeName-Converter	Erzeugt ein neues System.Windows.Input.InputScopeNameConverter-Objekt	WPK
New-InputScopePhrase	Erzeugt ein neues System.Windows.Input.InputScopePhrase-Objekt	WPK
New-Int16Animation	Erzeugt ein neues System.Windows.Media.Animation.Int16Animation-Objekt	WPK
New-Int16Animation-UsingKeyFrames	Erzeugt ein neues System.Windows.Media.Animation.Int16AnimationUsingKey-Frames-Objekt	WPK
New-Int16KeyFrame-Collection	Erzeugt ein neues System.Windows.Media.Animation.Int16KeyFrameCollection-Objekt	WPK
New-Int32Animation	Erzeugt ein neues System.Windows.Media.Animation.Int32Animation-Objekt	WPK
New-Int32Animation-UsingKeyFrames	Erzeugt ein neues System.Windows.Media.Animation.Int32AnimationUsingKey-Frames-Objekt	WPK
New-Int32Collection	Erzeugt ein neues System.Windows.Media.Int32Collection-Objekt	WPK
New-Int32Collection-Converter	Erzeugt ein neues System.Windows.Media.Int32CollectionConverter-Objekt	WPK

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
New-Int32Collection-ValueSerializer	Erzeugt ein neues System.Windows.Media.Converters.Int32CollectionValueSerializer-Objekt	WPK
New-Int32KeyFrame-Collection	Erzeugt ein neues System.Windows.Media.Animation.Int32KeyFrameCollection-Objekt	WPK
New-Int32RectConverter	Erzeugt ein neues System.Windows.Int32RectConverter-Objekt	WPK
New-Int32RectValue-Serializer	Erzeugt ein neues System.Windows.Converters.Int32RectValueSerializer-Objekt	WPK
New-Int64Animation	Erzeugt ein neues System.Windows.Media.Animation.Int64Animation-Objekt	WPK
New-Int64Animation-UsingKeyFrames	Erzeugt ein neues System.Windows.Media.Animation.Int64AnimationUsingKey-Frames-Objekt	WPK
New-Int64KeyFrame-Collection	Erzeugt ein neues System.Windows.Media.Animation.Int64KeyFrameCollection-Objekt	WPK
New-InvalidWmp-VersionException	Erzeugt ein neues System.Windows.Media.InvalidWmpVersionException-Objekt	WPK
New-IseScript	Erzeugt ein neues script file in the ISE-Objekt	PowerShellPack
New-Italic	Erzeugt ein neues System.Windows.Documents.Italic-Objekt	WPK
New-Item	Erstellt ein neues Element-Objekt	Microsoft.Power-Shell.Management
New-ItemProperty	Erstellt eine neue Eigenschaft für ein Element und legt den Wert fest. Sie können z.B.. mit New-ItemProperty Registrierungswerte und Daten erstellen und ändern, bei denen es sich um Eigenschaften eines Registrierungsschlüssels handelt.	Microsoft.Power-Shell.Management
New-ItemsControl	Erzeugt ein neues System.Windows.Controls.ItemsControl-Objekt	WPK
New-ItemsPanelTemplate	Erzeugt ein neues System.Windows.Controls.ItemsPanelTemplate-Objekt	WPK
New-ItemsPresenter	Erzeugt ein neues System.Windows.Controls.ItemsPresenter-Objekt	WPK
New-JournalEntryList-Converter	Erzeugt ein neues System.Windows.Navigation.JournalEntryListConverter-Objekt	WPK

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
New-JournalEntryUnified-ViewConverter	Erzeugt ein neues System.Windows. Navigation.JournalEntryUnifiedView-Converter-Objekt	WPK
New-JpegBitmapEncoder	Erzeugt ein neues System.Windows.Media. Imaging.JpegBitmapEncoder-Objekt	WPK
New-KeyBinding	Erzeugt ein neues System.Windows.Input. KeyBinding-Objekt	WPK
New-KeyConverter	Erzeugt ein neues System.Windows.Input. KeyConverter-Objekt	WPK
New-KeyGestureConverter	Erzeugt ein neues System.Windows.Input. KeyGestureConverter-Objekt	WPK
New-KeyGestureValue-Serializer	Erzeugt ein neues System.Windows.Input. KeyGestureValueSerializer-Objekt	WPK
New-KeySpline	Erzeugt ein neues System.Windows.Media. Animation.KeySpline-Objekt	WPK
New-KeySplineConverter	Erzeugt ein neues System.Windows.Key-SplineConverter-Objekt	WPK
New-KeyTimeConverter	Erzeugt ein neues System.Windows.Key-TimeConverter-Objekt	WPK
New-KeyValueSerializer	Erzeugt ein neues System.Windows.Input. KeyValueSerializer-Objekt	WPK
New-Label	Erzeugt ein neues System.Windows. Controls.Label-Objekt	WPK
New-LengthConverter	Erzeugt ein neues System.Windows. LengthConverter-Objekt	WPK
New-Line	Erzeugt ein neues System.Windows. Shapes.Line-Objekt	WPK
New-LinearByteKeyFrame	Erzeugt ein neues System.Windows.Media. Animation.LinearByteKeyFrame-Objekt	WPK
New-LinearColorKeyFrame	Erzeugt ein neues System.Windows.Media. Animation.LinearColorKeyFrame-Objekt	WPK
New-LinearDecimalKey-Frame	Erzeugt ein neues System.Windows.Media. Animation.LinearDecimalKeyFrame-Objekt	WPK
New-LinearDoubleKey-Frame	Erzeugt ein neues System.Windows.Media. Animation.LinearDoubleKeyFrame-Objekt	WPK
New-LinearGradientBrush	Erzeugt ein neues System.Windows.Media. LinearGradientBrush-Objekt	WPK
New-LinearInt16KeyFrame	Erzeugt ein neues System.Windows.Media. Animation.LinearInt16KeyFrame-Objekt	WPK
New-LinearInt32KeyFrame	Erzeugt ein neues System.Windows.Media. Animation.LinearInt32KeyFrame-Objekt	WPK

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
New-LinearInt64KeyFrame	Erzeugt ein neues System.Windows.Media.Animation.LinearInt64KeyFrame-Objekt	WPK
New-LinearPoint3DKeyFrame	Erzeugt ein neues System.Windows.Media.Animation.LinearPoint3DKeyFrame-Objekt	WPK
New-LinearPointKeyFrame	Erzeugt ein neues System.Windows.Media.Animation.LinearPointKeyFrame-Objekt	WPK
New-LinearQuaternionKeyFrame	Erzeugt ein neues System.Windows.Media.Animation.LinearQuaternionKeyFrame-Objekt	WPK
New-LinearRectKeyFrame	Erzeugt ein neues System.Windows.Media.Animation.LinearRectKeyFrame-Objekt	WPK
New-LinearRotation3DKeyFrame	Erzeugt ein neues System.Windows.Media.Animation.LinearRotation3DKeyFrame-Objekt	WPK
New-LinearSingleKeyFrame	Erzeugt ein neues System.Windows.Media.Animation.LinearSingleKeyFrame-Objekt	WPK
New-LinearSizeKeyFrame	Erzeugt ein neues System.Windows.Media.Animation.LinearSizeKeyFrame-Objekt	WPK
New-LinearThicknessKeyFrame	Erzeugt ein neues System.Windows.Media.Animation.LinearThicknessKeyFrame-Objekt	WPK
New-LinearVector3DKeyFrame	Erzeugt ein neues System.Windows.Media.Animation.LinearVector3DKeyFrame-Objekt	WPK
New-LinearVectorKeyFrame	Erzeugt ein neues System.Windows.Media.Animation.LinearVectorKeyFrame-Objekt	WPK
New-LineBreak	Erzeugt ein neues System.Windows.Documents.LineBreak-Objekt	WPK
New-LineGeometry	Erzeugt ein neues System.Windows.Media.LineGeometry-Objekt	WPK
New-LineSegment	Erzeugt ein neues System.Windows.Media.LineSegment-Objekt	WPK
New-LinkLabel	Erzeugt ein neues System.Windows.Forms.LinkLabel-Objekt	WPK
New-LinkTarget	Erzeugt ein neues System.Windows.Documents.LinkTarget-Objekt	WPK
New-LinkTargetCollection	Erzeugt ein neues System.Windows.Documents.LinkTargetCollection-Objekt	WPK
New-List	Erzeugt ein neues System.Windows.Documents.List-Objekt	WPK

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
New-ListBox	Erzeugt ein neues System.Windows.Controls.ListBox-Objekt	WPK
New-ListBoxItem	Erzeugt ein neues System.Windows.Controls.ListBoxItem-Objekt	WPK
New-ListItem	Erzeugt ein neues System.Windows.Documents.ListItem-Objekt	WPK
New-ListItemStructure	Erzeugt ein neues System.Windows.Documents.DocumentStructures.ListItemStructure-Objekt	WPK
New-ListStructure	Erzeugt ein neues System.Windows.Documents.DocumentStructures.ListStructure-Objekt	WPK
New-ListView	Erzeugt ein neues System.Windows.Controls.ListView-Objekt	WPK
New-ListViewItem	Erzeugt ein neues System.Windows.Controls.ListViewItem-Objekt	WPK
New-MarkupExtensionReturnTypeAttribute	Erzeugt ein neues System.Windows.Markup.MarkupExtensionReturnTypeAttribute-Objekt	WPK
New-MaskedTextBox	Erzeugt ein neues System.Windows.Forms.MaskedTextBox-Objekt	WPK
New-MaterialCollection	Erzeugt ein neues System.Windows.Media.Media3D.MaterialCollection-Objekt	WPK
New-MaterialGroup	Erzeugt ein neues System.Windows.Media.Media3D.MaterialGroup-Objekt	WPK
New-Matrix3DConverter	Erzeugt ein neues System.Windows.Media.Media3D.Matrix3DConverter-Objekt	WPK
New-Matrix3DValueSerializer	Erzeugt ein neues System.Windows.Media.Media3D.Converters.Matrix3DValueSerializer-Objekt	WPK
New-MatrixAnimationUsingKeyFrames	Erzeugt ein neues System.Windows.Media.Animation.MatrixAnimationUsingKeyFrames-Objekt	WPK
New-MatrixAnimationUsingPath	Erzeugt ein neues System.Windows.Media.Animation.MatrixAnimationUsingPath-Objekt	WPK
New-MatrixCamera	Erzeugt ein neues System.Windows.Media.Media3D.MatrixCamera-Objekt	WPK
New-MatrixConverter	Erzeugt ein neues System.Windows.Media.MatrixConverter-Objekt	WPK

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
New-MatrixKeyFrame-Collection	Erzeugt ein neues System.Windows.Media.Animation.MatrixKeyFrameCollection-Objekt	WPK
New-MatrixTransform	Erzeugt ein neues System.Windows.Media.MatrixTransform-Objekt	WPK
New-MatrixTransform3D	Erzeugt ein neues System.Windows.Media.Media3D.MatrixTransform3D-Objekt	WPK
New-MatrixValueSerializer	Erzeugt ein neues System.Windows.Media.onverters.MatrixValueSerializer-Objekt	WPK
New-MdiClient	Erzeugt ein neues System.Windows.Forms.MdiClient-Objekt	WPK
New-MediaElement	Erzeugt ein neues System.Windows.Controls.MediaElement-Objekt	WPK
New-MediaPermission	Erzeugt ein neues System.Security.Permissions.MediaPermission-Objekt	WPK
New-MediaPlayer	Erzeugt ein neues System.Windows.Media.MediaPlayer-Objekt	WPK
New-MediaTimeline	Erzeugt ein neues System.Windows.Media.MediaTimeline-Objekt	WPK
New-Menu	Erzeugt ein neues System.Windows.Controls.Menu-Objekt	WPK
New-MenuItem	Erzeugt ein neues System.Windows.Controls.MenuItem-Objekt	WPK
New-MenuScrolling-VisibilityConverter	Erzeugt ein neues System.Windows.Controls.MenuScrollingVisibilityConverter-Objekt	WPK
New-MenuStrip	Erzeugt ein neues System.Windows.Forms.MenuStrip-Objekt	WPK
New-MeshGeometry3D	Erzeugt ein neues System.Windows.Media.Media3D.MeshGeometry3D-Objekt	WPK
New-Model3DCollection	Erzeugt ein neues System.Windows.Media.Media3D.Model3DCollection-Objekt	WPK
New-Model3DGroup	Erzeugt ein neues System.Windows.Media.Media3D.Model3DGroup-Objekt	WPK
New-ModelUIElement3D	Erzeugt ein neues System.Windows.Media.Media3D.ModelUIElement3D-Objekt	WPK
New-ModelVisual3D	Erzeugt ein neues System.Windows.Media.Media3D.ModelVisual3D-Objekt	WPK
New-ModifierKeys-Converter	Erzeugt ein neues System.Windows.Input.ModifierKeysConverter-Objekt	WPK

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
New-ModifierKeysValue-Serializer	Erzeugt ein neues System.Windows.Input. ModifierKeysValueSerializer-Objekt	WPK
New-Module	Erstellt ein neues dynamisches Modul, das nur im Arbeitsspeicher vorhanden ist.	Microsoft.Power-Shell.Core
New-ModuleManifest	Erstellt ein neues Modulmanifest.	Microsoft.Power-Shell.Core
New-MonthCalendar	Erzeugt ein neues System.Windows.Forms. MonthCalendar-Objekt	WPK
New-MouseAction-Converter	Erzeugt ein neues System.Windows.Input. MouseActionConverter-Objekt	WPK
New-MouseActionValue-Serializer	Erzeugt ein neues System.Windows.Input. MouseActionValueSerializer-Objekt	WPK
New-MouseBinding	Erzeugt ein neues System.Windows.Input. MouseBinding-Objekt	WPK
New-MouseGesture	Erzeugt ein neues System.Windows.Input. MouseGesture-Objekt	WPK
New-MouseGesture-Converter	Erzeugt ein neues System.Windows.Input. MouseGestureConverter-Objekt	WPK
New-MouseGestureValue-Serializer	Erzeugt ein neues System.Windows.Input. MouseGestureValueSerializer-Objekt	WPK
New-MultiBinding	Erzeugt ein neues System.Windows.Data. MultiBinding-Objekt	WPK
New-MultiDataTrigger	Erzeugt ein neues System.Windows. MultiDataTrigger-Objekt	WPK
New-MultiTrigger	Erzeugt ein neues System.Windows. MultiTrigger-Objekt	WPK
New-NamedElement	Erzeugt ein neues System.Windows. Documents.DocumentStructures.Named-Element-Objekt	WPK
New-NameScope	Erzeugt ein neues System.Windows. NameScope-Objekt	WPK
New-NamespaceMapEntry	Erzeugt ein neues System.Windows. Markup.NamespaceMapEntry-Objekt	WPK
New-NavigationWindow	Erzeugt ein neues System.Windows. Navigation.NavigationWindow-Objekt	WPK
New-NlbCluster	Erstellt einen NLB-Cluster in der angegebenen Schnittstelle, der durch den Knoten und den Netzwerkadapternamen definiert wird.	NetworkLoad-BalancingClusters
New-NlbClusterIpv6-Address	Generiert eine IPv6-Adresse, die für die NLB-Cmdlets verwendet wird.	NetworkLoad-BalancingClusters

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
New-NullableBool-Converter	Erzeugt ein neues System.Windows. NullableBoolConverter-Objekt	WPK
New-NullExtension	Erzeugt ein neues System.Windows. Markup.NullExtension-Objekt	WPK
New-NumberSubstitution	Erzeugt ein neues System.Windows.Media. NumberSubstitution-Objekt	WPK
New-NumericUpDown	Erzeugt ein neues System.Windows.Forms. NumericUpDown-Objekt	WPK
New-Object	Erstellt eine Instanz eines Microsoft .NET Framework-Objekts oder COM-Objekts.	Microsoft.Power-Shell.Utility
New-ObjectAnimation-UsingKeyFrames	Erzeugt ein neues System.Windows.Media. Animation.ObjectAnimationUsingKey-Frames-Objekt	WPK
New-ObjectDataProvider	Erzeugt ein neues System.Windows.Data. ObjectDataProvider-Objekt	WPK
New-ObjectKeyFrame-Collection	Erzeugt ein neues System.Windows.Media. Animation.ObjectKeyFrameCollection-Objekt	WPK
New-OpenFileDialog	Erzeugt ein neues Microsoft.Win32.Open-FileDialog-Objekt	WPK
New-OrthographicCamera	Erzeugt ein neues System.Windows.Media. Media3D.OrthographicCamera-Objekt	WPK
New-OuterGlowBitmap-Effect	Erzeugt ein neues System.Windows.Media. Effects.OuterGlowBitmapEffect-Objekt	WPK
New-PackWebRequest-Factory	Erzeugt ein neues System.IO.Packaging. PackWebRequestFactory-Objekt	WPK
New-Page	Erzeugt ein neues System.Windows. Controls.Page-Objekt	WPK
New-PageContent	Erzeugt ein neues System.Windows. Documents.PageContent-Objekt	WPK
New-PageSetupDialog	Erzeugt ein neues System.Windows.Forms. PageSetupDialog-Objekt	WPK
New-Panel	Erzeugt ein neues System.Windows.Forms. Panel-Objekt	WPK
New-Paragraph	Erzeugt ein neues System.Windows. Documents.Paragraph-Objekt	WPK
New-ParagraphStructure	Erzeugt ein neues System.Windows.Docu-ments.DocumentStructures.Paragraph-Structure-Objekt	WPK
New-ParallelTimeline	Erzeugt ein neues System.Windows.Media. Animation.ParallelTimeline-Objekt	WPK

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
New-ParserContext	Erzeugt ein neues System.Windows.Markup.ParserContext-Objekt	WPK
New-PasswordBox	Erzeugt ein neues System.Windows.Controls.PasswordBox-Objekt	WPK
New-Path	Erzeugt ein neues System.Windows.Shapes.Path-Objekt	WPK
New-PathFigure	Erzeugt ein neues System.Windows.Media.PathFigure-Objekt	WPK
New-PathFigureCollection	Erzeugt ein neues System.Windows.Media.PathFigureCollection-Objekt	WPK
New-PathFigure-CollectionConverter	Erzeugt ein neues System.Windows.Media.PathFigureCollectionConverter-Objekt	WPK
New-PathFigure-CollectionValueSerializer	Erzeugt ein neues System.Windows.Media.Converters.PathFigureCollectionValue-Serializer-Objekt	WPK
New-PathGeometry	Erzeugt ein neues System.Windows.Media.PathGeometry-Objekt	WPK
New-PathSegment-Collection	Erzeugt ein neues System.Windows.Media.PathSegmentCollection-Objekt	WPK
New-PauseStoryboard	Erzeugt ein neues System.Windows.Media.Animation.PauseStoryboard-Objekt	WPK
New-Pen	Erzeugt ein neues System.Windows.Media.Pen-Objekt	WPK
New-PerspectiveCamera	Erzeugt ein neues System.Windows.Media.Media3D.PerspectiveCamera-Objekt	WPK
New-PictureBox	Erzeugt ein neues System.Windows.Forms.PictureBox-Objekt	WPK
New-PInvoke	Generiert C#-Wrappercode für den Zugriff auf eine WPI-Funktion via Platform Invoke	PSCodeGen
New-PixelFormat-Converter	Erzeugt ein neues System.Windows.Media.PixelFormatConverter-Objekt	WPK
New-PixelShader	Erzeugt ein neues System.Windows.Media.Effects.PixelShader-Objekt	WPK
New-PngBitmapEncoder	Erzeugt ein neues System.Windows.Media.Imaging.PngBitmapEncoder-Objekt	WPK
New-Point3DAnimation	Erzeugt ein neues System.Windows.Media.Animation.Point3DAnimation-Objekt	WPK
New-Point3DAnimation-UsingKeyFrames	Erzeugt ein neues System.Windows.Media.Animation.Point3DAnimationUsingKey-Frames-Objekt	WPK

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
New-Point3DCollection	Erzeugt ein neues System.Windows.Media.Media3D.Point3DCollection-Objekt	WPK
New-Point3DCollection-Converter	Erzeugt ein neues System.Windows.Media.Media3D.Point3DCollectionConverter-Objekt	WPK
New-Point3DCollection-ValueSerializer	Erzeugt ein neues System.Windows.Media.Media3D.Converters.Point3DCollection-ValueSerializer-Objekt	WPK
New-Point3DConverter	Erzeugt ein neues System.Windows.Media.Media3D.Point3DConverter-Objekt	WPK
New-Point3DKeyFrame-Collection	Erzeugt ein neues System.Windows.Media.Animation.Point3DKeyFrameCollection-Objekt	WPK
New-Point3DValue-Serializer	Erzeugt ein neues System.Windows.Media.Media3D.Converters.Point3DValue-Serializer-Objekt	WPK
New-Point4DConverter	Erzeugt ein neues System.Windows.Media.Media3D.Point4DConverter-Objekt	WPK
New-Point4DValue-Serializer	Erzeugt ein neues System.Windows.Media.Media3D.Converters.Point4DValue-Serializer-Objekt	WPK
New-PointAnimation	Erzeugt ein neues System.Windows.Media.Animation.PointAnimation-Objekt	WPK
New-PointAnimation-UsingKeyFrames	Erzeugt ein neues System.Windows.Media.Animation.PointAnimationUsingKey-Frames-Objekt	WPK
New-PointAnimation-UsingPath	Erzeugt ein neues System.Windows.Media.Animation.PointAnimationUsingPath-Objekt	WPK
New-PointCollection	Erzeugt ein neues System.Windows.Media.PointCollection-Objekt	WPK
New-PointCollection-Converter	Erzeugt ein neues System.Windows.Media.PointCollectionConverter-Objekt	WPK
New-PointCollection-ValueSerializer	Erzeugt ein neues System.Windows.Media.Converters.PointCollectionValueSerializer-Objekt	WPK
New-PointConverter	Erzeugt ein neues System.Windows.Point-Converter-Objekt	WPK
New-PointIListConverter	Erzeugt ein neues System.Windows.Media.Converters.PointIListConverter-Objekt	WPK
New-PointKeyFrame-Collection	Erzeugt ein neues System.Windows.Media.Animation.PointKeyFrameCollection-Objekt	WPK

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
New-PointLight	Erzeugt ein neues System.Windows.Media.Media3D.PointLight	WPK
New-PointValueSerializer	Erzeugt ein neues System.Windows.Converters.PointValueSerializer-Objekt	WPK
New-PolyBezierSegment	Erzeugt ein neues System.Windows.Media.PolyBezierSegment-Objekt	WPK
New-Polygon	Erzeugt ein neues System.Windows.Shapes.Polygon	WPK
New-Polyline	Erzeugt ein neues System.Windows.Shapes.Polyline	WPK
New-PolyLineSegment	Erzeugt ein neues System.Windows.Media.PolyLineSegment	WPK
New-PolyQuadratic-BezierSegment	Erzeugt ein neues System.Windows.Media.PolyQuadraticBezierSegment-Objekt	WPK
New-Popup	Erzeugt ein neues System.Windows.Controls.Primitives.Popup-Objekt	WPK
New-PrintDialog	Erzeugt ein neues System.Windows.Controls.PrintDialog-Objekt	WPK
New-PrintDialogException	Erzeugt ein neues System.Windows.Controls.PrintDialogException-Objekt	WPK
New-PrintPreviewControl	Erzeugt ein neues System.Windows.Forms.PrintPreviewControl-Objekt	WPK
New-PrintPreviewDialog	Erzeugt ein neues System.Windows.Forms.PrintPreviewDialog-Objekt	WPK
New-PriorityBinding	Erzeugt ein neues System.Windows.Data.PriorityBinding-Objekt	WPK
New-ProgressBar	Erzeugt ein neues System.Windows.Controls.ProgressBar-Objekt	WPK
New-PropertyGrid	Erzeugt ein neues System.Windows.Forms.PropertyGrid-Objekt	WPK
New-PropertyGroup-Description	Erzeugt ein neues System.Windows.Data.PropertyGroupDescription-Objekt	WPK
New-PropertyMap	Erzeugt ein neues System.Windows.Forms.Integration.PropertyMap-Objekt	WPK
New-PropertyMetadata	Erzeugt ein neues System.Windows.PropertyMetadata-Objekt	WPK
New-PropertyPath-Converter	Erzeugt ein neues System.Windows.PropertyPathConverter-Objekt	WPK
New-PSDrive	Erstellt in der aktuellen Sitzung ein Windows PowerShell-Laufwerk.	Microsoft.Power-Shell.Management

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
New-PSSession	Erstellt eine dauerhafte Verbindung mit einem lokalen Computer oder einem Remotecomputer.	Microsoft.PowerShell.Core
New-PSSessionOption	Erstellt ein Objekt, das erweiterte Optionen für eine PSSession enthält.	Microsoft.PowerShell.Core
New-QuadraticBezierSegment	Erzeugt ein neues System.Windows.Media.QuadraticBezierSegment-Objekt	WPK
New-QuaternionAnimation	Erzeugt ein neues System.Windows.Media.Animation.QuaternionAnimation-Objekt	WPK
New-QuaternionAnimationUsingKeyFrames	Erzeugt ein neues System.Windows.Media.Animation.QuaternionAnimationUsingKeyFrames-Objekt	WPK
New-QuaternionConverter	Erzeugt ein neues System.Windows.Media.Media3D.QuaternionConverter-Objekt	WPK
New-QuaternionKeyFrameCollection	Erzeugt ein neues System.Windows.Media.Animation.QuaternionKeyFrameCollection-Objekt	WPK
New-QuaternionRotation3D	Erzeugt ein neues System.Windows.Media.Media3D.QuaternionRotation3D-Objekt	WPK
New-QuaternionValueSerializer	Erzeugt ein neues System.Windows.Media.Media3D.Converters.QuaternionValueSerializer-Objekt	WPK
New-RadialGradientBrush	Erzeugt ein neues System.Windows.Media.RadialGradientBrush-Objekt	WPK
New-RadioButton	Erzeugt ein neues System.Windows.Controls.RadioButton-Objekt	WPK
New-Rect3DConverter	Erzeugt ein neues System.Windows.Media.Media3D.Rect3DConverter-Objekt	WPK
New-Rect3DValueSerializer	Erzeugt ein neues System.Windows.Media.Media3D.Converters.Rect3DValueSerializer-Objekt	WPK
New-Rectangle	Erzeugt ein neues System.Windows.Shapes.Rectangle-Objekt	WPK
New-RectangleGeometry	Erzeugt ein neues System.Windows.Media.RectangleGeometry-Objekt	WPK
New-RectAnimation	Erzeugt ein neues System.Windows.Media.Animation.RectAnimation-Objekt	WPK
New-RectAnimationUsingKeyFrames	Erzeugt ein neues System.Windows.Media.Animation.RectAnimationUsingKeyFrames-Objekt	WPK
New-RectConverter	Erzeugt ein neues System.Windows.RectConverter-Objekt	WPK

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
New-RectKeyFrame-Collection	Erzeugt ein neues System.Windows.Media.Animation.RectKeyFrameCollection-Objekt	WPK
New-RectValueSerializer	Erzeugt ein neues System.Windows.Converters.RectValueSerializer-Objekt	WPK
New-RelativeSource	Erzeugt ein neues System.Windows.Data.RelativeSource-Objekt	WPK
New-RemoveStoryboard	Erzeugt ein neues System.Windows.Media.Animation.RemoveStoryboard-Objekt	WPK
New-RepeatBehavior-Converter	Erzeugt ein neues System.Windows.Media.Animation.RepeatBehaviorConverter-Objekt	WPK
New-RepeatButton	Erzeugt ein neues System.Windows.Controls.Primitives.RepeatButton-Objekt	WPK
New-RequestCachePolicy-Converter	Erzeugt ein neues System.Windows.Media.RequestCachePolicyConverter-Objekt	WPK
New-ResizeGrip	Erzeugt ein neues System.Windows.Controls.Primitives.ResizeGrip-Objekt	WPK
New-ResourceDictionary	Erzeugt ein neues System.Windows.ResourceDictionary-Objekt	WPK
New-ResourceReference-ExpressionConverter	Erzeugt ein neues System.Windows.Markup.ResourceReferenceExpression-Converter-Objekt	WPK
New-ResourceReference-KeyNotFoundException	Erzeugt ein neues System.Windows.ResourceReferenceKeyNotFoundException-Objekt	WPK
New-ResumeStoryboard	Erzeugt ein neues System.Windows.Media.Animation.ResumeStoryboard-Objekt	WPK
New-RichTextBox	Erzeugt ein neues System.Windows.Controls.RichTextBox-Objekt	WPK
New-RightsManagement-Exception	Erzeugt ein neues System.Security.Rights-Management.RightsManagement-Exception-Objekt	WPK
New-RotateTransform	Erzeugt ein neues System.Windows.Media.RotateTransform-Objekt	WPK
New-RotateTransform3D	Erzeugt ein neues System.Windows.Media.Media3D.RotateTransform3D-Objekt	WPK
New-Rotation3D-Animation	Erzeugt ein neues System.Windows.Media.Animation.Rotation3DAnimation-Objekt	WPK
New-Rotation3D-AnimationUsingKey-Frames	Erzeugt ein neues System.Windows.Media.Animation.Rotation3DAnimationUsing-KeyFrames-Objekt	WPK

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
New-Rotation3DKey-FrameCollection	Erzeugt ein neues System.Windows.Media.Animation.Rotation3DKeyFrameCollection-Objekt	WPK
New-RoutedCommand	Erzeugt ein neues System.Windows.Input.RoutedCommand-Objekt	WPK
New-RoutedEventArgs	Erzeugt ein neues System.Windows.RoutedEventArgs-Objekt	WPK
New-RoutedEvent-Converter	Erzeugt ein neues System.Windows.Markup.RoutedEventConverter-Objekt	WPK
New-RoutedUICommand	Erzeugt ein neues System.Windows.Input.RoutedUICommand-Objekt	WPK
New-RowDefinition	Erzeugt ein neues System.Windows.Controls.RowDefinition-Objekt	WPK
New-Run	Erzeugt ein neues System.Windows.Documents.Run-Objekt	WPK
New-SaveFileDialog	Erzeugt ein neues Microsoft.Win32.Save-FileDialog-Objekt	WPK
New-ScaleTransform	Erzeugt ein neues System.Windows.Media.ScaleTransform-Objekt	WPK
New-ScaleTransform3D	Erzeugt ein neues System.Windows.Media.Media3D.ScaleTransform3D-Objekt	WPK
New-ScriptCmdlet	Erzeugt ein neues script cmdlet-Objekt	PSCodeGen
New-ScriptModuleFrom-CurrentLocation	Erzeugt ein neues basic script module (.PSM1) from location of the current file	PowerShellPack
New-ScrollableControl	Erzeugt ein neues System.Windows.Forms.ScrollableControl-Objekt	WPK
New-ScrollBar	Erzeugt ein neues System.Windows.Controls.Primitives.ScrollBar-Objekt	WPK
New-ScrollContent-Presenter	Erzeugt ein neues System.Windows.Controls.ScrollContentPresenter-Objekt	WPK
New-ScrollViewer	Erzeugt ein neues System.Windows.Controls.ScrollViewer-Objekt	WPK
New-Section	Erzeugt ein neues System.Windows.Documents.Section-Objekt	WPK
New-SectionStructure	Erzeugt ein neues System.Windows.Documents.DocumentStructures.Section-Structure-Objekt	WPK
New-SeekStoryboard	Erzeugt ein neues System.Windows.Media.Animation.SeekStoryboard-Objekt	WPK
New-Separator	Erzeugt ein neues System.Windows.Controls.Separator-Objekt	WPK

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
New-SerializerProvider	Erzeugt ein neues System.Windows.Documents.Serialization.SerializerProvider-Objekt	WPK
New-Service	Erstellt einen neuen Windows-Dienst.	Microsoft.PowerShell.Management
New-ServiceProviders	Erzeugt ein neues System.Windows.Markup.ServiceProviders-Objekt	WPK
New-SetStoryboardSpeedRatio	Erzeugt ein neues System.Windows.Media.Animation.SetStoryboardSpeedRatio-Objekt	WPK
New-Setter	Erzeugt ein neues System.Windows.Setter-Objekt	WPK
New-SetterBaseCollection	Erzeugt ein neues System.Windows.SetterBaseCollection-Objekt	WPK
New-SingleAnimation	Erzeugt ein neues System.Windows.Media.Animation.SingleAnimation-Objekt	WPK
New-SingleAnimationUsingKeyFrames	Erzeugt ein neues System.Windows.Media.Animation.SingleAnimationUsingKeyFrames-Objekt	WPK
New-SingleKeyFrameCollection	Erzeugt ein neues System.Windows.Media.Animation.SingleKeyFrameCollection-Objekt	WPK
New-Size3DConverter	Erzeugt ein neues System.Windows.Media.Media3D.Size3DConverter-Objekt	WPK
New-Size3DValueSerializer	Erzeugt ein neues System.Windows.Media.Media3D.Converters.Size3DValueSerializer-Objekt	WPK
New-SizeAnimation	Erzeugt ein neues System.Windows.Media.Animation.SizeAnimation-Objekt	WPK
New-SizeAnimationUsingKeyFrames	Erzeugt ein neues System.Windows.Media.Animation.SizeAnimationUsingKeyFrames-Objekt	WPK
New-SizeConverter	Erzeugt ein neues System.Windows.SizeConverter-Objekt	WPK
New-SizeKeyFrameCollection	Erzeugt ein neues System.Windows.Media.Animation.SizeKeyFrameCollection-Objekt	WPK
New-SizeValueSerializer	Erzeugt ein neues System.Windows.Converters.SizeValueSerializer-Objekt	WPK
New-SkewTransform	Erzeugt ein neues System.Windows.Media.SkewTransform-Objekt	WPK
New-SkipStoryboardToFill	Erzeugt ein neues System.Windows.Media.Animation.SkipStoryboardToFill-Objekt	WPK

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
New-Slider	Erzeugt ein neues System.Windows.Controls.Slider-Objekt	WPK
New-SolidColorBrush	Erzeugt ein neues System.Windows.Media.SolidColorBrush-Objekt	WPK
New-SortDescription-Collection	Erzeugt ein neues System.Component-Model.SortDescriptionCollection-Objekt	WPK
New-SoundPlayerAction	Erzeugt ein neues System.Windows.Controls.SoundPlayerAction-Objekt	WPK
New-Span	Erzeugt ein neues System.Windows.Documents.Span-Objekt	WPK
New-SpecularMaterial	Erzeugt ein neues System.Windows.Media.Media3D.SpecularMaterial-Objekt	WPK
New-SplineByteKeyFrame	Erzeugt ein neues System.Windows.Media.Animation.SplineByteKeyFrame-Objekt	WPK
New-SplineColorKeyFrame	Erzeugt ein neues System.Windows.Media.Animation.SplineColorKeyFrame-Objekt	WPK
New-SplineDecimalKey-Frame	Erzeugt ein neues System.Windows.Media.Animation.SplineDecimalKeyFrame-Objekt	WPK
New-SplineDoubleKey-Frame	Erzeugt ein neues System.Windows.Media.Animation.SplineDoubleKeyFrame-Objekt	WPK
New-SplineInt16KeyFrame	Erzeugt ein neues System.Windows.Media.Animation.SplineInt16KeyFrame-Objekt	WPK
New-SplineInt32KeyFrame	Erzeugt ein neues System.Windows.Media.Animation.SplineInt32KeyFrame-Objekt	WPK
New-SplineInt64KeyFrame	Erzeugt ein neues System.Windows.Media.Animation.SplineInt64KeyFrame-Objekt	WPK
New-SplinePoint3D-KeyFrame	Erzeugt ein neues System.Windows.Media.Animation.SplinePoint3DKeyFrame-Objekt	WPK
New-SplinePointKeyFrame	Erzeugt ein neues System.Windows.Media.Animation.SplinePointKeyFrame-Objekt	WPK
New-SplineQuaternion-KeyFrame	Erzeugt ein neues System.Windows.Media.Animation.SplineQuaternionKeyFrame-Objekt	WPK
New-SplineRectKeyFrame	Erzeugt ein neues System.Windows.Media.Animation.SplineRectKeyFrame-Objekt	WPK
New-SplineRotation3D-KeyFrame	Erzeugt ein neues System.Windows.Media.Animation.SplineRotation3DKeyFrame-Objekt	WPK
New-SplineSingleKey-Frame	Erzeugt ein neues System.Windows.Media.Animation.SplineSingleKeyFrame-Objekt	WPK

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
New-SplineSizeKeyFrame	Erzeugt ein neues System.Windows.Media.Animation.SplineSizeKeyFrame-Objekt	WPK
New-SplineThicknessKeyFrame	Erzeugt ein neues System.Windows.Media.Animation.SplineThicknessKeyFrame-Objekt	WPK
New-SplineVector3DKeyFrame	Erzeugt ein neues System.Windows.Media.Animation.SplineVector3DKeyFrame-Objekt	WPK
New-SplineVectorKeyFrame	Erzeugt ein neues System.Windows.Media.Animation.SplineVectorKeyFrame-Objekt	WPK
New-SplitContainer	Erzeugt ein neues System.Windows.Forms.plitContainer-Objekt	WPK
New-Splitter	Erzeugt ein neues System.Windows.Forms.Splitter-Objekt	WPK
New-SpotLight	Erzeugt ein neues System.Windows.Media.Media3D.SpotLight-Objekt	WPK
New-StackPanel	Erzeugt ein neues System.Windows.Controls.StackPanel-Objekt	WPK
New-StaticExtension	Erzeugt ein neues System.Windows.Markup.StaticExtension-Objekt	WPK
New-StaticResourceExtension	Erzeugt ein neues System.Windows.StaticResourceExtension-Objekt	WPK
New-StatusBar	Erzeugt ein neues System.Windows.Controls.Primitives.StatusBar-Objekt	WPK
New-StatusBarItem	Erzeugt ein neues System.Windows.Controls.Primitives.StatusBarItem-Objekt	WPK
New-StatusStrip	Erzeugt ein neues System.Windows.Forms.StatusStrip-Objekt	WPK
New-StopStoryboard	Erzeugt ein neues System.Windows.Media.Animation.StopStoryboard-Objekt	WPK
New-Storyboard	Erzeugt ein neues System.Windows.Media.Animation.Storyboard-Objekt	WPK
New-StoryBreak	Erzeugt ein neues System.Windows.Documents.DocumentStructures.StoryBreak-Objekt	WPK
New-StoryFragment	Erzeugt ein neues System.Windows.Documents.DocumentStructures.StoryFragment-Objekt	WPK
New-StoryFragments	Erzeugt ein neues System.Windows.Documents.DocumentStructures.StoryFragments-Objekt	WPK

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
New-StreamGeometry	Erzeugt ein neues System.Windows.Media.StreamGeometry-Objekt	WPK
New-StreamResourceInfo	Erzeugt ein neues System.Windows.Resources.StreamResourceInfo-Objekt	WPK
New-StringAnimationUsingKeyFrames	Erzeugt ein neues System.Windows.Media.Animation.StringAnimationUsingKeyFrames-Objekt	WPK
New-StringKeyFrameCollection	Erzeugt ein neues System.Windows.Media.Animation.StringKeyFrameCollection-Objekt	WPK
New-StrokeCollection	Erzeugt ein neues System.Windows.Ink.StrokeCollection-Objekt	WPK
New-StrokeCollectionConverter	Erzeugt ein neues System.Windows.StrokeCollectionConverter-Objekt	WPK
New-Style	Erzeugt ein neues System.Windows.Style-Objekt	WPK
New-StyleSelector	Erzeugt ein neues System.Windows.Controls.StyleSelector-Objekt	WPK
New-StyleTypedPropertyAttribute	Erzeugt ein neues System.Windows.StyleTypedPropertyAttribute-Objekt	WPK
New-StylusPointCollection	Erzeugt ein neues System.Windows.Input.StylusPointCollection-Objekt	WPK
New-StylusPointDescription	Erzeugt ein neues System.Windows.Input.StylusPointDescription-Objekt	WPK
New-TabControl	Erzeugt ein neues System.Windows.Controls.TabControl-Objekt	WPK
New-TabItem	Erzeugt ein neues System.Windows.Controls.TabItem-Objekt	WPK
New-Table	Erzeugt ein neues System.Windows.Documents.Table-Objekt	WPK
New-TableCell	Erzeugt ein neues System.Windows.Documents.TableCell-Objekt	WPK
New-TableCellStructure	Erzeugt ein neues System.Windows.Documents.DocumentStructures.TableCellStructure-Objekt	WPK
New-TableColumn	Erzeugt ein neues System.Windows.Documents.TableColumn-Objekt	WPK
New-TableLayoutPanel	Erzeugt ein neues System.Windows.Forms.TableLayoutPanel-Objekt	WPK
New-TableRow	Erzeugt ein neues System.Windows.Documents.TableRow-Objekt	WPK

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
New-TableRowGroup	Erzeugt ein neues System.Windows.Documents.TableRowGroup-Objekt	WPK
New-TableRowGroup-Structure	Erzeugt ein neues System.Windows.Documents.DocumentStructures.TableRowGroupStructure-Objekt	WPK
New-TableRowStructure	Erzeugt ein neues System.Windows.Documents.DocumentStructures.TableRowStructure-Objekt	WPK
New-TableStructure	Erzeugt ein neues System.Windows.Documents.DocumentStructure-Objekts.TableStructure	WPK
New-TabPage	Erzeugt ein neues System.Windows.Forms.TabPage-Objekt	WPK
New-TabPanel	Erzeugt ein neues System.Windows.Controls.Primitives.TabPanel-Objekt	WPK
New-Task	Erzeugt ein neues task definition –Objekt	TaskScheduler
New-TemplateBinding-ExpressionConverter	Erzeugt ein neues System.Windows.TemplateBindingExpressionConverter-Objekt	WPK
New-TemplateBinding-Extension	Erzeugt ein neues System.Windows.TemplateBindingExtension-Objekt	WPK
New-TemplateBinding-ExtensionConverter	Erzeugt ein neues System.Windows.TemplateBindingExtensionConverter-Objekt	WPK
New-TemplateKey-Converter	Erzeugt ein neues System.Windows.Markup.TemplateKeyConverter-Objekt	WPK
New-TemplatePart-Attribute	Erzeugt ein neues System.Windows.TemplatePartAttribute-Objekt	WPK
New-TextBlock	Erzeugt ein neues System.Windows.Controls.TextBlock-Objekt	WPK
New-TextBox	Erzeugt ein neues System.Windows.Controls.TextBox-Objekt	WPK
New-TextDecoration	Erzeugt ein neues System.Windows.Text-Decoration-Objekt	WPK
New-TextDecoration-Collection	Erzeugt ein neues System.Windows.Text-DecorationCollection-Objekt	WPK
New-TextDecoration-CollectionConverter	Erzeugt ein neues System.Windows.Text-DecorationCollectionConverter-Objekt	WPK
New-TextEffect	Erzeugt ein neues System.Windows.Media.TextEffect-Objekt	WPK
New-TextEffectCollection	Erzeugt ein neues System.Windows.Media.TextEffectCollection-Objekt	WPK

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
New-TextElementEditing-BehaviorAttribute	Erzeugt ein neues System.Windows.Documents.TextElementEditingBehavior-Attribute-Objekt	WPK
New-TextRunCache	Erzeugt ein neues System.Windows.Media.TextFormatting.TextRunCache-Objekt	WPK
New-ThemeDictionary-Extension	Erzeugt ein neues System.Windows.Theme-DictionaryExtension-Objekt	WPK
New-ThicknessAnimation	Erzeugt ein neues System.Windows.Media.Animation.ThicknessAnimation-Objekt	WPK
New-ThicknessAnimation-UsingKeyFrames	Erzeugt ein neues System.Windows.Media.Animation.ThicknessAnimationUsingKey-Frames-Objekt	WPK
New-ThicknessConverter	Erzeugt ein neues System.Windows.ThicknessConverter-Objekt	WPK
New-ThicknessKeyFrame-Collection	Erzeugt ein neues System.Windows.Media.Animation.ThicknessKeyFrameCollection-Objekt	WPK
New-Thumb	Erzeugt ein neues System.Windows.Controls.Primitives.Thumb-Objekt	WPK
New-TickBar	Erzeugt ein neues System.Windows.Controls.Primitives.TickBar-Objekt	WPK
New-TiffBitmapEncoder	Erzeugt ein neues System.Windows.Media.Imaging.TiffBitmapEncoder-Objekt	WPK
New-TimelineCollection	Erzeugt ein neues System.Windows.Media.Animation.TimelineCollection-Objekt	WPK
New-TimeSpan	Erstellt ein TimeSpan-Objekt.	Microsoft.Power-Shell.Utility
New-ToggleButton	Erzeugt ein neues System.Windows.Controls.Primitives.ToggleButton-Objekt	WPK
New-ToolBar	Erzeugt ein neues System.Windows.Controls.ToolBar-Objekt	WPK
New-ToolBarOverflow-Panel	Erzeugt ein neues System.Windows.Controls.Primitives.ToolBarOverflowPanel-Objekt	WPK
New-ToolBarPanel	Erzeugt ein neues System.Windows.Controls.Primitives.ToolBarPanel-Objekt	WPK
New-ToolBarTray	Erzeugt ein neues System.Windows.Controls.ToolBarTray-Objekt	WPK
New-ToolStrip	Erzeugt ein neues System.Windows.Forms.ToolStrip-Objekt	WPK

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
New-ToolStripContainer	Erzeugt ein neues System.Windows.Forms.ToolStripContainer-Objekt	WPK
New-ToolStripContentPanel	Erzeugt ein neues System.Windows.Forms.ToolStripContentPanel-Objekt	WPK
New-ToolStripDropDown	Erzeugt ein neues System.Windows.Forms.ToolStripDropDown-Objekt	WPK
New-ToolStripDropDownMenu	Erzeugt ein neues System.Windows.Forms.ToolStripDropDownMenu-Objekt	WPK
New-ToolStripPanel	Erzeugt ein neues System.Windows.Forms.ToolStripPanel-Objekt	WPK
New-ToolTip	Erzeugt ein neues System.Windows.Controls.ToolTip-Objekt	WPK
New-Track	Erzeugt ein neues System.Windows.Controls.Primitives.Track-Objekt	WPK
New-TrackBar	Erzeugt ein neues System.Windows.Forms.TrackBar-Objekt	WPK
New-Transform3DCollection	Erzeugt ein neues System.Windows.Media.Media3D.Transform3DCollection-Objekt	WPK
New-Transform3DGroup	Erzeugt ein neues System.Windows.Media.Media3D.Transform3DGroup-Objekt	WPK
New-TransformCollection	Erzeugt ein neues System.Windows.Media.TransformCollection-Objekt	WPK
New-TransformConverter	Erzeugt ein neues System.Windows.Media.TransformConverter-Objekt	WPK
New-TransformedBitmap	Erzeugt ein neues System.Windows.Media.Imaging.TransformedBitmap-Objekt	WPK
New-TransformGroup	Erzeugt ein neues System.Windows.Media.TransformGroup-Objekt	WPK
New-TransformValueSerializer	Erzeugt ein neues System.Windows.Media.Converters.TransformValueSerializer-Objekt	WPK
New-TranslateTransform	Erzeugt ein neues System.Windows.Media.TranslateTransform-Objekt	WPK
New-TranslateTransform3D	Erzeugt ein neues System.Windows.Media.Media3D.TranslateTransform3D-Objekt	WPK
New-TreeView	Erzeugt ein neues System.Windows.Controls.TreeView-Objekt	WPK
New-TreeViewItem	Erzeugt ein neues System.Windows.Controls.TreeViewItem-Objekt	WPK
New-Trigger	Erzeugt ein neues System.Windows.Trigger-Objekt	WPK

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
New-TriggerAction-Collection	Erzeugt ein neues System.Windows. TriggerActionCollection-Objekt	WPK
New-TrimSurrounding-WhitespaceAttribute	Erzeugt ein neues System.Windows. Markup.TrimSurroundingWhitespace-Attribute-Objekt	WPK
New-TypeExtension	Erzeugt ein neues System.Windows. Markup.TypeExtension-Objekt	WPK
New-UIElement	Erzeugt ein neues System.Windows. UIElement-Objekt	WPK
New-UIPropertyMetadata	Erzeugt ein neues System.Windows. UIPropertyMetadata-Objekt	WPK
New-Underline	Erzeugt ein neues System.Windows. Documents.Underline-Objekt	WPK
New-UniformGrid	Erzeugt ein neues System.Windows. Controls.Primitives.UniformGrid-Objekt	WPK
New-UnsignedPublish-License	Erzeugt ein neues System.Security.Rights-Management.UnsignedPublishLicense-Objekt	WPK
New-UserControl	Erzeugt ein neues System.Windows. Controls.UserControl-Objekt	WPK
New-UShortIListConverter	Erzeugt ein neues System.Windows.Media. Converters.UShortIListConverter-Objekt	WPK
New-ValueUnavailable-Exception	Erzeugt ein neues System.Windows.Data. ValueUnavailableException-Objekt	WPK
New-Variable	Erstellt eine neue Variable.	Microsoft.Power-Shell.Utility
New-Vector3DAnimation	Erzeugt ein neues System.Windows.Media. Animation.Vector3DAnimation-Objekt	WPK
New-Vector3DAnimation-UsingKeyFrames	Erzeugt ein neues System.Windows.Media. Animation.Vector3DAnimationUsingKey-Frames-Objekt	WPK
New-Vector3DCollection	Erzeugt ein neues System.Windows.Media. Media3D.Vector3DCollection-Objekt	WPK
New-Vector3DCollection-Converter	Erzeugt ein neues System.Windows.Media. Media3D.Vector3DCollectionConverter-Objekt	WPK
New-Vector3DCollection-ValueSerializer	Erzeugt ein neues System.Windows.Media. Media3D.Converters.Vector3DCollection-ValueSerializer-Objekt	WPK
New-Vector3DConverter	Erzeugt ein neues System.Windows.Media. Media3D.Vector3DConverter-Objekt	WPK

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
New-Vector3DKeyFrame-Collection	Erzeugt ein neues System.Windows.Media.Animation.Vector3DKeyFrameCollection-Objekt	WPK
New-Vector3DValue-Serializer	Erzeugt ein neues System.Windows.Media.Media3D.Converters.Vector3DValue-Serializer-Objekt	WPK
New-VectorAnimation	Erzeugt ein neues System.Windows.Media.Animation.VectorAnimation-Objekt	WPK
New-VectorAnimation-UsingKeyFrames	Erzeugt ein neues System.Windows.Media.Animation.VectorAnimationUsingKey-Frames-Objekt	WPK
New-VectorCollection	Erzeugt ein neues System.Windows.Media.VectorCollection-Objekt	WPK
New-VectorCollection-Converter	Erzeugt ein neues System.Windows.Media.VectorCollectionConverter-Objekt	WPK
New-VectorCollection-ValueSerializer	Erzeugt ein neues System.Windows.Media.Converters.VectorCollectionValueSerializer-Objekt	WPK
New-VectorConverter	Erzeugt ein neues System.Windows.Vector-Converter-Objekt	WPK
New-VectorKeyFrame-Collection	Erzeugt ein neues System.Windows.Media.Animation.VectorKeyFrameCollection-Objekt	WPK
New-VectorValueSerializer	Erzeugt ein neues System.Windows.Converters.VectorValueSerializer-Objekt	WPK
New-VideoDrawing	Erzeugt ein neues System.Windows.Media.VideoDrawing-Objekt	WPK
New-Viewbox	Erzeugt ein neues System.Windows.Controls.Viewbox-Objekt	WPK
New-Viewport2DVisual3D	Erzeugt ein neues System.Windows.Media.Media3D.Viewport2DVisual3D-Objekt	WPK
New-Viewport3D	Erzeugt ein neues System.Windows.Controls.Viewport3D-Objekt	WPK
New-Viewport3DVisual	Erzeugt ein neues System.Windows.Media.Media3D.Viewport3DVisual-Objekt	WPK
New-VirtualizingStack-Panel	Erzeugt ein neues System.Windows.Controls.VirtualizingStackPanel-Objekt	WPK
New-VisualBrush	Erzeugt ein neues System.Windows.Media.VisualBrush-Objekt	WPK
New-VScrollBar	Erzeugt ein neues System.Windows.Forms.VScrollBar-Objekt	WPK

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
New-WBBackupTarget	Erstellt ein neues WBBackupTarget-Objekt.	Windows.Server-Backup
New-WBFileSpec	Erstellt ein neues WBFileSpec-Objekt.	Windows.Server-Backup
New-WBPolicy	Erstellt eine neue Sicherungsrichtlinie (WBPolicy-Objekt).	Windows.Server-Backup
New-WebApplication	Erstellt eine neue IIS-Webanwendung.	WebAdministration
New-WebAppPool	Erstellt einen neuen IIS-Anwendungspool.	WebAdministration
New-WebBinding	Fügt einer vorhandenen Website eine neue Websitebindung hinzu.	WebAdministration
New-WebBrowser	Erzeugt ein neues System.Windows.Controls.WebBrowser-Objekt	WPK
New-WebBrowser-Permission	Erzeugt ein neues System.Security.Permissions.WebBrowserPermission-Objekt	WPK
New-WebFtpSite	Erstellt eine neue FTP7-Site.	WebAdministration
New-WebGlobalModule	Erstellt ein neues IIS-Modul.	WebAdministration
New-WebHandler	Erstellt einen neuen Konfigurationseintrag für den IIS-Anforderungshandler.	WebAdministration
New-WebManaged-Module	Fügt der IIS-Anforderungspipeline ein neues verwaltetes Modul hinzu.	WebAdministration
New-WebServiceProxy	Erstellt ein Webdienstproxyobjekt, mit dem Sie den Webdienst in Windows PowerShell verwenden und verwalten können.	Microsoft.PowerShell.Management
New-Website	Erstellt eine neue IIS-Website.	WebAdministration
New-WebVirtualDirectory	Erstellt ein neues virtuelles IIS-Verzeichnis.	WebAdministration
New-Whitespace-SignificantCollection-Attribute	Erzeugt ein neues System.Windows.Markup.WhitespaceSignificantCollection-Attribute-Objekt	WPK
New-Window	Erzeugt ein neues System.Windows.Window-Objekt	WPK
New-WindowCollection	Erzeugt ein neues System.Windows.WindowCollection-Objekt	WPK
New-WindowsFormsHost	Erzeugt ein neues System.Windows.Forms.Integration.WindowsFormsHost-Objekt	WPK
New-WmpBitmapEncoder	Erzeugt ein neues System.Windows.Media.Imaging.WmpBitmapEncoder-Objekt	WPK
New-WrapPanel	Erzeugt ein neues System.Windows.Controls.WrapPanel-Objekt	WPK

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
New-WSManInstance	Erstellt eine neue Instanz einer Verwaltungsressource.	Microsoft.WSMan.Management
New-WSManSessionOption	Erstellt eine Hashtabelle für die WS-Verwaltungssitzungsoption, die als Eingabeparameter für die folgenden Cmdlets der WS-Verwaltung verwendet wird: Get-WSManInstance Set-WSManInstance Invoke-WSManAction Connect-WSMan	Microsoft.WSMan.Management
New-XamlParseException	Erzeugt ein neues System.Windows.Markup.XamlParseException-Objekt	WPK
New-XamlReader	Erzeugt ein neues System.Windows.Markup.XamlReader-Objekt	WPK
New-XmlDataProvider	Erzeugt ein neues System.Windows.Data.XmlDataProvider-Objekt	WPK
New-XmlLanguageConverter	Erzeugt ein neues System.Windows.Markup.XmlLanguageConverter-Objekt	WPK
New-XmlNamespaceMapping	Erzeugt ein neues System.Windows.Data.XmlNamespaceMapping-Objekt	WPK
New-XmlNamespaceMappingCollection	Erzeugt ein neues System.Windows.Data.XmlNamespaceMappingCollection-Objekt	WPK
New-XmlnsDictionary	Erzeugt ein neues System.Windows.Markup.XmlnsDictionary-Objekt	WPK
New-Zip	Erzeugt ein neues Zip-Archiv.	PowerShellPack
New-ZoomPercentageConverter	Erzeugt ein neues System.Windows.Documents.ZoomPercentageConverter-Objekt	WPK
Out-Default	Sendet die Ausgabe an das Standardformatierungsprogramm und das standardmäßige Ausgabe-Cmdlet.	Microsoft.PowerShell.Utility
Out-File	Sendet die Ausgabe an eine Datei.	Microsoft.PowerShell.Utility
Out-GridView	Sendet die Ausgabe an eine interaktive Tabelle in einem eigenen Fenster.	Microsoft.PowerShell.Utility
Out-Host	Sendet die Ausgabe an die Befehlszeile.	Microsoft.PowerShell.Utility
Out-Null	Löscht die Ausgabe, statt sie an die Konsole zu senden.	Microsoft.PowerShell.Utility
Out-Printer	Sendet die Ausgabe an einen Drucker.	Microsoft.PowerShell.Utility

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
Out-String	Sendet Objekte als eine Reihe von Zeichenfolgen an den Host.	Microsoft.PowerShell.Utility
Pop-Location	Ändert den aktuellen Speicherort in den Speicherort, der zuletzt auf den Stapel verschoben wurde. Sie können den Speicherort vom Standardstapel oder von einem Stapel abrufen, den Sie mit dem Cmdlet "Push-Location" erstellen.	Microsoft.PowerShell.Management
Push-Location	Fügt den aktuellen Speicherort an oberster Stelle in der Liste von Speicherorten (einem "Stapel") hinzu.	Microsoft.PowerShell.Management
Read-Article	Markiert Artikel als gelesen.	PSRss
Read-Host	Liest eine Eingabezeile aus der Konsole.	Microsoft.PowerShell.Utility
Receive-Job	Ruft die Ergebnisse der Windows PowerShell-Hintergrundaufträge in der aktuellen Sitzung ab.	Microsoft.PowerShell.Core
Receive-SmigServerData	Ermöglicht es einem Zielserver, Freigaben, Ordner, Dateien und zugeordnete Berechtigungen sowie Freigabeeigenschaften zu empfangen, die von einem Quellserver migriert werden. Der Quellserver muss sich im gleichen Subnetz wie der Zielserver befinden, und das Cmdlet "Send-SmigServerData" muss auf dem Quellserver zur selben Zeit ausgeführt werden, zu der "Receive-SmigServerData" auf dem Zielserver ausgeführt wird.	Microsoft.Windows.ServerManager.Migration
Register-EngineEvent	Abonniert Ereignisse, die vom Windows PowerShell-Modul und dem Cmdlet "New-Event" generiert werden.	Microsoft.PowerShell.Utility
Register-ObjectEvent	Abonniert die Ereignisse, die von einem Microsoft .NET Framework-Objekt generiert werden.	Microsoft.PowerShell.Utility
Register-PowerShellCommand	Registriert einen PowerShell-Scriptblock.	WPK
Register-PSSessionConfiguration	Erstellt und registriert eine neue Sitzungskonfiguration.	Microsoft.PowerShell.Core
Register-ScheduledTask	Erzeugt einen geplanten Vorgang.	TaskScheduler
Register-WmiEvent	Abonniert ein WMI-Ereignis (Windows Management Instrumentation, Windows-Verwaltungsinstrumentation).	Microsoft.PowerShell.Management
Remove-ADComputer	Entfernt einen Active Directory-Computer.	ActiveDirectory

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
Remove-ADComputer-ServiceAccount	Entfernt mindestens ein Dienstkonto von einem Computer.	ActiveDirectory
Remove-ADDomain-ControllerPassword-ReplicationPolicy	Entfernt Benutzer, Computer und Gruppen aus der Zulassungs- oder Verweigerungsliste einer Kennwortreplikationsrichtlinie für schreibgeschützte Domänencontroller.	ActiveDirectory
Remove-ADFineGrained-PasswordPolicy	Entfernt eine abgestimmte ActiveDirectory-Kennwortrichtlinie.	ActiveDirectory
Remove-ADFineGrained-PasswordPolicySubject	Entfernt mindestens einen Benutzer aus einer abgestimmten Kennwortrichtlinie.	ActiveDirectory
Remove-ADGroup	Entfernt eine Active Directory-Gruppe.	ActiveDirectory
Remove-ADGroupMember	Entfernt mindestens ein Mitglied aus einer Active Directory-Gruppe.	ActiveDirectory
Remove-ADObject	Entfernt ein Active Directory-Objekt.	ActiveDirectory
Remove-ADOrganizatio-nalUnit	Entfernt eine Active Directory-Organisationseinheit.	ActiveDirectory
Remove-ADPrincipal-GroupMembership	Entfernt ein Mitglied aus mindestens einer Active Directory-Gruppe.	ActiveDirectory
Remove-ADService-Account	Entfernt ein Active Directory-Dienstkonto.	ActiveDirectory
Remove-ADUser	Entfernt einen Active Directory-Benutzer.	ActiveDirectory
Remove-Article	Entfernt RSS-Artikel	PSRss
Remove-BitsTransfer	Bricht einen Übertragungsauftrag des intelligenten Hintergrundübertragungsdiensts (BITS) ab.	BitsTransfer
Remove-ChildControl	Entfernt ein Kindelement.	WPK
Remove-Cluster	Zerstören eines vorhandenen Failoverclusters. Die betroffenen Server arbeiten nicht mehr als Cluster zusammen.	FailoverClusters
Remove-ClusterAccess	Entfernt einen Benutzer aus der Zugriffsliste für den Cluster.	FailoverClusters
Remove-ClusterGroup	Entfernen eines geclusterten Diensts oder einer Clusteranwendung (auch als Ressourcengruppe bezeichnet) von einem Failovercluster	FailoverClusters
Remove-ClusterNode	Entfernen eines Knoten aus einem Failovercluster. Ein entfernter Knoten ist nicht länger Teil des Clusters, es sei denn, er wird dem Cluster erneut hinzugefügt.	FailoverClusters
Remove-ClusterResource	Entfernen einer Clusterressource aus dem Failovercluster	FailoverClusters

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
Remove-ClusterResource-Dependency	Entfernen einer Abhängigkeit zwischen zwei Ressourcen in einem geclusterten Dienst oder einer Clusteranwendung innerhalb eines Failoverclusters	FailoverClusters
Remove-ClusterResource-Type	Entfernen eines Ressourcentyps aus einem Failovercluster	FailoverClusters
Remove-ClusterShared-Volume	Entfernen eines Volumes aus den freigegebenen Clustervolumes in einem Failovercluster und Verschieben des Volumes in den "Available Storage" des Clusters	FailoverClusters
Remove-Computer	Entfernt den lokalen Computer aus einer Arbeitsgruppe oder einer Domäne.	Microsoft.PowerShell.Management
Remove-DirectoryEntry	Entfernt einen Verzeichniseintrag.	ITVisions_PowerShell_Extensions
Remove-Event	Löscht Ereignisse aus der Ereigniswarteschlange.	Microsoft.PowerShell.Utility
Remove-EventLog	Löscht ein Ereignisprotokoll oder hebt die Registrierung einer Ereignisquelle auf.	Microsoft.PowerShell.Management
Remove-Feed	Enfernt einen RSS-Feed.	PSRss
Remove-GPLink	Entfernt eine Gruppenrichtlinienobjekt-Verknüpfung aus einer Site, Domäne oder Organisationseinheit.	GroupPolicy
Remove-GPO	Löscht ein Gruppenrichtlinienobjekt.	GroupPolicy
Remove-GPPrefRegistry-Value	Entfernt eines oder mehrere Registrierungseinstellungselemente aus der Computerkonfiguration oder der Benutzerkonfiguration in einem Gruppenrichtlinienobjekt.	GroupPolicy
Remove-GPRegistryValue	Entfernt eine oder mehrere registrierungsbasierte Richtlinieneinstellungen aus der Computerkonfiguration oder der Benutzerkonfiguration in einem Gruppenrichtlinienobjekt.	GroupPolicy
Remove-Item	Löscht die angegebenen Elemente.	Microsoft.PowerShell.Management
Remove-ItemProperty	Löscht die Eigenschaft und deren Wert aus einem Element.	Microsoft.PowerShell.Management
Remove-Job	Löscht einen Windows PowerShell-Hintergrundauftrag.	Microsoft.PowerShell.Core
Remove-Module	Entfernt Module aus der aktuellen Sitzung.	Microsoft.PowerShell.Core

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
Remove-NlbCluster	Löscht einen NLB-Cluster.	NetworkLoad-BalancingClusters
Remove-NlbClusterNode	Entfernt einen Knoten aus dem NLB-Cluster.	NetworkLoad-BalancingClusters
Remove-NlbClusterNode-Dip	Entfernt eine dedizierte IP-Adresse aus einem NLB-Cluster.	NetworkLoad-BalancingClusters
Remove-NlbClusterPort-Rule	Entfernt eine Portregel aus einem NLB-Cluster.	NetworkLoad-BalancingClusters
Remove-NlbClusterVip	Entfernt eine virtuelle IP-Adresse aus einem NLB-Cluster.	NetworkLoad-BalancingClusters
Remove-PSBreakpoint	Löscht Haltepunkte aus der aktuellen Konsole.	Microsoft.Power-Shell.Utility
Remove-PSDrive	Entfernt ein Windows PowerShell-Laufwerk von seinem Speicherort.	Microsoft.Power-Shell.Management
Remove-PSSession	Schließt eine oder mehrere Windows PowerShell-Sitzungen (PSSessions).	Microsoft.Power-Shell.Core
Remove-PSSnapin	Entfernt Windows PowerShell-Snap-Ins aus der aktuellen Sitzung.	Microsoft.Power-Shell.Core
Remove-Task	Entfernt eine reguläre Aufgabe	TaskScheduler
Remove-Variable	Löscht eine Variable und ihren Wert.	Microsoft.Power-Shell.Utility
Remove-WBBackupTarget	Entfernt die (vom WBBackupTarget-Objekt definierten) Sicherungsspeicherorte aus der Sicherungsrichtlinie (WBPolicy-Objekt).	Windows.Server-Backup
Remove-WBBareMetal-Recovery	Entfernt die Anforderung, alle für eine Bare-Metal-Recovery mit der aktuellen Sicherungsrichtlinie (WBPolicy-Objekt) benötigten Elemente einzubeziehen.	Windows.Server-Backup
Remove-WBFileSpec	Entfernt die Liste der in eine Sicherung einzuschließenden oder von einer solchen auszuschließenden Elemente, wie vom WBFileSpec-Objekt angegeben, aus einer Sicherungsrichtlinie (WBPolicy-Objekt).	Windows.Server-Backup
Remove-WBPolicy	Entfernt die derzeit festgelegte Sicherungs-richtlinie (WBPolicy-Objekt).	Windows.Server-Backup
Remove-WBSystemState	Entfernt die Anforderung, alle für eine Systemstatuswiederherstellung mit der aktuellen Sicherungsrichtlinie (WBPolicy-Objekt) benötigten Elemente einzubeziehen.	Windows.Server-Backup
Remove-WBVolume	Entfernt das (vom WBPolicy-Objekt angegebene) zu sichernde Volume aus der Sicherungsrichtlinie (WBVolume-Objekt).	Windows.Server-Backup

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
Remove-WebApplication	Entfernt eine IIS-Webanwendung.	WebAdministration
Remove-WebAppPool	Entfernt einen IIS-Anwendungspool.	WebAdministration
Remove-WebBinding	Entfernt eine Bindung aus einer IIS-Website.	WebAdministration
Remove-WebConfigurationBackup	Entfernt eine verfügbare IIS-Konfigurationssicherung.	WebAdministration
Remove-WebConfigurationLocation	Entfernt einen IIS-Konfigurationsspeicherort.	WebAdministration
Remove-WebConfigurationLock	Entfernt eine Sperre von Konfigurationseinstellungen.	WebAdministration
Remove-WebConfigurationProperty	Entfernt eine IIS-Konfigurationseigenschaft.	WebAdministration
Remove-WebGlobalModule	Entfernt ein IIS-Modul.	WebAdministration
Remove-WebHandler	Entfernt einen IIS-Anforderungshandler.	WebAdministration
Remove-WebManagedModule	Entfernt ein verwaltetes IIS-Modul.	WebAdministration
Remove-Website	Entfernt eine IIS-Website.	WebAdministration
Remove-WebVirtualDirectory	Entfernt ein virtuelles IIS-Verzeichnis.	WebAdministration
Remove-WindowsFeature	Mit dem Cmdlet "Remove-WindowsFeature" können Sie angegebene Rollen, Rollendienste und Features auf Computern unter Windows Server 2008 R2 entfernen.	ServerManager
Remove-WmiObject	Löscht eine Instanz einer vorhandenen WMI-Klasse (Windows Management Instrumentation, Windows-Verwaltungsinstrumentation).	Microsoft.PowerShell.Management
Remove-WSManInstance	Löscht eine Verwaltungsressourceninstanz.	Microsoft.WSMan.Management
Rename-ADObject	Ändert den Namen eines Active Directory-Objekts.	ActiveDirectory
Rename-Drive	Umbenennen eines Laufwerks.	PowerShellPack
Rename-GPO	Weist einem Gruppenrichtlinienobjekt einen neuen Anzeigenamen zu.	GroupPolicy
Rename-Item	Benennt ein Element in einem Windows PowerShell-Anbieternamespace um.	Microsoft.PowerShell.Management
Rename-ItemProperty	Benennt die Eigenschaft eines Elements um.	Microsoft.PowerShell.Management

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
Rename-Web-ConfigurationLocation	Benennt einen Konfigurationsspeicherort um.	WebAdministration
Repair-ClusterShared-Volume	Ausführen von Reparaturtools auf einem lokalen freigegebenen Clustervolume in einem Clusterknoten	FailoverClusters
Reset-ADServiceAccount-Password	Setzt das Kennwort für das Dienstkonto eines Computers zurück.	ActiveDirectory
Reset-ComputerMachine-Password	Setzt das Kennwort für das Computerkonto des Computers zurück.	Microsoft.Power-Shell.Management
Resolve-Path	Löst die Platzhalterzeichen in einem Pfad auf und zeigt den Inhalt des Pfads an.	Microsoft.Power-Shell.Management
Restart-Computer	Startet das Betriebssystem auf dem lokalen und Remotecomputern neu.	Microsoft.Power-Shell.Management
Restart-Service	Beendet einen oder mehrere Dienste und startet diese anschließend.	Microsoft.Power-Shell.Management
Restart-WebAppPool	Verwendet einen Anwendungspool wieder.	WebAdministration
Restart-WebItem	Startet einen Anwendungspool oder eine Website neu.	WebAdministration
Restore-ADObject	Stellt ein Active Directory-Objekt wieder her.	ActiveDirectory
Restore-Computer	Startet eine Systemwiederherstellung auf dem lokalen Computer.	Microsoft.Power-Shell.Management
Restore-GPO	Stellt ein Gruppenrichtlinienobjekt oder alle Gruppenrichtlinienobjekte in einer Domäne aus einer oder mehreren Gruppenrichtlinien-objekt-Sicherungsdateien wieder her.	GroupPolicy
Restore-WebConfiguration	Stellt eine IIS-Konfigurationssicherung wieder her.	WebAdministration
Resume-BitsTransfer	Setzt einen Übertragungsauftrag des intelligenten Hintergrundübertragungsdiensts (BITS) fort.	BitsTransfer
Resume-ClusterNode	Wiederaufnehmen der Aktivität auf einem Failoverclusterknoten, nachdem der Knoten angehalten (die Aktivität unterbrochen) wurde	FailoverClusters
Resume-ClusterResource	Deaktiviert die Wartung für eine Datenträgerressource oder ein freigegebenes Clustervolume in einem Failovercluster.	FailoverClusters
Resume-NlbCluster	Setzt alle Knoten eines NLB-Clusters fort.	NetworkLoad-BalancingClusters
Resume-NlbClusterNode	Setzt einen Knoten in einem NLB-Cluster fort.	NetworkLoad-BalancingClusters

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
Resume-Service	Setzt einen oder mehrere angehaltene (unterbrochene) Dienste fort.	Microsoft.PowerShell.Management
Save-IseFileWithAutoName	Sichert eine Datei innerhalb einer integrierten Scripting-Umgebung unter einem automatisch generierten Namen	PowerShellPack
Search-ADAccount	Ruft Active Directory-Benutzer-, -Computer- oder -Dienstkonten ab.	ActiveDirectory
Search-WindowsDesktop	Sucht auf dem Windows-System nach Objekten.	DesktopSearch
Select-AllInFile	Selektiert innerhalb der ISE den gesamten Text.	PowerShellPack
Select-CurrentText	Liefert den aktuell ausgewählten Text.	PowerShellPack
Select-Object	Wählt angegebene Eigenschaften eines Objekts oder eines Satzes von Objekten aus. Er kann auch eindeutige Objekte aus einem Array von Objekten oder eine angegebene Anzahl von Objekten vom Anfang oder Ende eines Arrays von Objekten auswählen.	Microsoft.PowerShell.Utility
Select-String	Sucht Text in Zeichenfolgen und Dateien.	Microsoft.PowerShell.Utility
Select-WebConfiguration	Gibt Webkonfigurationsobjekte zurück.	WebAdministration
Select-Xml	Sucht Text in einer XML-Zeichenfolge oder einem XML-Dokument.	Microsoft.PowerShell.Utility
Send-MailMessage	Sendet eine E-Mail.	Microsoft.PowerShell.Utility
Send-SmigServerData	Migriert Ordner, Dateien und zugeordnete Berechtigungen sowie Freigabeeigenschaften über Port 7000 von einem Quellserver zu einem Zielserver. Der Zielserver muss sich im gleichen Subnetz wie der Quellserver befinden, und das Cmdlet "Receive-SmigServerData" muss auf dem Zielserver zur selben Zeit ausgeführt werden, zu der "Send-SmigServerData" auf dem Quellserver ausgeführt wird.	Microsoft.Windows.ServerManager.Migration
Set-Acl	Ändert die Sicherheitsbeschreibung einer angegebenen Ressource, z. B. einer Datei oder eines Registrierungsschlüssels.	Microsoft.PowerShell.Security
Set-ADAccountControl	Ändert Benutzerkontensteuerungs-Werte (User Account Control, UAC) für ein Active Directory-Konto.	ActiveDirectory

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
Set-ADAccountExpiration	Legt das Ablaufdatum für ein Active Directory-Konto fest.	ActiveDirectory
Set-ADAccountPassword	Ändert das Kennwort eines Active Directory-Kontos.	ActiveDirectory
Set-ADComputer	Ändert ein Active Directory-Computerobjekt.	ActiveDirectory
Set-ADDefaultDomainPasswordPolicy	Ändert die Standardkennwortrichtlinie für eine Active Directory-Domäne.	ActiveDirectory
Set-ADDomain	Ändert eine Active Directory-Domäne.	ActiveDirectory
Set-ADDomainMode	Legt den Domänenmodus für eine Active Directory-Domäne fest.	ActiveDirectory
Set-ADFineGrainedPasswordPolicy	Ändert eine abgestimmte Active Directory-Kennwortrichtlinie.	ActiveDirectory
Set-ADForest	Ändert eine Active Directory-Gesamtstruktur.	ActiveDirectory
Set-ADForestMode	Legt den Gesamtstrukturmodus für eine Active Directory-Gesamtstruktur fest.	ActiveDirectory
Set-ADGroup	Ändert eine Active Directory-Gruppe.	ActiveDirectory
Set-ADObject	Ändert ein Active Directory-Objekt.	ActiveDirectory
Set-ADOrganizationalUnit	Ändert eine Active Directory-Organisationseinheit.	ActiveDirectory
Set-ADServiceAccount	Ändert ein Active Directory-Dienstkonto.	ActiveDirectory
Set-ADUser	Ändert einen Active Directory-Benutzer.	ActiveDirectory
Set-Alias	Erstellt oder ändert einen Alias (alternativer Name) für ein Cmdlet oder anderes Befehlselement in der aktuellen Windows PowerShell-Sitzung.	Microsoft.PowerShell.Utility
Set-AppLockerPolicy	Legt die AppLocker-Richtlinie für das angegebene Gruppenrichtlinienobjekt fest.	AppLocker
Set-AuthenticodeSignature	Fügt einem Windows PowerShell-Skript oder einer anderen Datei eine Authenticode-Signatur hinzu.	Microsoft.PowerShell.Security
Set-BitsTransfer	Ändert die Eigenschaften eines vorhandenen Übertragungsauftrags des intelligenten Hintergrundübertragungsdiensts (BITS).	BitsTransfer
Set-BpaResult	Mit dem Cmdlet "Set-BPAResult" können Sie vorhandene Ergebnisse eines Best Practices Analyzer (BPA)-Scans ein- oder ausschließen, um nur die gewünschten Scanergebnisse anzuzeigen.	BestPractices

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
Set-ClusterLog	Festlegen der Größe und Detailebene für das Clusterprotokoll	FailoverClusters
Set-ClusterOwnerNode	Angeben für eine Ressource in einem Failovercluster, welche Knoten die Ressource besitzen können. Angeben von Informationen über die bevorzugte Reihenfolge unter Besitzerknoten für einen geclusterten Dienst bzw. eine Clusteranwendung (Ressourcengruppe)	FailoverClusters
Set-ClusterParameter	Steuern bestimmter Eigenschaften eines Objekts in einem Failovercluster, z. B. einer Ressource, einer Gruppe oder eines Netzwerks	FailoverClusters
Set-ClusterQuorum	Konfigurieren von Quorumoptionen für einen Failovercluster	FailoverClusters
Set-ClusterResource-Dependency	Angeben der Ressourcen, von denen eine bestimmte Ressource innerhalb eines Failoverclusters abhängig ist. Vorhandene Abhängigkeiten werden durch die von Ihnen angegebenen Abhängigkeiten überschrieben.	FailoverClusters
Set-Content	Schreibt neuen Inhalt in ein Element oder ersetzt den Inhalt eines Elements durch neuen Inhalt.	Microsoft.PowerShell.Management
Set-Date	Ändert die Systemzeit auf dem Computer in die von Ihnen angegebene Zeit.	Microsoft.PowerShell.Utility
Set-DbTable	Speichert eine ADO.NET-DataTable.	ITVisions_ PowerShell_ Extensions
Set-DependencyProperty	Weist einem Objekt einen Wert für eine Abhängigkeitseigenschaft zu.	WPK
Set-DirectoryValue	Setz einen Wert in einem Verzeichnisdiensteintrag in einem von ADSI unterstützten Verzeichnisdienst (z.B. LDAP, Active Directory, WinNT)	ITVisions_ PowerShell_ Extensions
Set-ExecutionPolicy	Ändert die Benutzereinstellung für die Windows PowerShell-Ausführungsrichtlinie.	Microsoft.PowerShell.Security
Set-GPInheritance	Deaktiviert die Vererbung oder hebt die Deaktivierung der Vererbung für eine angegebene Domäne oder Organisationseinheit auf.	GroupPolicy
Set-GPLink	Legt die Eigenschaften der angegebenen Gruppenrichtlinienobjekt-Verknüpfung fest.	GroupPolicy

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
Set-GPPermissions	Gewährt für ein Gruppenrichtlinienobjekt oder alle Gruppenrichtlinienobjekte in einer Domäne eine Berechtigungsstufe für einen Sicherheitsprinzipal.	GroupPolicy
Set-GPPrefRegistryValue	Konfiguriert ein Registrierungseinstellungselement unter der Computerkonfiguration oder der Benutzerkonfiguration in einem Gruppenrichtlinienobjekt.	GroupPolicy
Set-GPRegistryValue	Konfiguriert eine oder mehrere registrierungsbasierte Richtlinieneinstellungen unter der Computerkonfiguration oder der Benutzerkonfiguration in einem Gruppenrichtlinienobjekt.	GroupPolicy
Set-Item	Ändert den Wert eines Elements in den Wert, der im Befehl angegeben wird.	Microsoft.PowerShell.Management
Set-ItemProperty	Erstellt oder ändert den Wert für eine Eigenschaft eines Elements.	Microsoft.PowerShell.Management
Set-Location	Legt den aktuellen Arbeitsspeicherort auf einen angegebenen Speicherort fest.	Microsoft.PowerShell.Management
Set-NlbCluster	Bearbeitet die Konfiguration eines NLB-Clusters.	NetworkLoadBalancingClusters
Set-NlbClusterNode	Bearbeitet die NLB-Clusterknoteneinstellungen.	NetworkLoadBalancingClusters
Set-NlbClusterNodeDip	Bearbeitet die dedizierte IP-Adresse eines NLB-Clusters.	NetworkLoadBalancingClusters
Set-NlbClusterPortRule	Bearbeitet die Portregeln für einen NLB-Cluster.	NetworkLoadBalancingClusters
Set-NlbClusterPortRuleNodeHandlingPriority	Legt die Hostpriorität einer Portregel für einen bestimmten NLB-Knoten fest.	NetworkLoadBalancingClusters
Set-NlbClusterPortRuleNodeWeight	Legt das Lastgewicht einer Portregel für einen bestimmten NLB-Knoten fest.	NetworkLoadBalancingClusters
Set-NlbClusterVip	Bearbeitet die virtuelle IP-Adresse eines NLB-Clusters.	NetworkLoadBalancingClusters
Set-PSBreakpoint	Legt einen Haltepunkt für eine Zeile, einen Befehl oder eine Variable fest.	Microsoft.PowerShell.Utility
Set-PSDebug	Aktiviert bzw. deaktiviert Skript-Debuggingfunktionen, legt die Ablaufverfolgungsebene fest und schaltet den Strict-Modus um.	Microsoft.PowerShell.Core
Set-PSSessionConfiguration	Ändert die Eigenschaften einer registrierten Sitzungskonfiguration.	Microsoft.PowerShell.Core

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
Set-Service	Startet, beendet und hält einen Dienst an und ändert seine Eigenschaften.	Microsoft.PowerShell.Management
Set-StrictMode	Erstellt und erzwingt Codierungsregeln in Ausdrücken, Skripts und Skriptblöcken.	Microsoft.PowerShell.Core
Set-TraceSource	Konfiguriert, startet und beendet eine Ablaufverfolgung von Windows PowerShell-Komponenten.	Microsoft.PowerShell.Utility
Set-Variable	Legt den Wert einer Variablen fest. Erstellt die Variable, wenn keine Variable mit dem angeforderten Namen vorhanden ist.	Microsoft.PowerShell.Utility
Set-WBPolicy	Legt das WBPolicy-Objekt als Sicherungsrichtlinie für die geplanten Sicherungen fest.	Windows.ServerBackup
Set-WBSchedule	Legt die Zeitpunkte fest, zu denen tägliche Sicherungen für die Sicherungsrichtlinie (WBPolicy-Objekt) erstellt werden.	Windows.ServerBackup
Set-WBVssBackupOptions	Legt einen Wert fest, der angibt, ob es sich bei den mit der Sicherungsrichtlinie (WBPolicy-Objekt) erstellten Sicherungen um VSS-Kopiesicherungen (Volumeschattenkopie-Dienst) oder um vollständige VSS-Sicherungen handelt.	Windows.ServerBackup
Set-WebBinding	Ändert eine Eigenschaft einer IIS-Sitebindung.	WebAdministration
Set-WebConfiguration	Legt den Wert einer IIS-Konfigurationseinstellung auf den im Befehl angegebenen Wert fest.	WebAdministration
Set-WebConfiguration-Property	Legt den Wert einer IIS-Konfigurationseinstellung auf den im Befehl angegebenen Wert fest.	WebAdministration
Set-WebGlobalModule	Konfiguriert ein IIS-Modul.	WebAdministration
Set-WebHandler	Konfiguriert einen IIS-Anforderungshandler.	WebAdministration
Set-WebManagedModule	Konfiguriert ein verwaltetes IIS-Modul.	WebAdministration
Set-WmiInstance	Erstellt oder aktualisiert eine Instanz einer vorhandenen WMI-Klasse (Windows Management Instrumentation, Windows-Verwaltungsinstrumentation).	Microsoft.PowerShell.Management
Set-WSManInstance	Ändert die Verwaltungsinformationen zu einer Ressource.	Microsoft.WSMan.Management
Set-WSManQuickConfig	Konfiguriert den lokalen Computer für die Remoteverwaltung.	Microsoft.WSMan.Management

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
Show-EventLog	Zeigt die Ereignisprotokolle des lokalen Computers oder eines Remotecomputers in der Ereignisanzeige an.	Microsoft.PowerShell.Management
Show-Member	Zeige die Mitglieder eines Objekts in einem GridView.	PowerShellPack
Show-SyntaxForCurrentCommand	Zeigt die Syntax der aktuell ausgewählten Funktion oder des cmdlet an.	PowerShellPack
Show-TypeConstructor	Zeigt die Konstruktoren für einen .NET-Typ.	PowerShellPack
Show-TypeConstructorForCurrentType	Zeigt Typkonstruktoren für den aktuell ausgewählten .NET-Typ an.	PowerShellPack
Show-UIElement	Zeigt ein UI-Element an.	WPK
Show-Window	Zeigt ein WPF-Steuerelement innerhalb einse Fensters an, und wird vom –Show-Parameter aller Kommandos innerhalb des WPK verwendet.	WPK
Sort-Object	Sortiert Objekte nach Eigenschaftenwerten.	Microsoft.PowerShell.Utility
Split-IseFile	Splits up a large file containing many functions into many files with one function each	PowerShellPack
Split-Path	Gibt den angegebenen Teil eines Pfads zurück.	Microsoft.PowerShell.Management
Start-BitsTransfer	Erstellt einen neuen Übertragungsauftrag des intelligenten Hintergrundübertragungsdiensts (BITS).	BitsTransfer
Start-Cluster	Starten des Clusterdiensts für alle Knoten des Clusters, mit denen noch nicht angefangen wurde.	FailoverClusters
Start-ClusterGroup	Schalten mindestens eines geclusterten Diensts und mindestens einer Clusteranwendung (auch als Ressourcengruppen bezeichnet) in einem Failovercluster in den Onlinemodus	FailoverClusters
Start-ClusterNode	Starten des Clusterdiensts auf einem Knoten in einem Failovercluster	FailoverClusters
Start-ClusterResource	Schalten einer Ressource in einem Failovercluster in den Onlinemodus	FailoverClusters
Start-FileSystemWatcher	Startet die Überwachung von Dateiänderungen.	PowerShellPack
Start-Job	Startet einen Windows PowerShell-Hintergrundauftrag.	Microsoft.PowerShell.Core

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
Start-NlbCluster	Startet alle Knoten in einem NLB-Cluster.	NetworkLoad-BalancingClusters
Start-NlbClusterNode	Startet einen NLB-Clusterknoten.	NetworkLoad-BalancingClusters
Start-PowerShell-Command	Startet ein PowerShell-Kommando	WPK
Start-Process	Startet einen oder mehrere Prozesse auf dem lokalen Computer.	Microsoft.Power-Shell.Management
Start-ProcessAsAdminist-rator	Startet einen Prozess mit der "Run as administrator"-Option.	PSUserTools
Start-Service	Startet einen oder mehrere beendete Dienste.	Microsoft.Power-Shell.Management
Start-Sleep	Hält die Aktivitäten in einem Skript oder einer Sitzung für die angegebene Zeitspanne an.	Microsoft.Power-Shell.Utility
Start-Task	Startet eine geplante Aufgabe.	TaskScheduler
Start-Trace	Startet das PowerShell-Tracing.	PSDiagnostics
Start-Transaction	Startet eine Transaktion.	Microsoft.Power-Shell.Management
Start-Transcript	Erstellt eine Aufzeichnung einer gesamten oder teilweisen Windows PowerShell-Sitzung in einer Textdatei.	Microsoft.Power-Shell.Host
Start-WBBackup	Startet einen einmaligen Sicherungsvorgang.	Windows.Server-Backup
Start-WebAppPool	Startet einen Anwendungspool.	WebAdministration
Start-WebItem	Startet einen Anwendungspool oder eine Site.	WebAdministration
Start-Website	Startet eine IIS-Website.	WebAdministration
Start-WPFJob	Startet einen WPF-Job, um ein UI im Hintergrund abzubilden.	WPK
Stop-Cluster	Beenden des Clusterdiensts auf allen Knoten in einem Failovercluster, wodurch alle im Cluster konfigurierten Dienste und Anwendungen beendet werden	FailoverClusters
Stop-ClusterGroup	Schalten mindestens eines geclusterten Diensts und mindestens einer Clusteranwendung (auch als Ressourcengruppen bezeichnet) in einem Failovercluster in den Offlinemodus	FailoverClusters
Stop-ClusterNode	Beenden des Clusterdiensts auf einem Knoten in einem Failovercluster	FailoverClusters

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
Stop-ClusterResource	Schalten einer Ressource in einem Failovercluster in den Offlinemodus	FailoverClusters
Stop-Computer	Beendet lokale und Remotecomputer (fährt sie herunter).	Microsoft.PowerShell.Management
Stop-Job	Beendet einen Windows PowerShell-Hintergrundauftrag.	Microsoft.PowerShell.Core
Stop-NlbCluster	Beendet alle Knoten eines NLB-Clusters.	NetworkLoadBalancingClusters
Stop-NlbClusterNode	Beendet einen Knoten in einem NLB-Cluster.	NetworkLoadBalancingClusters
Stop-PowerShellCommand	Beendet ein PowerShell-Kommando	WPK
Stop-Process	Beendet einen oder mehrere ausgeführte Prozesse.	Microsoft.PowerShell.Management
Stop-Service	Beendet einen oder mehrere ausgeführte Dienste.	Microsoft.PowerShell.Management
Stop-Task	Beendet eine reguläre Aufgabe.	TaskScheduler
Stop-Trace	Hält das PowerShell-Tracing an.	PSDiagnostics
Stop-Transcript	Beendet eine Aufzeichnung.	Microsoft.PowerShell.Host
Stop-WebAppPool	Beendet einen Anwendungspool.	WebAdministration
Stop-WebItem	Beendet einen Anwendungspool oder eine Site.	WebAdministration
Stop-Website	Beendet die angegebene Website.	WebAdministration
Suspend-BitsTransfer	Hält einen Übertragungsauftrag des intelligenten Hintergrundübertragungsdiensts (BITS) an.	BitsTransfer
Suspend-ClusterNode	Anhalten der Aktivität auf einem Failoverclusterknoten, d. h. Anhalten des Knotens	FailoverClusters
Suspend-ClusterResource	Aktivieren Sie die Wartung für eine Datenträgerressource oder ein freigegebenes Clustervolume, sodass Sie ein Datenträgerwartungstool ausführen können, ohne einen Failover auszulösen.	FailoverClusters
Suspend-NlbCluster	Hält alle Knoten eines NLB-Clusters an.	NetworkLoadBalancingClusters
Suspend-NlbClusterNode	Hält einen bestimmten Knoten in einem NLB-Cluster an.	NetworkLoadBalancingClusters
Suspend-Service	Hält einen oder mehrere ausgeführte Dienste an.	Microsoft.PowerShell.Management

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
Switch-CommentOrText	Schaltet zwischen kommentiertem und unkommentiertem Text hin und her.	PowerShellPack
Switch-Selected-CommentOrText	Schaltet zwischen Kommentaren im ausgewählten Text hin und her.	PowerShellPack
Tee-Object	Speichert die Befehlsausgabe in einer Datei oder Variablen und zeigt sie in der Konsole an.	Microsoft.PowerShell.Utility
Test-32Bit	Stellt fest, ob ein Computer eine 32-bit-Prozessorarchitektur hat.	PSSystemTools
Test-64Bit	Stellt fest, ob ein Computer eine 64-bit-Prozessorarchitektur hat.	PSSystemTools
Test-AppLockerPolicy	Testet, ob ein bestimmter Benutzer die Eingabedateien basierend auf der angegebenen AppLocker-Richtlinie ausführen darf.	AppLocker
Test-Cluster	Ausführen von Validierungstests für Failoverclusterhardware und -einstellungen. Tests können sowohl vor als auch nach der Clustereinrichtung ausgeführt werden.	FailoverClusters
Test-ClusterResource-Failure	Simulieren eines Fehler einer Clusterressource	FailoverClusters
Test-Computer-SecureChannel	Testet und repariert den sicheren Kanal zwischen dem lokalen Computer und seiner Domäne.	Microsoft.PowerShell.Management
Test-Connection	Sendet ICMP-Echoanforderungspakete ("Pings") an einen oder mehrere Computer.	Microsoft.PowerShell.Management
Test-DbConnection	Prüft, ob die Verbindung zu einer Datenbank möglich ist.	ITVisions_ PowerShell_ Extensions
Test-IsAdministrator	Zeigt an, ob der aktuelle Anwender ein Administrator ist.	PSUserTools
Test-ModuleManifest	Überprüft, ob in einer Modulmanifestdatei der Inhalt eines Moduls genau beschrieben wird.	Microsoft.PowerShell.Core
Test-Path	Bestimmt, ob alle Elemente eines Pfads vorhanden sind.	Microsoft.PowerShell.Management
Test-WSMan	Testet, ob auf einem lokalen Computer oder Remotecomputer der WinRM-Dienst ausgeführt wird.	Microsoft.WSMan.Management
Trace-Command	Konfiguriert und startet eine Ablaufverfolgung des angegebenen Ausdrucks oder Befehls.	Microsoft.PowerShell.Utility

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
Undo-Transaction	Setzt die aktive Transaktion zurück.	Microsoft.PowerShell.Management
Uninstall-ADRMS	Entfernt AD RMS von diesem Computer.	ADRMS
Uninstall-ADService-Account	Deinstalliert ein Active Directory-Dienstkonto auf einem Computer.	ActiveDirectory
Unlock-ADAccount	Entsperrt ein Active Directory-Konto.	ActiveDirectory
Unregister-Event	Beendet ein Ereignisabonnement.	Microsoft.PowerShell.Utility
Unregister-PowerShell-Command	Entfernt einen PowerShell-Skriptblock.	WPK
Unregister-PSSession-Configuration	Löscht eine registrierte Sitzungskonfiguration vom Computer.	Microsoft.PowerShell.Core
Update-ADRMS	Führt im Anschluss an ein Windows-Update eine Aktualisierung von AD RMS aus.	ADRMS
Update-ClusterIPResource	Erneuern oder Freigeben der DHCP-Lease für eine IP-Adressressource in einem Failovercluster	FailoverClusters
Update-ClusterVirtual-MachineConfiguration	Aktualisieren der Konfiguration eines virtuellen Clustercomputers innerhalb eines Failoverclusters	FailoverClusters
Update-FormatData	Aktualisiert die Formatierungsdaten in der aktuellen Sitzung.	Microsoft.PowerShell.Utility
Update-List	Fügt einem Eigenschaftenwert, der eine Auflistung von Objekten enthält, Elemente hinzu oder entfernt Elemente aus dem Eigenschaftenwert.	Microsoft.PowerShell.Utility
Update-TypeData	Aktualisiert die aktuelle erweiterte Typkonfiguration durch erneutes Laden der *.TYPES.PS1XML-Dateien in den Arbeitsspeicher.	Microsoft.PowerShell.Utility
Update-WPFJob	Aktualisiert einen laufenden WPF Job unter Ausführung eines PowerShell-Skripts	WPK
Use-Transaction	Fügt den Skriptblock der aktiven Transaktion hinzu.	Microsoft.PowerShell.Management
Wait-Event	Wartet, bis ein bestimmtes Ereignis ausgelöst wird, bevor die Ausführung fortgesetzt wird.	Microsoft.PowerShell.Utility
Wait-Job	Unterdrückt die Eingabeaufforderung, bis ein oder alle Windows PowerShell-Hintergrundaufträge abgeschlossen sind, die in der Sitzung ausgeführt werden.	Microsoft.PowerShell.Core

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

CO_Name	CO_DescShort	ModuleName
Wait-Process	Wartet, bis die Prozesse beendet wurden, bevor weitere Eingaben angenommen werden.	Microsoft.PowerShell.Management
Where-Object	Erstellt einen Filter, der steuert, welche Objekte über eine Befehlspipeline übergeben werden.	Microsoft.PowerShell.Core
Write-ColorizedHTML	Erzeugt eine farbige Ausgabe von HTML-Code.	PowerShellPack
Write-Debug	Schreibt eine Debugmeldung in die Konsole.	Microsoft.PowerShell.Utility
Write-Error	Schreibt ein Objekt in den Fehlerstream.	Microsoft.PowerShell.Utility
Write-EventLog	Schreibt ein Ereignis in ein Ereignisprotokoll.	Microsoft.PowerShell.Management
Write-Host	Schreibt eine benutzerdefinierte Ausgabe in einen Host.	Microsoft.PowerShell.Utility
Write-Output	Sendet die angegebenen Objekte an den nächsten Befehl in der Pipeline. Wenn es sich um den letzten Befehl in der Pipeline handelt, werden die Objekte in der Konsole angezeigt.	Microsoft.PowerShell.Utility
Write-Progress	Zeigt eine Statusanzeige in einem Windows PowerShell-Befehlsfenster an.	Microsoft.PowerShell.Utility
Write-Verbose	Schreibt Text in den Stream für ausführliche Meldungen.	Microsoft.PowerShell.Utility
Write-Warning	Schreibt eine Warnmeldung.	Microsoft.PowerShell.Utility

***Tabelle C.2**: Commandlets in PowerShell 2.0 und einigen Erweiterungen (Forts.)*

Stichwortverzeichnis

& 65, 144
$_ 124, 276
$(…) 180

A

Ablaufverfolgung 31, 308
About 117
AbsoluteTimerInstruction 253
abstract 703
Accelerator 121
Access Control Entry 500
Access Control List 500, 511
Access Control Type 500
Access Mask 500
AccessControl 499, 504
AccessMask 395
Account Manager 360
AccountDisabled 538
AceFlags 500
ACL 503, 516
Active Directory 31, 187, 227, 345–346, 356, 527, 534, 539, 554, 569–570, 618
  PowerShell 554
  Struktur 569
  Suche 543, 564
Active Directory Application Mode 570
Active Directory Management Objects 569
Active Directory Service Interface siehe ADSI
ActiveScriptEventConsumer 254
Active-Scripting 112
ActiveX Data Objects 417, 428, 523
ADAccount 559
ADComputer 559
Add-ADGroupMember 568
AddCommand() 683
Add-Computer 447
Add-Content 403–404
Add-DirectoryEntry 355
Add-DistributionGroupMember 580
Add-LDAPObject 551, 620–621
Add-Member 96, 273
Add-PSSnapin 634, 640
AddScript 680
AddScript() 683
Add-Type 277
Add-VirtualHardDisk 358
Add-WindowsFeature 459, 463–465
ADO.NET 417, 425, 523, 544
ADODB.Connection 524
ADPowerShell 554, 559
ADRMS 346
ADSI 519–520, 524–525, 528, 531
  .NET 517, 519, 524
  Bindung 526–527
  COM 524, 531
  Container 532
  Pfad 526
AdsPath 543
ADUser 559
ADWS 555
AgentPC 519
Aktivierung 361
Alias 53, 56, 185–186, 377
Aliaseigenschaft 90, 95
AliasInfo 56
AllowEmptyCollection 625
AllowEmptyString 625
AllowNull 625
AllSigned 113
Alvin Kersh 518
Änderungshistorie 422
Animation 611
Ankerelement 133
Anwendungspool 588, 590
Anzeigesprache 328
AppendChild(). 411
AppLockerPolicy 471–472
$Args 110, 124
Array 140–141, 143, 713
Aruk Kumaravel 525
AsJob 314
ASP.NET 227
Assembly 218, 333, 339, 639, 668
  verbreiten 711
AssocClass 482
ASSOCIATORS OF 250
Assoziation 235
  WMI 232, 235
Attribut 697, 702, 712
  indiziert 713
Aufzählung 221
Ausdruck
  Regulär 131
Ausdruckauflösung 125
Ausdrucksmodus 63
Ausführungsrichtlinie 112
Ausgabeobjekt 643
Auslagerungsdatei 361

Authentifizierung 538
AuthorizationRuleCollection 505–506

B

Background Intelligent Transfer Service 347
Backup 348
Backup-GPO 574
Base 544, 564
Basisauthentifizierung 283
Basisklasse 421
Batterie 451
Bedingung 150
Beep() 217
Befehl
  Extern 53, 64
Befehlsgeschichte 363
Befehlsmodus 63
Befehlsobjekt 426
Begin 385
BeginProcessing() 633, 638
Benutzer 227, 548, 698, 701
  anlegen 537
  löschen 539
  umbenennen 539
  verschieben 540
Benutzerdaten lesen 566
Benutzergruppe 568
Benutzerkennwort 538
Benutzerkontensteuerung 110, 196
Benutzerkonto 566
Benutzername 184
Benutzerschnittstelle 226
Berechnung 102
Best Practice 346, 596
Bezeichner 711
Beziehung 701
Bibliothek 708
Big Endian 543
Bild 353
Bildschirmschoner 279, 519
Binärdatei 404
Binärmodul 685
Bindung
  ADSI 526
  serverlos 527
  WMI 259
Bing 605
BIOS 226
Blatt 524
BMC 225
Boot-Konfiguration 226
break 145, 147, 160
Bypass 113
ByPropertyName 83–84
Byte 126
ByValue 83–84
BZIP2 393

C

C# 117, 277, 279, 614–615, 631–632, 637, 705
C++ 708–709
C++/CLI 631
Canvas 608
CATID 251
Checkpoint-Computer 448
Children 523
ChildSession 296
CHKDSK 227
CIM 226, 228
  Repository 235
Cisco 225
City 560–561
ClassCreationEvent 318
ClassDeletionEvent 318
ClassModificationEvent 318
Clear-Content 403
Clear-EventLog 284
Clear-Eventlog 495
Clear-History 363
Clear-Host 159, 365
Clear-Item 377
Click 609
CLIXML 411
Close() 607
-cmatch 131
cmd.exe 75
Cmdlet 665
Cmdlet Help Editor 668
cn 534
-cnotmatch 131
Codeeigenschaft 90, 94
CodeZone 28
Codezone 28
Codezone Premier Site 28
Collaboration Data Objects 216
COM 223, 709
  Kategorie 251
  Komponente 227
  Moniker 526
  Sicherheit 249
Command Line Event Consumer 255
Command Mode 63
Commandlet 31, 53, 64, 68, 85, 190, 615
  erstellen 615, 624, 631
  Erweiterung 213
  funktionsbasiert 615, 620
  Klasse 633
  Konvention 656, 673

Provider 186
Referenz 715
Verkettung 653
CommandNotFoundException 163
Comma-Separated Values 406
CommitChanges() 521, 531, 538
Common Intermediate Language 707
Common Language Runtime 707
Common Language Specification 707
Common Management Information Protocol 226
Common Type System 569, 707
Compare-Object 104
Complete-Transaction 309, 312
Component Object Model siehe COM
Computer 234, 548, 701
ComputerName 110
Computername 447
Computerverwaltung 445
ConfigurationNamingContext 557
Confirm 305, 665–666
Connection 424
Console.WriteLine() 661, 674
Container 532, 575
Container-Klasse 524
Continue 145, 147, 160, 162
ConvertFrom-StringData 327
Convert-Html 415
Convert-Path 190
ConvertTo-CSV 406
ConvertTo-DataTemplate 611
ConvertTo-WebApplication 589
ConvertTo-XML 413
Convert-Xml 413
copy 383
Copy-GPO 573
Copy-Item 160, 377, 383, 440
CORBA 709
Count 78, 140, 156
Country 561
CPU 104
Create() 396
CreateCommand() 426
CreateElement() 411
CreateInstance() 452
CreationTime 703
CSV 61, 404
CSV-Datei 105
CultureInfo 643
Cursor 421–422
CustomerID 627

D

Dana Scully 518
DataReader 420–421, 426, 428–429
DataRow 95–96
DataSet 420–421, 428–432
DataTable 430
Datei 227, 698
Eigenschaft 384, 387
kopieren 383
Rechte 227
verschieben 383
Dateiname 53
Dateinamenerweiterung 105, 107
Dateisystem 31, 186, 503, 701, 711
Dateisystemfreigabe 395
Datenabfrage 250
Datenbank 235, 355, 417
Datenbankmanagementsystem 424
Datenbankverbindung 424
Datenbankzeile 95
Datenbankzugriff 417
Datenbereich 327
Datenbindung 611
Datendatei 327
Datenmenge 185
Datenquellensteuerelement 420
Datentyp 120, 123, 142, 528, 654, 712
.NET 76
PowerShell 120
WMI 230
Datenzugriff 425
DateTime 78, 80, 87
Datum 138
Day 78
DbCommand 426
DbDataReader 428
DbProviderFactories 419
DCOM 236, 649, 661
Konfiguration 227
DDL 250
debug 305–306, 661
Debugging 31, 203
DebugPreference 661
Decimal 126
Deep Throat 518
Default Domain Policy 578
DefaultNamingContext 557
Deinstallation 457
Delete() 703
Delimiter 404
Deployment 711
Description 535, 561, 627
Deserialisierung 290
Desktop 227
Desktop-Anwendungen 708
Destruktor 698, 713
Dezimalzahl 126
DHCP 485–486

Dialogfenster 183
Digest 283
Directory 503
Directory Management Objects 569
DirectoryEntry 95–96, 216, 521–525, 528–533, 537, 540
DirectoryInfo 87, 98, 273, 387, 503, 703
DirectorySearcher 544
DirectorySecurity 504
DirectoryString 535
Disable-ComputerRestore 448
Disable-Mailbox 580
Disable-PSSessionConfiguration 295
Disk Quotas 227
DisplayName 560
Distinguished Name 526, 532, 535, 558, 560
Distributed COM 283
Distributed File System 227
DML 250
DMTF 225
DNS-Server 227
do 145
DockPanel 608
Dokument 403
Dokumentation
  .NET 72
  Active Directory 537
Dollarzeichen 102
Domäne 447, 527, 570, 698, 701
  hinzufügen 447
Dot Sourcing 61, 112, 333, 372, 616, 621
dotnetpro 28
DotNetTypes.Format.ps1xml 167, 171–172
Double 126
DownloadString() 216, 490
DriveType 219
Druckauftrag 451
  löschen 451
Drucker 181, 226
  verwalten 451
Druckerport 452
Druckerverwaltung 451

E

echo 63
Eigenschaft 90
Eigenschaftenzwischenspeicher 521
Eigenschaftssatz 90, 92
Eingabe 183
Eingabeaufforderung 365–366
Eingabemaske 599
Eingabeobjekt 652
Eingabesteuerelement 610
Einzelschrittmodus 307
Else 151
E-Mail 489, 492
  EmailEvent 253
EmailAddress 561
EmailEvent 253
Enable-ComputerRestore 448
Enable-PSRemoting 286–287
Enable-PSSessionConfiguration 295, 298
Enable-PSTrace 348
Encoding 404
End 385
EndProcessing() 633, 638
Enter-PSSession 288, 295, 299
Enum 220
EnumerateCollection 645
Enumeration 221
Enumerationsklasse 220
Ereignis 697, 713
  .NET 325
  PowerShell 317, 326
  WMI 253, 317
Ereignisabfrage 251
Ereigniskonsument 253–255
  permanent 254
  temporär 254
Ereignisprotokoll 105, 226–227, 236, 255, 493
  Überwachung 253, 319
Ereignisprovider 253
Ereignissystem 317
$Error 124, 163
ErrorAction 162–163
$ErrorActionPreference 163
ErrorRecord 159, 165
Event 229
Event siehe Ereignis
EventConsumer 229
EventViewerConsumer 254
Example 627
Exception 159, 163, 165, 657, 662
Exchange Management Shell 357, 579
Exchange Server 227, 237, 579
  Version 2007 72
ExecuteNonQuery() 426
ExecuteReader() 426, 428
ExecuteRow() 426
ExecuteScalar() 426
Exists() 530
exit 145
Exit-PSSession 289, 295
Export-Alias 61
Export-CliXml 411
Export-Console 335, 641
Export-Counter 499
Export-CSV 406
Export-Csv 99, 404

Export-ModuleMember 339
Express 631
Expression 179
–expression 372
Expression Mode 63
Extended Reflection 76
Extended Type System 76, 90, 95–96, 261, 524
Extrinsic Event 253

F

facsimileTelephoneNumber 561
Failover Cluster 346
$false 124
Fax 561
FBI 518, 540
Feature 460
FeatureOperationResult 465
Fehlerbehandlung 159, 708
Fehlerklasse 145, 165
Fehlersuche 305, 658
Fehlertext 145
Fernaufruf 284
Fernausführung 110, 283
  Hintergrundauftrag 316
Fernverwaltung 283
Fernzugriff 283
Field 90
File 503, 700
FileInfo 87, 98, 273, 387–388, 503, 703
FileInformation 473
FileSecurity 503–504
FileSystem 353, 377
FileSystemAccessRule 506
FileSystemInfo 703
FileSystemObject 700
FileSystemRights 221
FileSystemWatcher 325
FileVersionInfo 458
Find() 523
Firewall 361
For 146, 149
ForEach 79, 104–105, 145, 149, 523, 647
Foreach-Object 79, 102, 140, 149, 155, 308, 403, 646
Forest 570
Format 179
Formatkennzeichner 175
Format-List 75, 84, 167, 505
Format-Table 92, 105, 167, 172–173, 179, 505
Format-Wide 167
Format-Xml 408–409
Fox Mulder 518
FullArmor 345
FullName 703
function 53, 145, 153, 159, 185
Function Prompt 365
Funktion 53, 153, 185–186
  eingebaut 159

G

GeneralizedTime 535
Gesamtstruktur 570–571
Geschäftsanwendung 672
GetAccessRules() 505–506
Get-Acl 499, 503–505, 515
Get-ADDomain 571
Get-ADDomainController 571
Get-ADForest 571
Get-ADGroup 568
Get-ADGroupMember 568
Get-ADObject 55, 517, 550, 562, 564
Get-ADOptionalFeature 571
Get-ADOrganizationalUnit 565
Get-ADPrincipalGroupMembership 568–569
Get-ADRootDSE 571
Get-ADUser 566–567
Get-Alias 56–57
Get-AppLockerFileInformation 471
Get-AppLockerPolicy 471–472
Get-BPAModel 597
Get-BPAResult 597
Get-Bparesult 598
Get-CDRomDrive 355
Get-CDRomdrive 449
Get-ChildItem 54, 84–85, 98, 100–101, 105, 185, 188, 382–383, 439, 455
Get-ChildItem, 439
Get-Clipboard 602
Get-Command 56, 68–69, 342
Get-ComputerInfo 643
Get-Computername 636
Get-ComputerRestorePoint 448
Get-Content 188, 377, 403–404, 454
Get-Counter 284, 497–498
Get-Credential 303, 478, 489
Get-Cwd 188
Get-DataRow 355, 435
Get-DataTable 355, 435
Get-Date 80, 138–139
Get-DHCPServer 486
Get-DhcpServer 44
Get-DirectoryChildren 355
Get-DirectoryEntry 125, 355, 656
Get-DirSize 615
Get-Disk 378, 380, 449, 645, 649, 651–652, 654, 661
Get-DisplaySetting 450
Get-DomainController 44, 527
Get-DVDDrive 358
Get-Event 321

Get-EventLog 41, 105, 284, 493–494
Get-ExCommand 579
Get-ExecutionPolicy 43
Get-ExportedType 388
GetFactoryClasses() 419
Get-FileVersionInfo 388, 458
Get-FloppyDrive 358
Get-Flug 672
Get-Flugziele 672
Get-Font 447
Get-FreeDiskSpace 380–381
Get-GPInheritance 577
Get-GPO 573–574
Get-GPOReport 576
Get-GPPermissions 578
Get-GPPrefRegistryValue 578
Get-GPRegistryValue 578
Get-GPResultantSetOfPolicy 577
Get-GPStarterGPO 574
Get-Help 70, 72, 117, 615, 629, 668
Get-History 363
Get-Host 364
Get-HotFix 284
Get-HttpResource 491
Get-Item 384, 439, 588
Get-ItemProperty 384, 439
Get-Job 313–314
Get-Keyboard 449
Get-LDAPChildren 551, 619–620
Get-LDAPObject 551, 620
Get-Location 56, 188, 377
Get-LogicalDiskInventory 378
GetLongDateString() 80
GetLongTimeString() 80
Get-Mailbox 579–580
Get-Mailboxdatabase 579
Get-Member 86, 88, 91, 95, 99, 219, 263, 270, 525, 645
Get-Members 540
Get-MemoryDevice 355, 449
Get-Methode 91
Get-Module 339, 341, 685
Get-MountPoint 44
Get-MultiTouchMaximum 450
GetNames() 220
Get-NetworkAdapter 355, 449
GetObject() 224
Get-OSVersion 445
Get-Passagier 672
Get-PipelineInfo 86–87, 655
Get-PointingDevice 449
Get-PowerShellDataSource 611
Get-Process 53–56, 62, 75, 79–80, 84, 88, 95, 98–99, 102, 104, 173, 181, 284, 308, 477, 654–655
Get-Processor 355, 449–450
Get-PSDrive 378
Get-PSProvider 187
Get-PSSession 295
Get-PSSessionConfiguration 295, 297
Get-pssessionconfiguration 297
Get-PSSnapIn 336
Get-Random 127
GetRelated() 482
Get-ReparsePoint 393
Get-Service 84–85, 98, 103, 111, 284, 481
Get-SHA1 385
GetShortDateString() 80
Get-ShortPath 383
GetShortTimeString() 80
Get-SoundDevice 449
Get-Storagegroup 579
Get-Tapedrive 449
GetTempName() 223
Getter 91, 713
Get-TerminalSession 44
Get-TraceSource 308
Get-Transaction 309, 312
GetType() 86, 123, 179–180
Get-USBController 355, 449
Get-Variable 120, 124
Get-Videocontroller 355, 449
Get-VirtualHardDisk 358
Get-VMHost 358
Get-WebApplication 584
Get-WebitemState 590
Get-Website 584, 588
Get-WebvirtualDirectory 584
Get-WindowsEdition 446
Get-WindowsFeature 459–463
Get-WinEvent 37, 284
Get-WmiObject 50, 215, 259, 265–266, 284, 382, 449–450, 453, 482–483, 485, 497, 519
Get-Wmiobject Win32_Processor 450
Gigabyte 127
Gitternetz 608
GivenName 560–561
Gleichheitszeichen 143
Global 607
$global 607
Global Assembly Cache 218
Global Unique Identifier 532, 709
Google 605
GPMC 573
Grafikkarte 227, 264
Grid 608–609
GridView 169
Group 524
GROUP BY 250, 318
Group-Object 100, 105, 494

Gruppe 548
anlegen 541
auflisten 540
Mitglied aufnehmen 542
Gruppenmitgliedschaft 542
Gruppenrichtlinie 345–346, 573–574, 577
Vererbung 577
Gruppierung 100
GZIP 393

H

Hardlink 391
Hardware 227, 355
Hardwareverwaltung 449
Hash-Tabelle 141–142
Hashtable 142–143
HAVING 250, 318
Heimatordner 124
HelpMessage 625
Herausgeber 115
Here-String 128
Herunterfahren 448
Hexadezimalzahl 126
Hilfe 67
Hintergrundauftrag 313
Hintergrundprozess 480
History 363
HKCU 185
HKEY_CURRENT_USER 440
HKEY_LOCAL_MACHINE 440
HKLM 185
$Home 124
HomeDirectory 561
HomeDrive 561
$Host 364
Hosting 677
Hotfix 227
HTML 415, 599
HTTP 283
HTTPS 283
HTTP-Server 490
Hyper-V 359

I

IADs 521, 524
IADsComputer 524
IADsContainer 521, 524
IADsGroup 524
IADsUser 524, 538
Identität 589
Identity 500
IdentityReference 508
IDL 246
IEnumerable 523
if 145, 151
IIS 527
IIS-Anwendung 589
IISwebserver 265
–imatch 131
Impersonifizierung 529
WMI 249
Import-Alias 61
Import-CliXml 363, 412
Import-Counter 499
Import-Csv 105, 383
Import-GPO 574
Import-INIFile 406
Import-LocalizedData 330
Import-Module 339, 342, 691–692
Index 140
Indexer 713
Inheritance Flags 500
INI-Datei 406
Innertext 411
-inotmacht 131
$Input 124
InputBox() 183
–InputObject 84
InputObject 219, 655–656, 658
Inquire 162
Installation 456
Installationsordner 38, 94, 124
Installationstechnologie 456
installutil.exe 634, 639
InstanceCreationEvent 318–319, 321
InstanceDeletionEvent 229, 253, 318
InstanceModificationEvent 253, 318
Instanz 698
Instanziierung 699
Instanzmitglied 713
int 120
Int32 120
Int64 126
INTEGER 535
Integrated Scripting Environment 37, 197, 677
Interface 700
Interface Definition Language 246
InternalHost 364
International .NET Association 28
Internet Control Message Protocol 486
Internet Information Server 227
Internet Information Services 228, 527, 581
Interpretermodus 195
IntervallTimerInstruction 253
Intrinsic Event 253
InvalidOperationException 523
Invoke-BPAModel 597
Invoke-Command 288–289, 291–293, 296, 302, 314
Invoke-DbCommand 435

Invoke-Expression 144
Invoke-History 363
InvokeMethod() 259
Invoke-SqlCommand 355
Invoke-WmiMethod 259, 269
IP
    Adresse 251, 485
    Konfiguration 251
IP Routing 227
ipconfig 64
IPHostEntry 488
IRQ 227
ISA 250
ISE siehe Integrated Scripting Environment
IsePack 353
Item() 522
–ItemsSource 611
iX 28

J

Java 709
Jeffrey Snover 108
John Doggett 518
Join 117, 144
–join 117, 144
Join() 130
Join-String 130
JScript .NET 277
Junction Point 391–392
Just-In-Time-Compiler 707

K

Kennwort 184, 520, 538
Kerberos 249, 283
Kill() 79
Kilobyte 127
Klammer
    rund 133
Klammeraffe 63
Klasse 120, 217, 230, 234, 698, 700, 702, 712
    .NET 633, 709, 711
    CIM 228
    COM 215, 223
    Commandlet 633, 653, 675
    WMI 228
Klassendiagramm 703–704
Klassenhierarchie 703
Klassenmitglied 217, 713
Klassenname 216
Kommandomodus 195
Kommandozeilenbefehl 64
Kommentar 118
Komponentenorientierung 706, 708
Komposition 611
Komprimierung 393
Konstruktor 698, 713
Kontakt 548
Konvention 673
Kreuzzuweisung 143

L

Label 179
LastAccessTime 703
$LastExitCode 124
Laufwerk 185–186, 191, 378, 440
LDAP 525, 527, 543
    Suchanfrage 521, 523, 543
    Suche 549
LDAP-Query 544
Leaf 532
Leistung 497
Leistungsdaten 497
Leistungsindikator 497
Length 273
Limit-EventLog 284, 493, 495
LinearGradientBrush 608
Linie 702
Literal 174
Little Endian 543
Lizensierung 361
Log File Event Consumer 255
Lokalisierung 233, 328
Loopback 286

M

m 625
Machine.config 419
MachineName 285, 637
MailMessage 488
MAML 668
Managed Code 520
Managed Object Format 246
Managed Provider 417, 544
ManagementBaseObject 258
ManagementClass 95–96, 258, 645
ManagementEventWatcher 319
ManagementObject 96, 258–259, 263–264, 482, 645–646
ManagementObjectCollection 645–646
ManagementObjectSearcher 265, 645
ManagementScope 319
Mandatory 624, 649
Manifest 340
Manifestmodul 685
Maschinencode 707
Match 96
-match 131
MaximumDriveCount 191

$MaximumErrorCount 124
Measure-Command 308
Measure-Object 102, 104, 403
Megabyte 127
Mehrsprachigkeit 328
Message 159
MessageBox 184
Metamodell 235
Metaobjekt 532
Method siehe Methode
Methode 90, 697, 702, 713
  Getter 713
  Setter 713
Microsoft Access 432
Microsoft Certified Solution Developer 28
Microsoft Exchange Server 357, 579
Microsoft Message Queing 351
Microsoft Office 228
Microsoft Outlook 480
Microsoft SQL Server 418
Microsoft Word 224
Microsoft.GroupPolicy 574
Microsoft.VisualBasic.Interaction 183
Microsoft.Win32 455
Microsoft.Win32.RegistryKey 439
Minute 78
Mitglied 697
  .NET 712
  statisch 713
  WMI 267
MMC 248
Modul 339, 341
Modulo 143
MOF 246
Monad 32
Monica Reyes 518
Moniker 526
Mono 709
Month 78
more 159, 174
Most Valuable Professional 28
Mount-SpecialFolder 379
move 383
Move-ADObject 562
Move-Item 377, 383
Move-Mailbox 580
MSCL 49
mscorlib.dll 711
MSDN 603
MSI 456–457, 634
MTA 602
Multithreading 708

N

Name 84, 560
Namensauflösung 488
Namenskonvention 673
Namensraum 230, 232, 234–235, 503, 569, 709
  .NET 709, 711
  ADSI 526
  WMI 232, 235
Namensraumhierarchie 710
Namespace 229
NamespaceCreationEvent 318
NamespaceDeletionEvent 318
Namespace-ID 526
NamespaceModificationEvent 318
NativeObject 522, 525
Navigation 185
Navigation Provider 186
Navigationsbefehl 188
Navigationsmodell 377
Navigationsparadigma 185
Navigationsprovider 517, 557, 583
.NET 31, 75, 87, 123, 139, 183, 216, 599, 631, 658, 672, 705
  Klasse 215, 709
  Runtime Host 49
  Software Development Kit 72
  Version 2.0 49
.NET Code Wise Community 28
.NET Data Provider 417–418
.NET Framework 285, 333, 460, 705–706, 710
netstat 64
Network Load Balancing 227
NetworkInterface 643
Netzlaufwerk 226
Netzlaufwerksverbindung 227
Netzwerkkarte 227, 251, 485, 698
Netzwerkkonfiguration 361, 485
Netzwerkmanagement 225, 345
Netzwerkverbindung 227
Neustart 465
Neustarten 448
New-ADGroup 568
New-ADObject 562
New-ADOrganizationalUnit 565–566
New-ADUser 566–567
New-AppLockerPolicy 471, 474
New-Buchung 672–673
New-Button 72, 606, 608
New-CheckBox 610
New-ComboBox 610
New-Elippse 611
New-Event 326
New-EventLog 284, 493–494
New-GPLink 575

New-GPO 573
New-GPStarterGPO 574
New-Grid 608
New-Hardlink 391–392
New-HardwareProfile 358
New-Image 611
New-Int64Animation 611
New-Item 185, 188, 377, 404, 439
New-Itemproperty 440
New-Junction 392
New-Label 603, 608
New-Line 611
New-ListBox 610
New-Mailbox 579
New-Mailboxdatabase 579
New-MediaElement 611
New-Menu 610
New-Module 339
New-Object 215–218, 223, 714
New-PasswordBox 608
New-ProgressBar 610
New-PSDrive 191, 440, 455, 558
New-PSSession 287–288, 292, 295–296, 299, 302
New-RadioButton 610
New-Rectangle 611
NewRichTextBox 610
New-ScrollBar 610
New-Service 481
New-Shortcut 390
New-Slider 610
New-StatusBar 610
New-Storagegroup 579
New-Storyboard 611
New-Symlink 393
New-TextBlock 610
New-TextBox 608
New-TimeSpan 139
New-TreeView 610
New-UrlShortcut 391
New-ViewBox 611
New-VirtualDVDDrive 358
New-VirtualNetworkAdapter 358
New-VM 358
New-WebApplication 590
New-WebAppPool 588
New-WebServiceProxy 491–492
New-WebserviceProxy 492
New-Website 585
New-WebVirtualDirectory 589
New-Window 608
Node 411
Non-Terminating Error 159, 661, 663
Notation 702
  umgekehrt polnische 543
NoteProperty 273, 406
Notes 627
Notizeigenschaft 90, 93, 99
-notmatch 131
Novell 527
NT Event Log Event Consumer 255
NTAccount 504, 506, 508
NtAccount 508
NTFS 237
NTLM 283
NTSecurityDescriptor 534
$null 103, 111, 150, 153, 181

O

Object 703
Object Linking and Embedding Database 428
Object[] 156
ObjectCategory 534, 548, 565
ObjectClass 534, 548, 560, 565
ObjectGUID 535, 560
ObjectSecurity 503
ObjectSecurityDescriptor 534
ObjectSid 534
ObjectVersion 535
Objekt 674, 697–698, 700–702
  .NET 601
  Dynamisch 273
  WMI 228
Objektadapter 96, 261–262, 420
Objektassoziation
  WMI 235
Objektbaum 701
Objektidentifikation
  ADSI 526–527
Objektmenge 383
Objektorientierung 75, 697, 706, 708
Objekt-Pipeline 383
Objekttyp 698
ODBC 417–418
Office 561
$OFS 124
OLEDB 417–418, 428, 523
  Provider 523, 544
OleDbCommand 426, 432
OleDbConnection 424, 428, 432
OleDbDataAdapter 432
OLEDB-Provider 432
On_Click 609
ONELEVEL 544
OneLevel 564
Open Database Connectivity
  Einstellung 227
Open() 424
Operator 98, 140, 143
Oracle 418

OracleCommand 426
OracleConnection 424
Ordner 54, 66, 188, 382–383
  Dateisystem 227
Organisationseinheit 548
  anlegen 542
Out-Default 167, 170, 680, 682
Out-File 181
Out-GridView 167, 169
Out-Gridview 37
Out-Host 167, 173–174
Outlook 360
Out-Null 167, 181
Out-Printer 181, 451

P

Page File 361
Panel 608
-Parameter 72
Parameter 54, 157, 627, 649, 658
  Funktion 157
  Skript 110
Parameterliste 111
ParentSession 296
PascalCasing 711
Performance Counter Provider 497
Performance Monitor 226, 237
PERL 117
Pfad 233
  ADSI 526
  Verzeichnisdienst 526
  WMI 230, 232–233
Pfadangabe 188
Pfeilspitze 702
Pflichtparameter 41
PHP 117
PhysicalDeliveryOfficeName 536, 561
Ping 64, 227, 486–487, 661
Ping-Host 44, 486
Pipeline 31, 49, 75, 84–85, 87, 102, 181, 268
  Ausgabe 643
  Eingabe 651
Pipeline Processor 81, 634
Pipelining 75, 185
Plattform Invoke 279
Plattformunabhängigkeit 707
Platzhalter 175
Polymorphismus 704
PoshConsole 204
Position 625, 649
Postfach 579
Postfix-Notation 543
Power Management 237
PowerGadget 345
PowerGUI 197
PowerShell 31, 56, 75
  Editor 193
  Extension 355
  Hosting 31, 613
  Konsole 193
  Laufwerk 185, 191, 440
  Protokollierung 307
  Remoting 210
  Sicherheit 112
  Skriptsprache 117
PowerShell Analyzer 205
PowerShell Community Extensions 349, 517
PowerShell Plus 200, 656
PowerShell.exe 343
PowerShellPlus 197, 200, 211
PowerTab 210
PrimalScript 206, 211
Principal 504
Privileg 249
Process 84, 99, 385, 655
ProcessRecord() 633, 638, 649
Professional Developer Conference 32
profile.ps1 45, 218, 616, 658
ProfilePath 561
Profilskript 197
Programmgruppe 227
Programmiersprache 150
Programmiersprachenunabhängigkeit 707
Projektvorlage 632
Prompt 365
Propagation Flags 500
Property 697
Property 90, 712
Property Cache 531
PropertyCollection 522
PropertyDataCollection 259, 261
PropertyGrid 601
PropertyNames 522
PropertyValueCollection 522, 530
ProtectedFromAccidentalDeletion 561, 564
Protokolldatei 255
Provider 187
  ADO.NET 417
  Dateisystem 377
  PowerShell 186
  Verzeichnisdienst 517
  WMI 228
Proxy 492
Prozedur 153
Prozess 104, 227, 235
  auflisten 181, 477
  beenden 479
.ps1 61
PSBase 525, 531, 533

PSCmdlet 633
PSCodeGen 353
PSComputerName 294
PSCredential 184, 478
PSCustomObject 99, 275, 406
psd1 328–329
PSDiagnostics 346, 348
PSDriveInfo 378
$PSHome 124
$PSHost 124
PSHost 678
PSHostRawUserInterface 679–680
PSHostUserInterface 679–680
PSImageTools 353
PSModulePath 340, 686
PSObject 96, 682–683
PSRemotingJob 314
PSRSS 353
PSSession 294
PSSnapIn 635
PSSystemTools 353, 378, 450
PSUserTools 353
PSVariable 120
public 649
Python 117

Q

Quantifizierer 133
Quantor 133
Quest 345, 356

R

Read-Host 183
Receive-Job 313–315
Rechenleistung 104
recurse 55
REFERENCES OF 250
Refresh() 703
RefreshCache() 531
Regel 470
Register-Event 323
Register-ObjectEvent 401
Register-PSSessionConfiguration 295, 298
register-psSessionConfiguration 298
Register-WMIEvent 320
Registrierungsdatenbank 31, 185–186, 190–191, 227, 439
  Schlüssel 439
Registry 235, 237, 439
RegistryKey 455, 503
RegistrySecurity 504
RegistryValueChangeEvent 253, 318
Regulärer Ausdruck 131
Relative Distinguished Name 532–533
Remote Desktop Service 346
Remote Procedure Call 284
Remote Server Administration Tools 346, 556
Remoting 110
Remove_DirectoryEntry 665
Remove-ADGroup 568
Remove-ADGroupMember 568
Remove-ADObject 562, 564
Remove-ADOrganizationalUnit 565
Remove-ADUser 566
Remove-Buchung 672
Remove-Computer 447
Remove-DirectoryEntry 355, 656
Remove-Event 322
Remove-EventLog 284, 493
Remove-GPLink 575
Remove-GPO 573
Remove-GPPrefRegistryValue 578
Remove-GPRegistryValue 578
Remove-Item 377, 384, 440
Remove-ItemProperty 441
Remove-Job 313, 316
Remove-LDAPObject 551, 620
Remove-Module 339, 344
Remove-PSSession 293, 295–296
Remove-Variable 123
Remove-WebApplication 591
Remove-WebAppPool 591
Remove-Website 591
Remove-WebVirtualDirectory 591
Remove-WindowsFeature 459, 465
Remove-WmiObject 259, 269
Rename-ADObject 562
Rename-Computer 447
Rename-Drive 382
Rename-GPO 573
Rename-Item 383
Replikation 226
Repository 235
Resolve-Assembly 219
Resolve-Host 488
Resolve-Path 189
ResponseHeaders 216
Restart-Computer 284, 448, 465
Restart-Service 302, 304, 481, 483
Restore-ADObject 562
Restore-Computer 448
Restore-GPO 574
Restricted 113
Resume-Service 481, 483
return 145, 190, 634
Richtlinienergebnisbericht 577
Rolle 460
Rollendienst 460
rootcimv2 234

RSS 353, 490–491
Rückgabeobjekt 643
RuleCollection 472
RuleCollections 472
Runspace 205
RuntimeException 163

S

SAM 527
SAMAccountName 534, 538, 543, 561
Sapien 206, 209, 359–361
Schablone 698–699
Schalter 55, 675
Schema 532, 559
  Active Directory 537
  WMI 235
Schemaabfrage 251
SchemaNameCollection 523
SchemaNamingContext 557
Schleife 149
Schlüssel 185
Schlüsselattribut
  WMI 229
Schnittstelle 421, 700, 704
  .NET 714
Scripting.FileSystemObject 223
ScriptMethod 273
SDDL 516
sealed 703
SearchScope 564
Security Descriptor 500
Security Descriptor Definition Language 297
Security Identifier 499, 504, 506, 508
Security Service Provider 249
SELECT 250, 317
  WQL 250–251, 253
Select
  PowerShell 99
SelectNodes() 409
Select-Object 41, 84–85, 92, 96, 99, 101, 105, 173, 267, 409, 658
SelectSingleNode() 409
Select-String 64, 81, 404
Select-Xml 409
Semaphore 503
Semikolon 105
Send-MailMessage 488–489
Send-SmtpMail 488–489
Serialisierung 88, 290
Seriennummer 445
ServiceController 290
Serviceorientierung 706, 708
Set-Acl 499, 512, 515
Set-ADGroup 568
Set-ADObject 562–563
Set-ADOrganizationalUnit 565–566
Set-ADUser 566
Set-Alias 61, 110
Set-AppLockerPolicy 471
Set-AuthenticodeSignature 114
Set-BPAResult 597
Set-Clipboard 602
Set-Content 377, 403–404, 490
Set-DataRow 435
Set-DataTable 355, 435
Set-Date 139
Set-DistributionGroup 580
Set-ExecutionPolicy 109, 113
Set-Executionpolicy 113
Set-FileTime 388
Set-GPInheritance 578
Set-GPLink 575
Set-GPPermissions 578
Set-GPPrefRegistryValue 578
Set-GPRegistryValue 578
SetInfo() 521, 531
Set-Item 302, 377
Set-ItemProperty 388, 441
Set-Location 56, 185, 188, 377, 439
Set-Mailbox 580
Set-Methode 91
Set-PSDebug 122, 306–307
Set-PSSessionConfiguration 295
Set-Service 481, 484
Set-StrictMode 123
Setter 91, 713
Set-TraceSource 308
Set-Variable 120, 124, 607
Set-VolumeLabel 382
Set-WmiInstance 259, 269
Shell 31, 75
Shell.Application 394
ShouldProcess() 665–666
Show-EventLog 284, 494
Show-Service 284
Sicherheit
  COM 248
  Dateisystem 227, 237
  PowerShell 112
  WMI 248
Sicherheitsabfrage 665
Sicherheitsbeschreibung 500
Sicherheitseinstellung 499
Sicherheitsmodell 31
Sicherheitsrichtlinie 114
SID 499
Side-by-Side Executing 708
SilentlyContinue 162
Simple Network Management 226, 237

Simple Object Access Protocol 283
Sitzung 294–297
Skript 107–108, 117
  PowerShell 107
Skriptblock 606
Skriptdatei 107
Skripteigenschaft 90, 93
Skriptmodul 685
Smoking Man 518
SMTP 488–489
SmtpClient 488
SNA Server 237
Snap-In 213, 339, 343, 631, 633–636, 658, 668
Snippet 200
SOAP 236
Software 226–227, 361
  deinstallieren 457
  installieren 456
  inventarisieren 453
  verwalten 453
Software Restriction Policy 470
Softwareentwicklungsplattform 708
Sortieren 100
Sort-Object 84–85, 96, 100, 105, 156, 634
Speicher 78
Speicherbereinigung 708
Speicherverbrauch 422
Speicherverwaltung 708
Spitzname 699
Split 117, 144
–split 117, 144
Split-String 130
Sprache 328
Sprachkürzel 328
SQL 250
SqlCommand 426
SqlConnection 424, 428
SqlDataSourceEnumerator 419
SqlServerCe 418
STA 602
StackPanel 608
$StackTrace 124
Stammzertifizierungsstelle 115
Standarddrucker 181
Start-Job 313–314
Startmenü 227
Start-Process 477–478
Start-PSSession 316
Start-Service 291, 481, 483
Start-Sleep 116
Start-Trace 349
Start-Transaction 309, 311
Start-Webitem 590
Start-Website 590
Stop 162
Stop-Computer 284, 448
Stop-Job 313, 315
Stop-Process 79, 477, 479, 655
StopProcessing() 633
Stop-Service 481, 483
Stop-Trace 349
Stop-Webitem 590
Stop-Website 590
Stored Procedure 431
Streaming 82
StreetAddress 561
SUBTREE 544
SubTree 564
Suche
  Active Directory 543
  Assembly 389
  LDAP 523
  Textdatei 404
  Verzeichniseintrag 533
  XML 409
SupportsShouldProcess 665
Surname 561
Suspend-Service 481, 483
switch 145, 151
SwitchParameter 675
Symbolic Link 391, 393
Synopsis 627
System 710–711
System ACL 504
System Center Virtual Machine Manager 358
System Management Server 237
System.ApplicationException 163
System.Boolean 189
System.Collections.Hashtable 141
System.Console 217
System.Data 216
System.Data.Odbc 418
System.Data.OLEDB 418
System.Data.OleDb 418, 425
System.Data.OracleClient 418, 425
System.Data.SqlClient 418, 425
System.Data.SqlServerCe 418
System.DateTime 120, 138
System.Diagnostics.EventLog 493
System.Diagnostics.Process 76, 79, 87, 171, 477, 654
System.DirectoryServices 524
System.DirectoryServices 216, 517–519, 521, 524, 528, 534, 537, 544, 569
System.DirectoryServices.ActiveDirectory 569
System.Directoryservices.DirectoryEntry 217
System.dll 711
System.Enum 220
System.Environment 288, 445, 637–638, 643
System.Globalization.CultureInfo 365

System.Int32 120, 126
System.IO.Directory 503
System.IO.DirectoryInfo 273, 382
System.IO.DriveInfo 219, 378, 380
System.IO.DriveType 219
System.IO.File 503
System.IO.FileInfo 273, 382, 384
System.Management 216, 645
System.Management-.Automation 682
System.Management.Automation 72, 633, 635, 645, 678
System.Management.Automation.Cmdlet 634
System.Management.Automation.PathInfo 189–190
System.Management.Automation.PSCustomObject 406
System.Management.Automation.PSDriveInfo 378
System.Management.ManagementObject 139, 267
System.Net.WebClient 216, 490
System.Object 87–88, 275, 654, 658
System.Reflection 218
System.Security 504
System.Security.AccessControl 503
System.ServiceProcess.ServiceController 87, 481–482, 654
System.String 128, 637
System.TimeSpan 139
System.Type 86, 123, 179
System.Windows 603
System.Windows.FontStyle 605
System.Windows.Forms 218–219, 599
System.Xml.Node 411
System32 105
Systemattribut
  WMI 230
Systemdienst 80, 227, 481
  auflisten 251
  überwachen 253
SystemEvent 318
Systemklassen
  WMI 229
Systemmanagement 225
SystemParametersInfo 279
Systemwiederherstellung 448

T

Tab Completion 195
Tabellenform 179
TabPanel 608
Tabulatorvervollständigung 195
TAR 393
TaskScheduler 353
TechNet 67
Tee-Object 103
Telnet 288
Terminal Services 227
Terminating Error 159, 661
Test-32Bit 450
Test-64Bit 450
Test-AppLockerPolicy 471, 473
Test-Assembly 388
Test-Connection 486–487
Test-DbConnection 435
Test-ModuleManifest 340
Test-Path 189
Test-ServiceHealth 579
Test-UserGroupMembership 542
Test-Xml 407–408
Textanzeige 610
Textdatei 105, 403
$this 276, 607
Thread-Modell 602
throw 145, 163
ThrowTerminatingError() 661
TimeSpan 308
Ton 217
ToString() 87–88, 654, 703
TotalProcessorTime 92, 180
Trace-Command 372
Transaktion 309
Transformation 611
Translate() 508
Trap 159, 164
trap 145, 159, 163
Treiber 361
Troubleshooting Pack 346, 593
$true 124
true 450
Trusted Host 302
Trustee 500
Try-Catch-Finally 159, 164
Typ 711
  Namensgebung 711
Typbezeichner 120
types.ps1xml 62, 94, 96, 267
Typisierung 120

U

Überladung 158
Uhrzeit 138
Umgebungsvariable 186, 226
Umlaut 404
Undefined 113
Undo-Transaction 309, 311–312
UniformGrid 608
Universal Coordinated Time 231, 264
Unix 31, 75, 117, 377, 391

Unregister-PSSessionConfiguration 295, 299
Unrestricted 113
Unterbrechungsfreie Stromversorgung siehe USV
Unternamensraum 709
Unterordner 87, 383
Unterroutine 153
Unterschlüssel 185
until 145
Update 227, 361
UsePropertyCache 531
User 524, 534, 548
user32.dll 279
UseTransaction 310
USV 451
UTF8 404

V

ValidateCount 625
Validate-CustomerID 625
ValidateLength 625, 675
ValidateNotNull 625, 675
ValidatePattern 625, 675
ValidateRange 625
ValidateScript 625
ValueFromPipeline 624, 652–653, 658
ValueFromPipelineByPropertyName 624, 653
ValuesCollection 522
Variable 86, 102–103, 119, 123, 174, 185–186
  Auflösung 125
Variablenauflösung 125, 174
Variablenkennzeichner 103, 119
VB 277, 279
Verbindungszeichenfolge 424, 431
Verbose 305–306, 661
VerbosePreference 661
VerbsCommon 674
VerbsCommunications 674
VerbsData 674
VerbsDiagnostic 674
VerbsLifeCycle 674
VerbsSecurity 674
Vererbung 235, 702, 714
Vererbungsdiagramm 703
Vererbungshierarchie 257, 559, 703
  WMI 235
Vergleich 104
Vergleichsoperator 96
Verifikation 708
Verknüpfung 390
Verzeichnisattribut 530
Verzeichnisdienst 95, 236, 355, 532, 544
Verzeichnisdienstklasse 532
Verzeichnisobjekt 528, 534
Verzweigung 103
Video 611
View 172
VirtualizingStackPanel 608
Virtuelle Maschine 359
Virus 114
Visual Basic 277, 705
Visual Basic .NET 631–632
Visual Basic 6.0 708
Visual Studio 614, 631–632, 658
Visual Studio 2005 631, 635
Visual Studio 2008 631, 635
Visual Studio 2010 631
Visual Web Developer Express 632
void 181
VS Command Shell 210

W

Wait-Job 313, 315
Wait-Process 480
Walter Skinner 518
WBEM 225
Web Administration 346
Web Based Enterprise Management 225–226
Web Service Description Language 492
Webanwendung 708
Weblog 490
Webserver 586
Webservices 491
Website 586, 591
Well-Known GUID 528
Well-Known Object 528
Well-Known Security Principal 508
WellKnownSidType 509
Werkzeug 193
Wertemenge 140
WhatIf 305, 384, 665–666
WHERE 250
Where-Object 56–57, 76, 80, 84–85, 96, 102–103, 105, 481, 634, 645–646, 658
while 145
Whistler 232
Width 179
Wiederherstellungspunkt 448
Win32 228
Win32_Account 519
Win32_ACE 398
Win32_Battery 451
Win32_Bios 446
Win32_BootConfiguration 446
Win32_CDRomDrive 267
Win32_CDRomdrive 449
Win32_CodecFile 455
Win32_ComponentCategory 251
Win32_ComputerShutdownEvent 253, 318

Win32_Computersystem 445
Win32_Currenttime 139
Win32_currenttime 139
Win32_Desktop 519
Win32_Diskdrive 449
Win32_Group 519
Win32_Keyboard 449
Win32_LocalTime 139
Win32_LocigalDisk 381
Win32_LogicalDisk 234, 251, 263, 378, 380, 645
Win32_MappedLogicalDisk 382
Win32_MemoryDevice 449
Win32_NetworkAdapter 449
Win32_NetworkAdapterConfiguration 251, 485
Win32_NTLogEvent 251, 253, 319
Win32_OperatingSystem 445
Win32_OSRecoveryConfiguration 446
Win32_PerfRawData 497
Win32_PerfRawData_PerfOS_Processor 497
Win32_PerfRawData_PerfProc_Process 497
Win32_PingStatus 486–487
Win32_PointingDevice 449
Win32_PowerManagementEvent 318
Win32_Printer 451
Win32_Printjob 451
Win32_Process 319
Win32_Processor 449
Win32_ProcessStartTrace 318
Win32_Product 453, 456–457
Win32_Quickfixengineering 455
Win32_SecurityDescriptor 398
Win32_Service 251, 253, 319
Win32_Share 395–396
Win32_SoundDevice 449
Win32_SystemConfigurationChangeEvent 253, 318
Win32_Tapedrive 449
Win32_TCPIPPrinterPort 452
Win32_Trustee 398
Win32_USBController 449
Win32_UserAccount 519
Win32_VideoController 264, 449
Win32_Videocontroller 267
Win32_WindowsProductActivation 446
Win32-API 279
$window 607
Windows 2000 232
Windows 7 196, 345
Windows 9x 236
Windows Communication Foundation 227
Windows Driver Model 237
Windows Firewall 361
Windows Forms 599, 601
Windows Installer 237, 470
Windows ME 236
Windows PowerShell 31
Windows PowerShell Community Extensions 349
Windows Presentation Foundation siehe WPF
Windows Remote Management 37, 236, 283, 285–287, 313
Windows Script Host 50, 110, 112
Windows Server 2003 32, 37, 232, 236, 544
Windows Server 2008 37, 196
Windows Server 2008 R2 196, 345
Windows Server Backup 348
Windows Troubleshooting Platform 593
Windows Vista 37, 196, 393, 705
Windows XP 49, 232, 255
WinMgmt.exe 236
WinRM 236
WITHIN 250, 318
WKGUID 528
WMI 31, 139, 225, 228, 234, 260, 283, 645, 649, 661
  Class Explorer 257
  Command Shell 49
  Data Query 250
  Ereignis 253
  Event Query 251, 255
  Klasse 257
  Namespace 232
  Object Browser 256–257
  Query Language 250, 255, 265
  Repository 215, 235, 254, 271
  Schema 235, 251
  Schema-Query 251
  Steuerung 235
WMI Object Browser 256
WMI Query Language 317
WMICLASS 225
WMIClass 260
WMISEARCHER 225, 265–266
WorkingSet 95
WorkingSet64 76
World Wide Wings 672
WOW64 38
WPF 599, 602
WPF PowerShell Kit 353, 602–603
WPK 353, 602–603
WQL 250, 645
WrapPanel 608
Write-BZip2 393
Write-Clipboard 602
WriteDebug() 661
Write-Error 174
WriteError() 661, 663
Write-EventLog 284, 493, 495
Write-GZip 44, 393
Write-Host 174
WriteObject 638, 647
WriteObject() 634, 645
Write-Tar 393

WriteVerbose() 661, 663
Write-Warn 174
WriteWarning() 661, 663
Write-Zip 393–394
WScript.Shell 390
WSH 525
WSMan 186, 303
WS-Management 283–284, 286, 303
Wurzelnamensraum 709
www.IT-Visions.de 345, 355, 378, 435

X

XAML 611
XCopy-Deployment 709, 711
XFilesServer 518
XML 335, 406, 411, 413, 668
Xml 407
XML Application Markup Language siehe XAML
XmlAttribute 409
XmlDocument 413
XmlElement 409
XML-Schema 408
XML-Webservice 491, 709
XPathDocumentNavigator 409
XslCompiledTransform 413

Y

Year 78

Z

Zahl 126
Zahlenliteral 126
Zeichenkette 125, 128, 130, 175, 655, 674
  Operation 129
  trennen 130
  verbinden 130
Zeichensatz 404
Zeitmessung 308
Zeitplandienst 227
Zertifikat 114
Zertifikatsspeicher 31, 185–186
Zertifikatsverwaltung 115
ZIP 393
Zufallszahl 127
Zugriffsrechteliste 499, 504
Zuweisungsoperator 143
Zwischenablage 602
Zwischencode 707
Zwischenschritt 102
Zwischenspeicher 422