



Günter Born

Inside Windows Scripting Host

Windows-Scripting für Power-User, Programmierer und Administratoren

Leseprobe

Günter Born: Inside Windows Scripting Host
Microsoft Press Deutschland, Edisonstr. 1, 85716 Unterschleißheim
Copyright © by Microsoft Press Deutschland

Das in diesem Buch enthaltene Programmmaterial ist mit keiner Verpflichtung oder Garantie irgendeiner Art verbunden. Autoren, Übersetzung und der Verlag übernehmen folglich keine Verantwortung und werden keine daraus folgende oder sonstige Haftung übernehmen, die auf irgendeine Art aus der Benutzung dieses Programmmaterials oder Teilen davon entsteht.

Das Werk einschließlich aller Teile ist urheberrechtlich geschützt. Jede Verwertung außerhalb der engen Grenzen des Urheberrechtsgesetzes ist ohne Zustimmung des Verlags unzulässig und strafbar. Das gilt insbesondere für Vervielfältigungen, Übersetzungen, Mikroverfilmungen und die Einspeicherung und Verarbeitung in elektronischen Systemen.

ISBN 3-86063-495-X
© Microsoft Press Deutschland 1999
Alle Rechte vorbehalten

Born: Inside Windows Scripting Host

	Vorwort	1
0		3
	Einleitung	3
	Zur Benutzung dieses Buches	3
	Noch ein paar Hinweise	5
1		7
	Einführung in den WSH	7
	Was ist der Windows Scripting Host (WSH)?	7
	Wozu braucht man den WSH	8
	Der WSH als Lösung	8
	Was kann man mit dem WSH tun?	11
	Welche Sprachen unterstützt der WSH?	11
	Was ist bei WSH-Skripten anders als bei VB/VBA und JavaScript?	12
	Skripte erstellen und ausführen	13
	Das erste Skript erstellen	13
	Gleiches Beispiel aber in JScript	15
	WSH-Skripte ausführen	16
	Skript unter Windows ausführen	16
	Skript mit Parametern starten	17
	Skripte unter MS-DOS ausführen	21
	Parameterübergabe beim Aufruf von cscript.exe	22
	Skripteigenschaften festlegen	24
	Hinweise zu den Windows-Skriptbeispielen	26
2		29
	Entwicklungshilfen für WSH-Skripte	29
	Editoren zur Skriptentwicklung	30
	Wie wird ein Skriptprogramm erstellt?	30
	So läßt sich eine Skriptdatei erstellen	32
	Eine Vorlage für Skriptdateien	33
	So können Sie Skriptdateien laden	35
	So richten Sie neue Befehle zum Bearbeiten ein	35
	Hilfen zur Skriptbearbeitung	37
	M&I WinEditor	38
	EditPlus	38
	Microsoft Word 97 als Skripteditor	39
	Die Dokumentvorlage <i>Script.dot</i>	40
	Weitere Editiermöglichkeiten	42
	Hilfen zur Skriptentwicklung	42
	OLE/COM Object Viewer	43
	Unterstützung bei der VBScript-Codeeingabe	44
	Wo gibt's Infos zu Automatisierungsobjekten	47
	Der Objektkatalog, das Fenster zu Automatisierungsobjekten	48
	So beeinflussen Sie die Anzeige des Objektkatalogs	50
	ActiveX-Module installieren/deinstallieren	52
	ActiveX-Objekte installieren	53
	Installation per Webseite	53
	So sorgt der Webautor für die Installation des ActiveX-Moduls	53
	ActiveX-Komponente per Entwicklungsumgebung registrieren	55
	So können Sie eine lokale OCX-Datei auch registrieren	56
	Die Registrierung für eine OCX-Datei entfernen	57
	ActiveX-Komponenten entfernen: Methode 1	57
	ActiveX-Komponenten entfernen: Methode 2	58
	Skriptdateien testen	59
	So zeigt der WSH auftretende Fehler	59
	Programmablauf verfolgen	60
	Test mit dem Microsoft Script Debugger	63
	Vorbereiten des Skripts zum Debuggen	63

Trick: So geht's noch einfacher	64
Das Skript zum Testen starten	65
Befehle des Debuggers	68
Die Anweisungen schrittweise ausführen	69
Den Ablauf gezielt unterbrechen	69
Haltepunkte setzen	70
Anzeige der Aufrufeliste	70
Werte anzeigen	71
3	73
Einführung in VBScript	73
Anweisungen, Folgezeilen und Kommentare	74
VBScript-Anweisungen	74
Fortsetzungszeilen	74
Kommentare	75
Hinweise zur Struktur eines VBScript-Programms	75
Variablen und Konstanten	78
Konstanten	78
Vordefinierte Konstante	79
Variablen	79
Hinweise zu VBScript-Datentypen	80
Variant Subtypen	81
Variablendeklaration mit <i>Option Explicit</i> erzwingen	82
Einsatz der Dim-Anweisung	85
Public und Private in der Variablendeklaration	85
Felder mit DIM deklarieren	86
Variablenamen	87
Operatoren	87
Arithmetische Operatoren	87
Zuweisung von Objektreferenzen mit <i>Set</i>	88
Logische Operatoren	88
Vergleichs-Operatoren	91
Prioritäten von Operatoren	92
Kontrollstrukturen	93
IF ... Then	93
IF ... Then ... Else	93
IF ... Then ... ElseIf	94
Select Case	94
Schleifen	95
Do While ... Loop	95
Do Until ... Loop	96
Do ... Loop While	96
Do ... Loop Until	96
Exit DO	97
For ... Next	97
For Each... Next	98
Exit For	98
While... WEnd	98
Prozeduren und Funktionen	98
Funktionen	99
Gültigkeit von Funktionen	100
Funktionsargumente mit ByRef/ByVal übergeben	101
Interne Funktionen	101
Prozeduren	102
Anmerkungen zur Parameterübergabe (ByRef, ByVal)	103
4	107
Einführung in JScript	107
Grundlagen und Hinweise	107
Was ist JScript?	107
Die Struktur eines JScript-Programms	108

Kommentare	110
Hinweise zu JScript-Anweisungen.....	110
Fortsetzungszeilen	110
Mehrere Anweisungen pro Zeile.....	111
Konstante.....	111
Variablen	112
Bemerkungen zur Gültigkeit von Variablen	112
Variablennamen.....	113
Hinweise zu Werten und Datentypen.....	115
Hinweise zu Untertypen.....	116
Spezielle Zeichen in Zeichenketten	116
Ausdrücke und Operatoren.....	117
Zuweisungsoperator.....	117
Vergleichsoperatoren.....	117
Berechnungsoperatoren	118
Inkrement- und Dekrement-Operatoren	119
Logische Operatoren.....	120
Kontrollstrukturen	121
if-Anweisung	121
Conditional-Operator.....	122
for-Schleife	123
For in-Schleife	124
while-Schleife	125
do while-Schleife	125
switch-Anweisung.....	126
break- und continue-Anweisungen	127
Built-In Objekte und Funktionen.....	127
Funktionen.....	128
Built-in Funktionen.....	128
Objekte	129
5	131
Einstieg in die WSH-Skriptprogrammierung.....	131
Objekte, Eigenschaften und Methoden	131
Etwas Theorie	132
... und nun zur Praxis	133
Welche Objekte stehen zur Verfügung?	135
Wo erhalten Sie Informationen über Objekte und Eigenschaften?	136
Dialoge im WSH anzeigen	136
Nutzen der Echo-Methode.....	136
Die Echo-Methode in VBScript nutzen	137
Die Echo-Methode in JScript nutzen	138
So erhalten Sie einen Zeilenumbruch im Dialogfeld	140
VBScript-Ausgaben per MsgBox-Funktion.....	141
Schaltflächenabfrage per MsgBox-Funktion	144
Beispiel 2: VBScript-Dialogfelderanzeige mit Prozeduraufruf	147
Anwenden der Popup-Methode	151
Dialogfelder in JScript ausgeben	151
Klappt Popup auch in VBScript?.....	155
6	157
Nutzen der WSH-Objekte, Methoden und Eigenschaften	157
Arbeiten mit dem WScript-Objekt.....	158
Anzeige der WSH- und Skript-Eigenschaften	158
Die Lösung in VBScript	160
Die Eigenschaften in JScript anzeigen.....	161
Eigenschaften der Script-Engine anzeigen.....	162
Auswerten der Skript-Parameter.....	164
Die komplette Lösung in VBScript.....	166
Parameteranzeige in JScript.....	168
Zugriff auf Umgebungsvariablen.....	169

Allgemeine Vorbemerkungen	169
Zugriff auf Umgebungsvariable per Skript	170
Umgebungsvariablen in VBScript nutzen	171
So lesen Sie Umgebungsvariablen in JScript	174
Umgebungsvariablen setzen	175
Löschen von Umgebungsvariablen	177
Umgebungsvariablen expandieren	180
Objekte holen und freigeben	182
Objekte instanziiieren	182
Hinweise zu Objektereferenzen	184
CreateObject oder GetObject anwenden?	185
Freigabe mit DisconnectObject	187
Programmaufrufe per WSH	188
Hinweise zur <i>Run</i> -Methode	188
Aufrufen einer Windows-Anwendung	190
Den Editor aus VBScript aufrufen	191
Den Windows-Rechner aus JScript aufrufen	191
Warten auf den Prozeß und Auswerten des Rückgabecodes	193
MS-DOS-Befehle per RUN ausführen	195
7	197
Benutzerdialoge und Formulare	197
Benutzereingaben im Skripten	198
Benutzereingabe in VBScript	198
Beispiel: Benutzereingabe mit wählbarer Sprache	201
Problem: Benutzereingaben in JScript	205
JScript: Benutzereingabe per Internet Explorer	206
Wie läßt sich eine InputBox-Funktion in einem HTML-Dokument realisieren?	207
Wie läßt sich die Funktion im HTML-Dokument aufrufen?	208
Lösung 1: <i>InputBox</i> -Implementierung für JScript	209
ActiveX-Modul für JScript-Benutzereingaben	215
Formulare in Skripten nutzen	218
Vorbemerkungen zur Formulareingabe	218
Internet Explorer zur Formulareingabe	218
HTML-Anweisungen für die Formularausgabe	219
DHTML zur Verwaltung des Formulars	220
Hinweise zu DHTML-Objekten	221
WSH-Skript zur Formulareingabe	224
JScript-WSH-Skript zur Formularabfrage	229
Formulareingabe mit Callback-Funktion	231
About-Formular per ActiveX-Komponente	234
So sieht der Aufruf des Formulars im WSH-Skript aus	238
Embeddialog als Formular	240
Entwurf des Formulars	240
Globale Variablen zum Speichern der Eingabewerte definieren	244
Implementierung der Methoden für die Formularhandhabung	245
So nutzen Sie die ActiveX-Komponente im WSH-Skript	246
Formularaufruf in JScript	248
Dialogfeld zur Kennworteingabe	249
8	255
Erweiterte WSH-Funktionen	255
Umgang mit Verknüpfungen	255
Hinweise zu Verknüpfungen	256
Verknüpfungen mit <i>CreateShortcut</i> anlegen	257
Beispiel: Eine Verknüpfung zur aktuellen Skriptdatei anlegen	257
Die Implementierung in JScript	260
Arbeiten mit dem <i>SpecialFolders</i> -Objekt	262
Anzeige aller vorhandenen <i>Special Folders</i> -Einträge	264
Anzeige der Special Folders-Einträge in JScript	266
Der Pfad eines Spezialordners ermitteln	267

Erstellen einer Verknüpfung auf dem Desktop	269
JScript: Verknüpfung auf dem Desktop anlegen	272
Eine Verknüpfung im Startmenü anlegen	274
Die Implementierung in JScript	278
Verknüpfung mit Parametern anlegen	281
Verknüpfung zu Webseiten anlegen	284
Netzwerkressourcen nutzen	286
Benutzername, Domäne, Computername	286
Zuweisen von Druckern	288
Kontrolle und Verwaltung der Druckerzuordnung	289
Eine Druckerzuweisung in VBScript setzen	289
Auf ein Wort: Laufzeitfehlerbehandlung in VBScript	291
Die JScript-Implementierung der Druckerzuweisung	293
Auflisten aller zugewiesenen Druckerausgänge	295
Zuweisen von Netzlaufwerken	296
Die Implementierung in JScript	299
Umgang mit der Registrierung	301
Vorbemerkungen zur Registrierung	301
Registrierungszugriff in WSH-Skripten	303
Beispiel: Registrierungszugriff in VBScript	305
Beispiel: Registrierungszugriff in JScript	308
Laufzeitfehler in WSH-Skripten abfangen	310
Existenz eines Schlüssels mit <i>KeyExist</i> in VBScript abfragen	313
Nutzen der <i>WSHKeyExist</i> -Methode	315
Nutzung der <i>WSHKeyExist</i> -Methode in VBScript	316
Nutzung der <i>WSHKeyExist1</i> -Methode in JScript	319
Anpassen des Windows-Installationspfads	321
Zuletzt angemeldeten Benutzer ausblenden	323
9	325
Arbeiten mit dem Dateisystem	325
Eine Einführung	326
Das FileSystemObject-Objektmodell	326
Anlegen des FileSystemObject-Objekts	327
Zugriff auf die Methoden des FileSystemObject-Objekts	328
Umgang mit Laufwerken	329
Auflisten aller Laufwerke	329
Anzeige der vorhandenen Laufwerke in VBScript	330
Anzeige der vorhandenen Laufwerke in JScript	333
Auflisten der Eigenschaften eines Laufwerks	335
Laufwerksdaten in JScript abfragen	339
Zugriff auf Dateien und Ordner	341
Anzeige aller Unterorder in einem Ordner	341
Die Lösung zur Ordneranzeige in JScript	343
Ordner anlegen, verschieben und löschen	345
Einen neuen Ordner anlegen	345
Den übergeordneten Ordner ermitteln	347
Einen Ordner umbenennen/kopieren/verschieben	347
Ordner löschen	349
Die Implementierung des Beispiels <i>Folders1.vbs</i>	349
<i>Folder1.js</i> : Die Implementierung in JScript	352
Dateien in einem Ordner auflisten	354
Zugriff auf Dateiattribut und das Dateidatum	357
Datei kopieren und löschen	360
Sicherungskopien mit Skripten erstellen	363
Textdateiinhalte bearbeiten	369
Auslesen einer Textdatei	369
Schreiben in eine Textdatei	372
Anhängen an eine Textdatei	374
Dialoge zur Dateiauswahl	376
Dialog zur Ordnerauswahl nutzen	376

Dialogfeld zur Dateiauswahl.....	379
10	383
ActiveX-Steuerelemente erstellen	383
Einführung in die VB 5 CCE.....	383
Was ist die VB 5 CCE?	384
Alternativen zur ActiveX-Programmierung.....	384
Anlegen eines ActiveX-Projekts.....	385
Kurzübersicht über die Entwicklungsumgebung.....	386
Was sind Projekte, Klassen Module und Eigenschaften?	387
Das Codefenster.....	389
Das Eigenschaftenfenster.....	391
Steuerelemente in eine Formular einfügen.....	392
Das OCX-Modul der ActiveX-Komponente erzeugen	394
Eine Installationsdatei für die ActiveX-Komponente erstellen.....	396
Informationen über Objekte abrufen.....	397
Objektbeschreibung im Objektkatalog erstellen	399
Ein Beispiel für ein Projekt.....	399
ActiveX-Beispiele	401
Implementierung der <i>WSHKeyExist</i> -Methode	401
Wartefunktion in Skripten	403
Ausgabe von Soundereignissen	407
WSHShell- und WSHAppActivate-Methoden.....	412
Implementierung der <i>WSHShell</i> -Methode	412
Implementierung der <i>WSHAppActivate</i> -Methode.....	414
Das <i>SendKeys</i> -Problem.....	414
Eine <i>WSHSendKeys</i> -Methode implementieren	417
Hinweise zur <i>WSHSendKeys</i> -Methode	417
Implementierungshinweise zu <i>WSHSendKeys</i>	420
Erste Anwendung der <i>WSHSendKeys</i> -Methode.....	420
Anlegen eines neuen Benutzers	423
Direkter Aufruf von Funktionen der Systemsteuerung	426
Fenster manipulieren.....	431
Die <i>WSHFindWindow</i> -Methode	432
Die <i>WSHMoveWindow</i> -Methode	432
Die <i>WSHShowWindow</i> -Methode	433
Die <i>WSHSetForegroundWindow</i> -Methode.....	434
Die <i>WSHFlashWindow</i> -Methode.....	434
Anwendung der Methoden zur Fenstermanipulation	435
Exit - Schnellausstieg aus Windows	438
11	441
WSH-Erweiterungen mit Microsoft Office.....	441
Umgang mit Microsoft Excel.....	441
Das Excel 97-Objektmodell.....	442
Beispiel zum Zugriff auf Excel.....	443
Festlegen der Fenstereigenschaften.....	443
Excel-Tabelleninhalte erstellen.....	447
Excel-Beispiel: WSH-Informationen in Tabelle schreiben.....	452
Excel-XLS-Datei laden und Daten lesen	456
Umgang mit Microsoft Word	459
Das Word 97-Objektmodell.....	459
Beispiel zum Zugriff auf Word.....	460
Ein Word-Dokument erstellen und beschreiben	464
Word-DOC-Datei laden und Daten lesen	469
Word <i>Öffnen</i> -Dialog per Skript aufrufen.....	472
Umgang mit Access	475
Das Microsoft Access 97-Objektmodell.....	475
Das DBEngine-Objektmodell.....	475
Das Workspace-Objekt.....	476
Das Database-Objekt	477

Zugriff auf Access	477
12	485
WSH-Fragen, Tips und Antworten	485
Programmierung	485
Wie kann ich Debuggen?	486
Fehlerbehandlung in Skripten	486
WSH- und Script-Engine-Eigenschaften	487
Callback-Funktionen nutzen	487
So geht's in VBScript	488
So geht's in JScript	490
Skriptaufrufe und Argumente	492
Ein Skript per Drag-und-Drop starten	492
WSH-Skriptaufruf per Windows NT-Scheduler	493
Zugriff auf die Aufrufparameter des Skripts	493
Aufrufen fremder Anwendungen	494
Lange Dateinamen in Skripten	494
Nutzen der Run-Methode für Systemaufrufe	495
Benutzerdialoge und Ausgaben	495
Benutzerausgaben in Skripten	495
Zeilenwechsel und Tabulatoren	496
Benutzereingaben in Skripten	497
Dateibehandlung	497
Feststellen, ob eine Datei/ein Ordner existiert	497
Tip: Prüfen, ob eine Access-Datenbank geöffnet ist	498
Kopieren von Dateien	499
Wiedergabe von Sound und Multimediadateien	499
Soundausgabe per WSHPlaySound-Methode	499
Soundausgabe per <i>WSHMCIEExecute</i> -Methode	501
Soundausgabe per MCI-Steuerelement	502
Multimediadateien wiedergeben	504
Verschiedenes	505
Aufruf einer DFÜ-Verbindung	506
Ermitteln der IP-Adresse	507
Anordnen der Desktop-Fenster	509
Ordner als Shell- oder Explorerfenster öffnen	511
Zugriff auf Shell-Dialoge	512
Benutzerdefinierter Dateidialog nutzen	515
Hinweise zur Implementierung der ActiveX-Komponente	517
Abfrage der Systeminformationen	520
A	523
Windows Scripting Host-Objektreferenz.....	523
Das WSH-Objektmodell	524
Das <i>WScript</i> -Objekt	524
Eigenschaften des <i>WScript</i> -Objektes	524
WScript.Application	525
WScript.Arguments	526
WScript.FullName	526
WScript.Name	527
WScript.Path	527
WScript.ScriptFullName	527
WScript.ScriptName	528
WScript.Version	528
Methoden des <i>WScript</i> -Objekts	529
WScript.CreateObject	529
WScript.DisconnectObject	530
WScript.Echo	530
WScript.GetObject	531
Das <i>WshArguments</i> -Objekt	533
Eigenschaften des <i>WshArguments</i> -Objekts	534

WshArguments.Item	534
WshArguments.Count	534
WshArguments.length	535
Das WshShell-Objekt	535
Eigenschaften des WshShell-Objekts	535
WshShell.Environment	535
WshShell.SpecialFolders	537
Methoden des <i>WshShell</i> -Objekts	538
WshShell.CreateShortcut	538
WshShell.ExpandEnvironmentStrings	539
WshShell.Popup	539
WshShell.RegDelete	541
WshShell.RegRead	542
WshShell.RegWrite	542
WshShell.Run	543
<i>WshNetwork</i> -Objekt	544
Die Eigenschaften des <i>WshNetwork</i> -Objekts	544
WshNetwork.ComputerName	545
WshNetwork.UserDomain	545
WshNetwork.UserName	545
Die Methoden des <i>WshNetwork</i> -Objekts	545
WshNetwork.AddPrinterConnection	546
WshNetwork.EnumNetworkDrives	546
WshNetwork.EnumPrinterConnections	547
WshNetwork.MapNetworkDrive	547
WshNetwork.RemoveNetworkDrive	548
WshNetwork.RemovePrinterConnection	548
WshNetwork.SetDefaultPrinter	549
<i>WshCollection</i> -Objekt	549
Die Eigenschaften des <i>WshCollection</i> -Objekts	550
WshCollection.Item	550
WshCollection.Count	550
WshCollection.length	550
<i>WshEnvironment</i> -Objekt	551
Die Eigenschaften des <i>WshEnvironment</i> -Objekts	551
WshEnvironment.Item	551
WshEnvironment.Count	552
WshEnvironment.length	552
Die Methoden des <i>WshEnvironment</i> -Objekts	552
WshEnvironment.Remove	553
<i>WshShortcut</i> -Objekt	553
Die Eigenschaften des <i>WshShortcut</i> -Objekts	554
WshShortcut.Arguments	554
WshShortcut.Description	554
WshShortcut.FullName	554
WshShortcut.Hotkey	554
WshShortcut.IconLocation	555
WshShortcut.TargetPath	555
WshShortcut.WindowStyle	556
WshShortcut.WorkingDirectory	556
Die Methoden des <i>WshShortcut</i> -Objekts	556
<i>WshSpecialFolders</i> -Objekt	556
Die Eigenschaften des <i>WshSpecialFolders</i> -Objekts	557
WshSpecialFolders.Item	557
WshSpecialFolders.Count	558
WshSpecialFolders.length	558
<i>WshUrlShortcut</i> -Objekt	559
Die Eigenschaften des <i>WshUrlShortcut</i> -Objekts	559
WshUrlShortcut.FullName	559
WshUrlShortcut.TargetPath	559

Die Methoden des WshUrlShortcut-Objekts	560
WshUrlShortcut.Save	560
B	561
Literatur	561
Stichwortverzeichnis	

Arbeiten Sie mit Microsoft Windows 98 oder Windows 2000? Dann interessiert Sie bestimmt die Frage, was sich mit dem Windows Scripting Host anfangen läßt. Diese Funktion wurde ja standardmäßig in die beiden oben erwähnten Betriebssysteme integriert. Aber auch Anwender von Windows 95 und Windows NT 4.0 können sich die betreffende Erweiterung kostenlos aus dem Internet von der Microsoft Website (<http://msdn.microsoft.com/scripting>) herunterladen.

Zumindest die Tatsache, dass Sie diese Seite lesen, beweist Ihr Interesse für diese Thematik. Der Windows Scripting Host (in diesem Buch auch als WSH bezeichnet) eröffnet im Betriebssystem gänzlich neue Möglichkeiten. Endlich verfügt Windows über eine adäquate Programmiersprache, mit der Sie bestimmte Aufgaben lösen und automatisieren können. Leider gibt es aber eine riesige Hürde zu nehmen: Windows enthält selbst keine (bzw. kaum) Dokumentation zu diesem Thema. Die Programmierreferenz zu den WSH-Objekten liefert zwar die meisten Informationen, aber die Beispiele helfen kaum beim Einstieg oder Umstieg. Eigentlich muß man bereits alles wissen, bevor man in der Referenz mit Erfolg nachschlagen kann. Weiterhin bietet es sich an, in WSH-Skripten die unter Windows installierten Automatisierungsobjekte zu nutzen. Und hier hilft die Programmierreferenz zum WSH überhaupt nicht weiter.

Das vorliegende Buch entstand aus dem Ansatz heraus, diesen Informationsmangel zu beheben, den Lesern die Möglichkeiten des Windows Scripting Host zu zeigen und vor allem einen schnellen und effizienten Einstieg in die Thematik zu ermöglichen.

Zur Benutzung dieses Buches

Diese Buch beschreibt den Umgang mit dem Windows Scripting Host aus verschiedenen Sichten. Durch eigene leidvolle Erfahrungen weiß ich, wo es bei den ersten Schritten klemmt. Einsteiger finden im ersten Kapitel einige Hinweise, was sich hinter dem WSH verbirgt und was sich damit anstellen läßt. Hier setze ich lediglich voraus, dass Sie zumindest rudimentäre Kenntnisse einer Programmiersprache besitzen. Eine Einführung in die objektorientierte Programmierung erhalten Sie in ➤ Kapitel 5.

Nachdem ich mich nun bereits fast sechs Monate mit der Thematik auseinandersetze (und seit mehreren Jahren mit JScript, VBScript und VBA arbeite), kenne ich aber auch die Probleme der »Profis«. Daher habe ich ➤ Kapitel 2 eigens dem Thema Entwicklungsumgebungen und Debugging gewidmet. Der Windows Scripting Host bietet keine Entwicklungsumgebung und unterstützt die Fehlerbehebung nur rudimentär. Mit

einigen Tricks und Zusatzwerkzeugen läßt sich aber sehr schnell eine halbwegs komfortable Entwicklungsumgebung schaffen. Zumindest hatte ich beim Schreiben dieses Kapitels das Gefühl, einen riesigen Sprung nach vorne geschafft zu haben (als ich mir nochmals meine ersten Ansätze zum Erstellen von Skripten im Windows Editor vor Augen hielt).

Die restlichen Kapitel führen Sie dann schrittweise in die Möglichkeiten und Techniken des Windows Scripting Host ein. Sie lernen einfache Programme in VBScript und JScript zu erstellen. Von Kapitel zu Kapitel wird dann die Thematik anspruchsvoller und auch schwieriger. Sofern Sie bereits mit JavaScript, Visual Basic oder VBScript gearbeitet haben, dürfte dies aber kein Problem darstellen. Die benötigten Informationen zu Objekten, Eigenschaften und Methoden lernen Sie in den betreffenden Abschnitten kennen. Für besonders eilige Zeitgenossen gibt es noch eine Sammlung an Rezepten, die in **↗** Kapitel 12 aufgezeigt werden.

Im **↗** Anhang finden Sie dann die WSH-Objektreferenz. Die Informationen befinden sich zwar auch auf der CD-ROM und im Internet. Persönlich finde ich es aber wesentlich einfacher, wenn ich beim Entwurf eines Skripts am Schreibtisch sitze und etwas in einem Buch nachschlagen kann (außerdem enthält das Buch die deutsche Fassung). Um Ihnen das Arbeiten mit diesem Buch zu erleichtern, wurden die nachfolgenden Begriffe zur Auszeichnung im Text verwendet.

Begriff	Bemerkung
Hinweis	Zusätzliche Hinweise zu bestimmten Optionen oder Hintergrundinformationen wurden mit dem nebenstehenden Vermerk im Text versehen.
Tip	Arbeitstechniken, die eine besonders elegante Vorgehensweise erlauben, oder Wissenswertes sind mit der nebenstehenden Auszeichnung markiert.
Wichtig	Einige Funktionen (wie z. B. das Formatieren einer Festplatte) bergen Risiken. Um Sie auf diese Gefahren hinzuweisen, finden Sie an den betreffenden Stellen den nebenstehenden Vermerk.

Zusätzlich wurden Begriffe wie Dateinamen und so weiter kursiv im Text hervorgehoben. Vermutlich haben Sie keine Zeit, das gesamte Buch von der ersten bis zur letzten Seite zu lesen. Die Gliederung in mehrere Kapitel trägt thematisch verwandten Funktionen und unterschiedlichen Schwierigkeitsgraden Rechnung. Über das Inhaltsverzeichnis sowie über den Index erhalten Sie einen direkten Zugriff zu den jeweiligen Themen.

Abschließend noch ein kurzer Hinweis: Ich habe mich in diesem Buch bemüht, deutschsprachige Begriffe zu verwenden (was durch Microsoft Press forciert wurde). Allerdings muß ich im Bereich des Programmierens immer wieder feststellen, dass sich

vieles doch durch englische Begriffe leichter und eindeutiger ausdrücken läßt. Anstelle des Begriffs »Steuerelemente« werden Sie daher gelegentlich auf den Terminus »Controls« stoßen. Um die Sprache nicht zu stark zu »verkomplizieren«, wurden auch Abkürzungen wie OLE, ADO, ATL etc. benutzt. Dort, wo die Begriffe eingeführt werden, finden Sie die jeweilige Erläuterung zur betreffenden Abkürzung. Sie können ggf. über das Stichwortverzeichnis nachschlagen, was die eine oder andere Abkürzung bedeutet.

Noch ein paar Hinweise

Die Beispiele dieses Buches wurden unter Windows 98 entwickelt und sind auch auf der Begleit-CD-ROM zu diesem Buch enthalten. Weitere Hinweise zum Inhalt der CD-ROM entnehmen Sie bitte der auf dem Medium enthaltenen Dokumentation. Hierzu laden Sie die im Hauptverzeichnis der CD-ROM enthaltene Datei *Start.htm* im Internet Explorer (ein Doppelklick auf das Dateisymbol genügt zur Anzeige). Die auf der CD-ROM zusammengestellten Programme und Informationen werden auf der Basis »AS-ISX« weitergegeben. Autor und Verlag übernehmen weder eine Unterstützung beim Einsatz noch eine Haftung für die Folgen der Anwendung. Für Rückmeldungen und Hinweise auf Fehler bin ich aber dankbar. Bitte haben Sie aber Verständnis, wenn ich nicht jedes Problem, welches vielleicht bei Ihnen auftritt, beantworten kann. Sollte bei Ihnen ein Beispiel nicht laufen, überprüfen Sie, ob die Voraussetzungen erfüllt sind (z. B. Dateien installiert, ActiveX-Komponenten registriert etc.). An den betreffenden Stellen liefert das Buch die entsprechenden Hinweise. Beachten Sie auch: Ein Autor kann und soll kein Ersatz für den Microsoft-Support sein (auch wenn einige Leser dies glauben).

Weiterhin mußte ich wegen des vorgegebenen Seitenumfangs auf die Behandlung spezieller Themen wie ADO, ADSI etc. verzichten. Mit den in diesem Buch vermittelten Kenntnissen sollte der interessierte Leser jedoch in die Thematik einsteigen können. Auf der Begleit-CD-ROM finden Sie beispielsweise im Inet SDK zusätzliche Informationen zu diesen Themen.

Zum Zeitpunkt, als dieses Buch geschrieben wurde, entstand bereits eine neue Version des WSH mit erweiterten Funktionen. Diese Version wird wohl zukünftig mit Microsoft Windows 2000 (und die Script-Engine mit dem Internet Explorer 5) ausgeliefert. Sofern Sie ältere Fassungen des WSH bzw. der Script-Engines verwenden und Probleme mit den Beispielen bekommen, sollten Sie auf die neueste Version aktualisieren. Auf der Website von Microsoft finden Sie unter <http://msdn.microsoft.com/scripting/related/default.htm> eine Reihe von Links zum Windows Scripting Host.

Was ist der Windows Scripting Host (WSH)?	7
Skripte erstellen und ausführen	13

In diesem Kapitel finden Sie eine kurze Einführung in die Thematik des Windows Scripting Host.

- ♦ Erfahren Sie, was sich hinter dem Begriff Windows Scripting Host verbirgt.
- ♦ Lesen Sie nach, welche Programmiersprachen durch den Windows Scripting Host unterstützt werden.
- ♦ Lernen Sie, wie sich Skriptprogramme unter Windows und aus MS-DOS heraus aufrufen lassen.
- ♦ Sehen Sie nach, wie die Eigenschaften der Skriptdateien eingestellt werden.

Mit diesem Wissen kennen Sie die grundlegenden Konzepte des Windows Scripting Host. Weiterhin können Sie WSH-Skriptdateien unter Windows ausführen.

Was ist der Windows Scripting Host (WSH)?

Die beiden nachfolgenden Abschnitte beleuchten, wofür der WSH gebraucht wird und welche Programmiersprachen er unterstützt.

Wozu braucht man den WSH?

In den Windows-Versionen vor Windows 98 gab es kaum Möglichkeiten zur Automatisierung bestimmter Aufgaben. MS-DOS stellt zwar eine Art Stapelverarbeitungssprache in Form der BAT-Dateien zur Verfügung, die sich auch unter Windows im MS-DOS-Fenster nutzen läßt. Aber die BAT-Dateien besitzen mehrere Nachteile. Sie können nur MS-DOS-Befehle aneinanderreihen. Schleifen und Verzweigungen lassen sich nur rudimentär einsetzen und erfordern häufig diffizile Tricks. Das Öffnen von Fenstern und Dialogfeldern ist überhaupt nicht möglich. Der unter Windows 3.1 enthaltene Recorder konnte zwar Windows-Eingaben (Tastencodes, Mausklicks) aufzeichnen und wiedergeben. Das Werkzeug erlaubte jedoch keinen Eingriff in die aufgezeichneten »Makros«. Außerdem wurde der Makrorecorder unter Windows 95 nicht mehr unterstützt.

Dies alles führte dazu, dass das Ausführen wiederkehrender Aufgaben unter Windows 95 bzw. Windows NT 4.0 kaum automatisierbar war. Um eine oder mehrere Datei(en) zu kopieren, können Sie zwar ein Stapelverarbeitungsprogramm einsetzen. Aber es sind keine (oder nur rudimentäre Benutzerabfragen) möglich. Eine Verknüpfung können Sie nur manuell anlegen und das Öffnen von Dialogen ist überhaupt nicht möglich. Das Starten von Programmen kann zwar aus MS-DOS-Stapelverarbeitungsprogrammen heraus erfolgen. Durch den *Start*-Befehl von MS-DOS erhalten Sie sogar eine gewisse Kontrolle über Windows (siehe /5/ im Literaturverzeichnis). Aber irgendwie ist dieser Ansatz unbefriedigend.

Hinweis

Lediglich das Installieren von Anwendungen läßt sich ab Windows 95/NT 5.0 über INF-Dateien gestalten (siehe /1,2/ im Literaturverzeichnis).

Der WSH als Lösung

Aus diesem Ansatz heraus bestand bereits seit Jahren die Forderung bzw. der Bedarf nach einer »Programmiersprache«, die direkt unter Windows – also ohne Visual Basic oder Office-Anwendungen – funktioniert. Diese Programmiersprache mußte die wichtigsten Aufgaben unter Windows unterstützen und möglichst den Zugriff auf die Ressourcen des Betriebssystems erlauben. Windows stellt in der Shell viele Objekte oder Pseudoobjekte zur Verfügung, die sich vom Benutzer manipulieren lassen. Sie können beispielsweise Verknüpfungen erstellen, Verbindungen zu Netzwerkressourcen aufbauen, Ordner und Dateien manipulieren, Drucker einrichten, auf die Registrierung zugreifen, Programme starten etc. Dies alles sollte irgendwie auch mit einer Programmiersprache machbar bzw. automatisierbar sein.

Die Lösung entwickelte sich dann evolutionär, da Microsoft seit dem Erscheinen von Windows 3.1 verschiedene Programmierumgebungen zur Verfügung stellte. Mit Visual Basic stand bereits eine Entwicklungsumgebung für Windows-Anwendungen zur Verfügung, die mit Visual Basic 4 auch 32-Bit-Anwendungen in Windows 95 unterstützte. Der Vorteil dieses Ansatzes: es lassen sich sogar direkt ablauffähige EXE-Programme erstellen. Aber niemand kauft sich eine Entwicklungsumgebung wie Visual Basic, um ein paar Zeilen Code zum automatischen Anlegen einer Verknüpfung zu erstellen. Außerdem sollte ein Skriptprogramm auch ohne aufwendige Entwicklungsumgebung (ggf. direkt im Windows-Editor) zu erstellen sein.

Der nächste Schritt in diese Richtung kam mit Microsoft Office 97. Enthielten die früheren Office-Programme bereits Makrosprachen, wurden in Microsoft Office 97 alle Anwendungen einheitlich mit Visual Basic für Applikationen 5.0 (VBA) ausgestattet. Bei VBA handelt es sich um eine Sprache, die auf Visual Basic aufsetzt (mit geringfügigen Modifikationen im Sprachumfang). Die Office-Anwendungen enthalten bereits die benötigte Entwicklungsumgebung zum Erstellen der Programme (siehe /3/ im Literaturverzeichnis). Dies bedeutete, dass ein Anwender von Microsoft Word 97 beispielsweise ein VBA-Programm erstellen und unter Word 97 ausführen konnte. Dieser Ansatz hatte aber im Hinblick auf Windows einige Haken. So waren die Möglichkeiten von VBA primär auf die Handhabung der Funktionen der jeweiligen Anwendung abgestimmt. Nur mit detaillierten Kenntnissen der VBA-Objekte ließ sich auch auf die Windows-Ressourcen zugreifen. Außerdem war diese Lösung zu aufwendig; nicht alle Windows-Anwender verfügen über Microsoft Office 97-Anwendungen oder möchten diese für einfache Skripte einsetzen.

Genaugenommen muß an dieser Stelle aber erwähnt werden, dass nicht alle Office-Anwendungen mit VBA 5.0 ausgestattet sind. Outlook unterstützt nur den reduzierten Umfang der Sprache VBScript. Hinweis

Mit dem Einsatz der Internet-Browser kamen erstmals Skriptsprachen unter der Windows-Benutzeroberfläche auf (wenn man von Spezialsprachen wie Pearl etc. absieht). In einem HTML-Dokument lassen sich Skripte einbetten, die beim Laden des Dokuments beziehungsweise bei Ereignissen wie beispielsweise dem Anklicken eines Elements im Dokument ausgeführt werden. Während Netscape im Navigator seit der Version 2.0 die Skriptsprache JavaScript unterstützte, ging Microsoft mit dem Internet Explorer 3.0 einen Schritt weiter. Dieser Browser unterstützt sowohl JavaScript als auch VBScript, eine Untermenge der Microsoft-Sprache Visual Basic. Das gleiche gilt für den Internet Explorer 4.0, der ebenfalls beide Sprachen unterstützt. Der Internet Explorer benutzt aber JScript, die Microsoft Implementierung von ECMAScript (eines herstellerübergreifenden auf JavaScript basierenden Standards). Dem HTML-Autor stehen damit zwei mächtige Programmiersprachen zur Verfügung. Über ActiveX-Steuerelemente (ActiveX-Controls) steht dabei der Zugriff auf alle

Funktionen des Betriebssystems oder der Anwendungen zur Verfügung. Anwender des Internet Information Server konnten zusätzlich Skripte zur Gestaltung von Active Server Pages verwenden.

Aber kommen wir zurück zu Windows. Die obigen Ausführungen zeigen, dass Visual Basic als Kern in verschiedenen Microsoft-Anwendungen verfügbar ist. Es war also nur noch eine Frage der Zeit, bis das Sprachmodul von Visual Basic (auch als Sprach-Engine oder Scripting Host bezeichnet) für Windows verfügbar gemacht würde. Bei der Einführung von Windows 98 entschloß sich Microsoft, den Programmteil, der im Internet Explorer oder im Internet Information Server für die Verarbeitung von JScript- und VBScript-Programmen zuständig war, als eigenständige Komponente zur Verfügung zu stellen. Diese Komponente wurde mit dem Begriff Windows Scripting Host (WSH) versehen. Diese Komponente ist in der Lage, ein in einer Datei vorliegendes Skript direkt auszuführen, d. h. die Anweisungen der Skriptdatei werden interpretiert und sofort ausgeführt.

Mit diesen Anweisungen lassen sich die verschiedenen Objekte der Windows-Shell manipulieren. Sie benötigen beispielsweise nur wenige Zeilen, um eine Verknüpfung per Skript auf dem Desktop einzurichten. Oder der Aufruf eines Meldungsfelds mit einem Text ist sogar mit einem einzigen Befehl der Art:

```
MsgBox "Hallo"
```

möglich, wobei hier die Notation der Programmiersprache VBScript gewählt wurde. Der obige Befehl öffnet ein Dialogfeld mit der Meldung »Hallo« (Abbildung 1.1).

Abbildung 1.1:
Einfaches
Dialogfeld



Ich werde weiter unten noch auf den Aufbau eines solchen Skripts zu sprechen kommen. Aber Sie sehen bereits, wie einfach die Anzeige von Dialogen wird.

Hinweis

Hinter den Kulissen werden dabei zwei Dateien für den Windows Scripting Host verwendet. *wscript.exe* stellt den Windows-basierenden Host zur Ausführung von Skripten dar, während *cscript.exe* die Skriptausführung von der MS-DOS-Eingabeaufforderung erlaubt. Die Datei *wscript.exe* befindet sich unter Windows 98 im Windows-Ordner, während *cscript.exe* im Unterordner *\Command* hinterlegt wird. Und für die Besitzer von Windows 95 sowie Windows NT 4.0 gibt es die gute Nachricht, dass die Skript-Engine des WSH kostenlos unter der Adresse <http://msdn.microsoft.com/scripting/>

zum Download bereitsteht. Wer keinen Internetanschluß besitzt, findet die betreffende Version im Ordner *\Tools\WshEngine* auf der Begleit-CD-ROM.

Was kann man mit dem WSH tun?

Sobald Sie über geeignete Skripte verfügen, eröffnet der Windows Scripting Host vielfältige Möglichkeiten:

- ◆ Sie können direkt Dialogfelder zur Benutzerführung anzeigen. Weiterhin bietet sich die Möglichkeit, Benutzereingaben abzurufen.
- ◆ Durch geeignete Objekte können Sie direkt auf die Windows-Shell zugreifen. Dies erlaubt Ihnen Verknüpfungen anzulegen oder Netzwerkverbindungen herzustellen.
- ◆ Sie können die Umgebungsvariable auslesen und Informationen über Windows abrufen. Zusätzlich lassen sich Programme starten und als Automatisierungsobjekte nutzen.

Mit den geeigneten Automatisierungsobjekten stehen Ihnen aber auch Operationen zur Handhabung von Dateien oder zum Zugriff auf beliebige Windows-Funktionen zur Verfügung. Sie können beispielsweise direkt die Windows-Registrierung manipulieren oder auf die Windows-API zugreifen (wobei letzteres jedoch einige Tricks erfordert).

Welche Sprachen unterstützt der WSH?

Der in Windows 98 und Windows 2000 (NT 5.0) enthaltene Windows Scripting Host (sowie die Version, die sich in Windows 95/NT 4.0 installieren läßt) unterstützt zwei Programmiersprachen:

- ◆ **VBScript:** Diese Sprache setzt auf der Syntax von Visual Basic auf, läßt aber verschiedene spezielle Sprachkonstrukte wie Datentypen für Variablen, Declare-Anweisungen etc. weg. Wer in Visual Basic programmiert hat, kommt mit VBScript sehr schnell zurecht. Mit etwas Übung lassen sich sogar VB- bzw. VBA-Programme direkt in VBScript umsetzen (es müssen lediglich die nicht unterstützten Sprachkonstrukte entfernt werden).
- ◆ **JScript:** Dies ist die Microsoft-Implementierung der Sprache ECMAScript. Bei ECMAScript handelt es sich um die herstellerübergreifende Standardisierung einer Programmiersprache, die ihre Wurzeln in der von Netscape im Navigator eingeführten Sprache JavaScript besitzt. Wer mit JavaScript gearbeitet hat, kann nahtlos auf JScript umsteigen.

Mit diesen beiden Programmiersprachen stehen Ihnen eigentlich alle Möglichkeiten zur Skriptprogrammierung offen. Microsoft hat aber die Schnittstelle des WSH offen gehalten. Dies bedeutet, Dritthersteller können ihre eigenen Parser integrieren und auf diese Weise weitere Skriptsprachen wie Pearl, TCL und Rexx unterstützen. Der im Literaturverzeichnis unter /9/ aufgeführte Titel enthält beispielsweise auf der Begleit-CD-ROM einen Host für Rexx.

Hinweis

In diesem Buch werden aber nur die beiden standardmäßig unterstützten Skriptsprachen VBScript und JScript besprochen, wobei der Schwerpunkt auf VBScript liegt.

Was ist bei WSH-Skripten anders als bei VB/VBA und JavaScript?

Sofern Sie bereits in VBA bzw. VB programmiert oder Skripte in VBScript bzw. JScript für HTML-Dokumente erstellt haben, dürfte der Umstieg zur WSH-Skriptprogrammierung kein Thema sein. Allerdings ergeben sich einige Unterschiede zwischen WSH-Skripten und den obigen Anwendungsfeldern:

- ♦ In Visual Basic steht Ihnen eine mächtige Entwicklungsumgebung zur Erstellung von Anwendungen zur Verfügung. Die Programme lassen sich in ausführbare EXE-Anwendungen übersetzen. Diese Funktionalität ist im Windows Scripting Host nicht vorhanden. Alle Skriptanweisungen werden in Textdateien mit den Erweiterungen *.vbs* oder *.js* hinterlegt und direkt aus diesen Dateien interpretiert. Der Vorteil: Sie können Ihre Skripte mit einem einfachen Texteditor erstellen.
- ♦ Die in VB und VBA verfügbaren Sprachkonstrukte zur Deklaration externer Funktionen und Prozeduren stehen nicht zur Verfügung. Ebenso fehlen Routinen zur detaillierten Behandlung von Laufzeitfehlern oder typisierte Variablen. In VBScript erhalten alle Variablen den Datentyp *Variant*.
- ♦ Einige in JavaScript verfügbare interne Objekte und Funktionen lassen sich in WSH-Skripten nicht verwenden. So macht das in einem Browser hinterlegte Objektmodell zum Zugriff auf die Dokumentobjekte im WSH-Skript keinen Sinn.

Gegenüber den in HTML-Dokumenten verwendeten Skripten ergibt sich ein weiterer gravierender Unterschied: WSH-Skripte kennen nicht die im Browser enthaltene ausgefeilte Ereignisverwaltung. Ereignisse wie *onClick* machen im WSH-Skript keinen Sinn, da das Skript selbst keine sichtbaren Formulare oder Fenster besitzt. Sie können zwar Dialogfelder öffnen, deren Schaltflächen sich anklicken lassen. Die Ereignisverarbeitung bezieht sich aber auf das Objekt, welches das Dialogfeld anzeigt und nicht auf das eigentliche von ihnen unterstützte Skript.

Es gibt aber einige rudimentäre Möglichkeiten, mit denen Sie sogenannte Hinweis Callback-Funktionen in einem Skript realisieren können. Dann kann das Skript auf ein externes Ereignis reagieren. Details zur Realisierung einer Callback-Funktion lernen Sie in den nachfolgenden Kapiteln kennen (z. B. bei der Formularausgabe per Internet Explorer). Eine detaillierte Sprachbeschreibung finden Sie in den Sprachreferenzen zu VBScript und JScript auf der Begleit-CD-ROM.

Skripte erstellen und ausführen

Der folgende Abschnitt zeigt exemplarisch, wie Sie ein Skript erstellen und anschließend ausführen. Diese Schritte werden an einem sehr einfachen Skript demonstriert, welches ein Meldungsfeld mit dem Text *Hallo* ausgibt.

Das erste Skript erstellen

Windows 98 enthält zwar einige Beispiele im Ordner `\Windows\Samples\Wsh`. Für die ersten »Gehversuche« empfiehlt es sich aber, direkt mit einem selbst erstellten Skript zu beginnen. Anhand dieses Beispiels lernen Sie die prinzipielle Vorgehensweise bei der Skriptprogrammierung sowie die Möglichkeiten zur Ausführung kennen. Das erste Skript soll in VBScript verfaßt werden und lediglich ein Dialogfeld (siehe **Abbildung 1.1**) mit dem Text *Hallo* auf dem Bildschirm anzeigen. Hierzu gehen Sie folgendermaßen vor:



Abbildung 1.2:
VBScript-
Programm

1. Öffnen Sie den Windows-Editor. Anschließend geben Sie die Anweisungen aus **Abbildung 1.2** im Textfenster ein.
2. Speichern Sie die Datei in einem Ordner der Festplatte, wobei der Dateiname beliebig sein darf. Als Dateinamenerweiterung müssen Sie allerdings *.vbs* verwenden.

Hinweis

Beim Speichern müssen Sie unbedingt die Dateinamenerweiterung *.vbs* (bzw. *.js* bei JScript) im Dialogfeld *Speichern unter* angeben, da der WSH sonst die Skriptsprache beim Ausführen des Skripts nicht erkennen kann. Doppelklicken Sie unter Windows auf eine solche Skriptdatei mit einer fehlerhaften Dateinamenerweiterung (z. B. *.vb*), öffnet das Betriebssystem das Dialogfeld *Öffnen mit*, da keine Anwendung für den Dateityp registriert ist. Haben Sie dagegen die Dateinamenerweiterung *.txt* verwendet, wird Windows den Texteditor aufrufen und den Inhalt des Skripts anzeigen. Rufen Sie dagegen die Skriptdatei mit der fehlerhaften Dateinamenerweiterung im MS-DOS-Fenster über das Programm *cscript.exe* auf, meldet der WSH einen Fehler, da die Dateinamenerweiterung nicht registriert ist.

Tip

Auf der Begleit-CD-ROM finden Sie im Ordner *\Beisp\Kap01* die beiden Dateien *VBScript.vbs* und *JScript.js*. Es handelt sich um Vorlagen, die Sie zur Skripterstellung nutzen können. Eine bestehende Skriptdatei laden Sie im Windows-Editor, indem Sie die Datei mit der rechten Maustaste anklicken und im Kontextmenü den Befehl *Bearbeiten* wählen.

Kommen wir nochmals zum Quellcode des ersten Skriptbeispiels zurück. Vereinbarungsgemäß soll als Programmiersprache VBScript gewählt werden. Dann müssen die Programmanweisungen der Syntax dieser Sprache entsprechen. Das folgende Listing zeigt die Anweisungen.

Listing 1.1:
*Das erste
VBScript-
Programm*

```

'*****
' File:   Hallo.vbs (WSH-Beispiel in VBScript)
' Autor:  (c) G. Born
'
' Es wird ein einfaches Dialogfeld geöffnet.
'*****

MsgBox "Hallo"

'*****

*** Ende -> WSH powered by Günter Born ***
'*****

```

Sofern Sie bisher noch nicht mit VBScript oder anderen Skriptsprachen gearbeitet haben, sollten Sie sich nicht durch die obigen Zeilen verwirren lassen. Jede Programmiersprache erlaubt Ihnen, Kommentarzeilen zu hinterlegen. Diese Kommentarzeilen werden vom Übersetzer oder Interpreter des Programmes ignoriert. In der Regel benutzt man solche Kommentare, um dem Leser Hinweise zur Funktion bestimmter Anweisungen oder zur Funktion des Programmes zu geben. Persönlich habe ich mir seit Jahren

angewöhnt, die Programme ausgiebig zu kommentieren, um später sehr leicht den Sinn der jeweiligen Anweisungen nachvollziehen zu können (spätestens drei Tage nach der Erstellung des Quellcodes weiß ich meist nicht mehr, was das Programm tun sollte – und dies gilt insbesondere, wenn mehrere Versionen eines Programmes vorliegen).

In VBScript werden Kommentare (in Anlehnung an die Visual Basic-Syntax) durch ein Hochkomma eingeleitet. Die Anweisung:

```
' Dies ist ein Kommentar
```

definiert einen gültigen Kommentar. Der komplette Inhalt der Zeile, der hinter dem Kommentarzeichen ' folgt, wird dann vom Windows Scripting Host ignoriert. Sie können einen Kommentar auch an das Ende einer Zeile mit einer ausführbaren Anweisung stellen. Die Zeile:

```
MsgBox "Windows Scripting Host" ' Text ausgeben
```

bewirkt die Anzeige eines Dialogfelds mit dem Text *Windows Scripting Host*, der Rest der Zeile wird aber ignoriert. Die Anweisung:

```
' MsgBox "Windows Scripting Host" ' Text ausgeben
```

führt dagegen dazu, dass der komplette Befehl als Kommentar übergangen wird.

Ein Blick in das obige Listing zeigt, dass dort heftig mit Kommentaren gearbeitet wurde. Eigentlich hätte die Anweisung:

```
MsgBox "Hallo"
```

zur Anzeige des gewünschten Dialogfelds gereicht. Zur Erhöhung der Transparenz habe ich jedoch alle Beispiele dieses Buches mit Kommentaren versehen. So wird jedes Skript mit einem Kommentarblock eingeleitet, der den Dateinamen sowie die Funktion des Skripts beschreibt. Das Programmende wird ebenfalls mit einem Kommentarblock signiert. Dies verhindert, dass irrtümlich die letzten Zeilen unbemerkt verloren gehen.

Sie finden die Beispieldatei *Hallo.vbs* im Ordner *\Beisp\Kap01* auf der [Hinweis](#) Begleit-CD-ROM zu diesem Buch.

Gleiches Beispiel, aber in JScript

Möchten Sie das erste Skriptprogramm lieber in JScript erstellen, gehen Sie wie beim Erstellen eines VBScript-Programms vor. Anstelle der obigen Anweisungen müssen Sie jedoch JScript-Befehle im Editor eingeben. Das Beispiel sieht in JScript folgendermaßen aus:

```
//*****  
// File:   Hallo.js (WSH-Beispiel in JScript)  
// Autor:  (c) G. Born  
//  
// Es wird ein einfaches Dialogfeld geöffnet.  
//*****
```

Listing 1.2:
Beispiel in JScript

```
WScript.Echo("Hallo");
```

```
//*****  
//*** Ende -> WSH powered by Günter Born      ***  
//*****
```

Hinweis

Beim Speichern müssen Sie unbedingt die Dateinamenerweiterung *.js* im Dialogfeld *Speichern unter* angeben, da der WSH sonst die Skriptsprache beim Ausführen des Skripts nicht erkennen kann.

In JScript ergeben sich einige Änderungen gegenüber dem obigen VBScript-Beispiel. Als erstes sind Kommentare in JScript durch zwei Slash-Zeichen *//* einzuleiten. In obigem Beispiel sehen Sie diese Kommentarzeilen zu Beginn und am Ende des Skripts. Die zweite Neuerung kommt bei der eigentlichen Anweisung zur Ausgabe des Dialogfelds. JScript kennt die *MsgBox*-Funktion nicht. Sie müssen sich daher etwas anderes einfallen lassen. In der WSH-Objektreferenz können Sie aber nachlesen, dass der Windows Scripting Host im Objekt *WScript* die Methode *Echo* unterstützt (auf die Feinheiten der Begriffe Objekt, Methode etc. kommen wir später noch zu sprechen). Mit der Anweisung:

```
WScript.Echo("Hallo");
```

erreichen Sie die Anzeige eines Dialogfelds mit dem Text *Hallo*. Eine weitere Neuerung besteht darin, dass in JScript alle Anweisungszeilen (bis auf einige wenige Ausnahmen) mit einem Semikolon abzuschließen sind.

Hinweis

Auf der Begleit-CD-ROM finden Sie das Beispiel *Hallo.js* im Ordner *\Beisp\Kap01*.

WSH-Skripte ausführen

WSH-Skripte werden in Dateien mit den Dateinamenerweiterungen *.vbs* und *.js* gespeichert. Sie können die in diesen Dateien gespeicherten Skriptprogramme sowohl unter Windows als auch über die MS-DOS-Eingabeaufforderung ausführen.

Skript unter Windows ausführen

Unter Windows genügt zur Ausführung des Skripts ein Doppelklick auf das Symbol der Skriptdatei. Bei der Installation des Windows Scripting Host wird der Dateityp für die Skriptsprachen registriert. Beim Doppelklick führt Windows 98 daher das betreffende Skript direkt über das Programm *Wscript.exe* aus.

Sie können zum ersten Test den Ordner `\Beisp\Kap01` auf der Begleit-CD-ROM im Explorer-Fenster öffnen. Dann genügt zum Starten der Skriptprogramme ein Doppelklick auf die jeweilige Skriptdatei. Das Ergebnis des Skripts *Hallo.vbs* sehen Sie in **Abbildung 1.1**, während in **Abbildung 1.3** die Ausgabe von *Hallo.js* zu sehen ist. Das Programm meldet sich in beiden Fällen mit einem einfachen Dialogfeld, welches Sie per *OK*-Schaltfläche schließen können.



Abbildung 1.3:
Ausgabe des
Programmes
Hallo.js

Das Verhalten der Skriptdatei (ob diese beispielsweise einen Dialog anzeigt) hängt von den internen Programmanweisungen ab. Die unterschiedlichen Texte in der Titelzeile resultieren aus den unterschiedlichen Funktionen, die in den beiden Skripten zur Ausgabe verwendet wurden. In VBScript wurde eine interne Funktion benutzt, folglich erscheint der Text *Visual Basic* in der Titelzeile. Bei JScript-Programm weist die Titelzeile darauf hin, dass eine Methode des WSH zur Ausgabe verwendet wurde. Sie haben aber die Möglichkeit, den Inhalt der Titelzeile selbst zu definieren. Auf diese Fragestellungen kommen wird in späteren Kapiteln noch zurück.

Hinweis

Skript mit Parametern starten

Bei den obigen Beispielen werden zwar keine Parameter benötigt. Es gibt aber sicherlich Fälle, in denen dem Skript der Name einer zu bearbeitenden Datei übergeben wird. Das folgende VBScript-Programm demonstriert diesen Sachverhalt, indem es alle beim Aufruf übergebenen Parameter in einem Dialogfeld anzeigt (**Abbildung 1.4**).



Abbildung 1.4:
Anzeige der
Aufrufparameter

Listing 1.3:
Programm zur
Abfrage der
Aufrufparameter

```

'*****
' File:   Param.vbs (WSH-Beispiel in VBScript)
' Autor:  (c) G. Born
'
' Anzeige des übergebenen Parameters in einem
' Dialogfeld.
'*****

text = "Argumente" + vbCRLF + vbCRLF

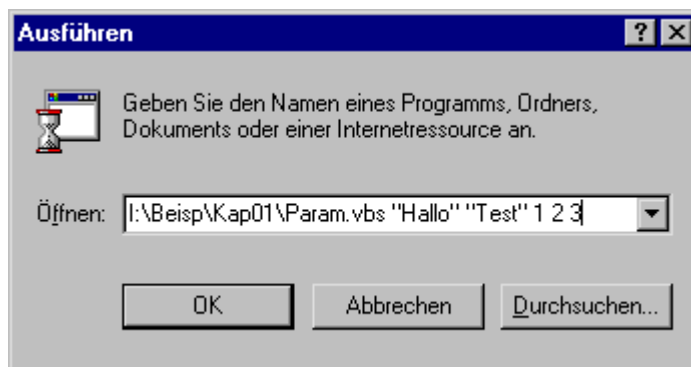
Set objArgs = Wscript.Arguments      ' Objekt erzeugen
For I = 0 to objArgs.Count - 1      ' über alle Argumente
    text = text + objArgs(I) + vbCRLF ' hole Argument
Next

Wscript.Echo text ' zeige Argument per Echo
'*****
'*** Ende -> WSH powered by Günter Born ***
'*****

```

Das Skript greift auf die *Arguments*-Auflistung des *WScript*-Objekts zurück und liest dann die jeweiligen Parameter aus. Auf die Einzelheiten des Zugriffs auf die Parameter komme ich in weiteren Kapiteln detaillierter zurück. An dieser Stelle interessiert nur die Frage, wie läßt sich ein solcher Parameter unter Windows angeben. Ein Doppelklick auf den Dateinamen hilft nicht weiter, da er nur das Skript ausführt.

Abbildung 1.5:
Skriptaufruf über
das Dialogfeld
Ausführen



Um ständig wechselnde Parameter an ein Skript zu übergeben, läßt sich das Dialogfeld *Ausführen* (Befehl *Ausführen* im Startmenü) verwenden. In **Abbildung 1.5** sehen Sie ein Beispiel für einen solchen Aufruf. Sie müssen beim Aufruf den Pfad zur Skriptdatei direkt angeben. Die Parameter sind getrennt durch Leerzeichen an den Namen der Skriptdatei anzuhängen. Das obige Skript liest die Parameter aus und zeigt diese in dem in **Abbildung 1.4** gezeigten Dialogfeld zur Demonstration an.

Tip Beginn

An dieser Stelle möchte ich Sie gleich auf einige besondere Probleme im Zusammenhang mit Parametern hinweisen. In **Abbildung 1.5** sehen Sie, dass ich einige Parameter in Anführungszeichen gesetzt habe. Dies ist immer dann erforderlich, wenn Sie einen Text als Parameter übergeben, der Leerzeichen enthält. Ich hatte ja bereits ausgeführt, dass das Leerzeichen als Separator dient. Möchten Sie beispielsweise den Namen »Günter Born« als Parameter übergeben, könnten Sie dies in der Form:

```
I:\Beisp\Kap01\Param.vbs Günter Born
```

tun. Das Skript interpretiert die beiden Zeichenketten »Günter« und »Born« jedoch als getrennte Parameter, d. h. Ihr Programm wird mit hoher Wahrscheinlichkeit einen Fehler liefern. Um Zeichenketten, die Leerzeichen enthalten, als Parameter zu übergeben, müssen Sie diese in Anführungszeichen stellen.

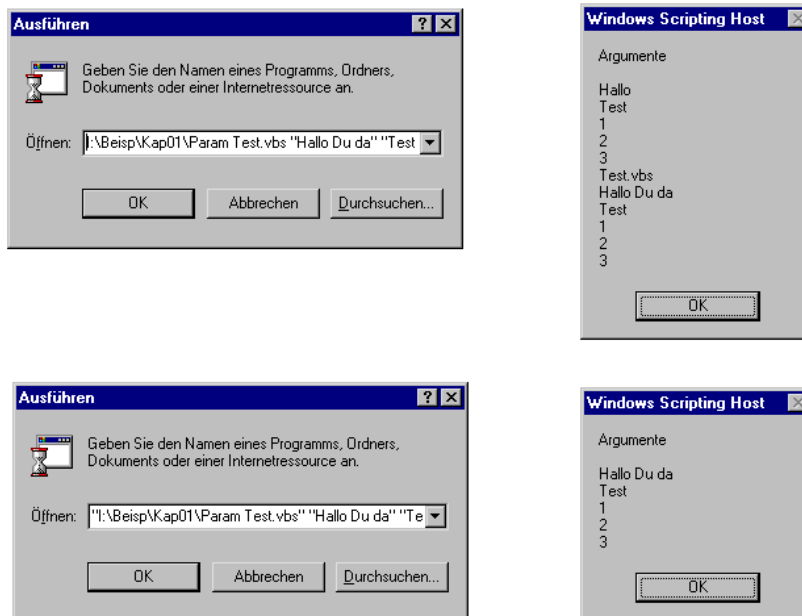


Abbildung 1.6:
Beispiele für
fehlerhafte
Parameterauswertung

Der gleiche Befehl sieht dann folgendermaßen aus:

```
I:\Beisp\Kap01\Param.vbs "Günter Born"
```

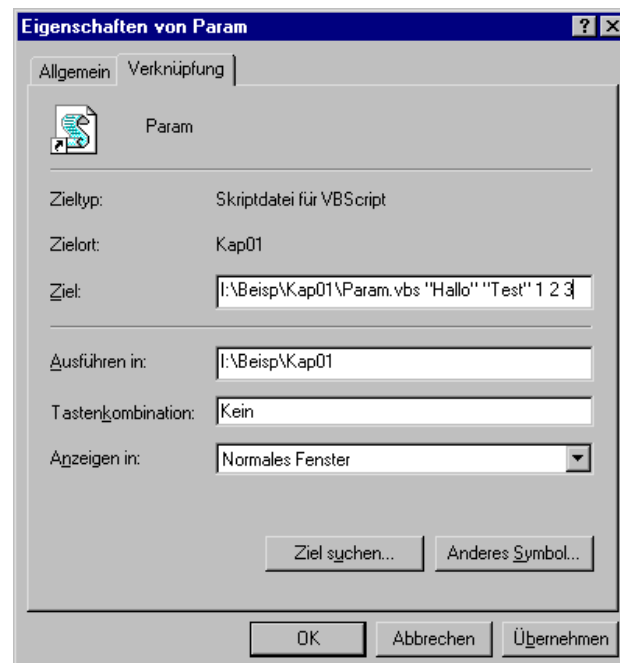
Verwenden Sie lange Dateinamen bei der Benennung der Skriptdateien? Auch hier lauern einige Fallen. Persönlich vermeide ich die langen Namen nach Möglichkeit. Beim Aufruf solcher Skripte unter MS-DOS werden meist die 8.3-Aliasnamen anstelle der langen Dateinamen verwendet. Und dann müssen Sie sich mit der Tilde herumschlagen (z. B. `Param~1.vbs`). Aber es gibt noch eine weitere Falle, die bei der Verwendung langer Dateinamen

unbemerkt lauert. Ich habe diesen Fall einmal in **Abbildung 1.6** in verschiedenen Dialogfeldern provokativ gegenübergestellt. Links sehen Sie den Aufruf im Dialogfeld *Ausführen*, während im rechts danebenstehenden Dialogfeld die Ausgabe des Skripts zu sehen ist. Während das Dialogfeld unten rechts die Parameter richtig auflistet, zeigt das rechts oben angeordnete Dialogfeld zusätzliche Parameter an. Zuerst glaubte ich an einen Fehler im Skript. Aber ein Test brachte schnell die Erklärung: Bei diesem Skript wurde ein langer Dateiname benutzt. Beim ersten Skriptaufruf wurden lediglich die Parameter in Anführungsstriche gesetzt. Der WSH kommt dann »durcheinander« und liefert eine fehlerhafte Anzahl an Parametern. Erst wenn Sie den Namen und Pfad der Skriptdatei in Anführungszeichen einfassen (dies wurde beim zweiten Aufruf genutzt), zeigt das Skript die übergebenen Parameter richtig an.

Sie können dieses Verhalten selbst testen. Auf der Begleit-CD-ROM finden Sie im Ordner `\Beisp\Kap01` die Skriptdateien *Param.vbs* und *Param.js*, die die übergebenen Parameter anzeigen. Die beiden Dateien *Param Test.vbs* und *Param Test.js* besitzen einen identischen Inhalt. Sie wurden lediglich so umbenannt, dass sich ein langer Dateiname ergab.

Tip Ende

Abbildung 1.7:
Verknüpfungseigen-
schaften auf eine
Skriptdatei



Vermutlich wird Ihnen die Übergabe der Parameter über das Dialogfeld *Ausführen* zu aufwendig. Sofern sich die Parameter selten ändern, können Sie aber zu einem Trick greifen und eine Verknüpfung zur Skriptdatei einrichten.

1. Klicken Sie mit der rechten Maustaste auf das Symbol der Skriptdatei.
2. Wählen Sie im Kontextmenü den Befehl *Verknüpfung erstellen*.
3. Anschließend klicken Sie mit der rechten Maustaste auf die Verknüpfungsdatei und wählen im Kontextmenü den Befehl *Eigenschaften*.
4. Auf der Registerkarte *Verknüpfung* können Sie nun im Feld *Ziel* die Parameter hinter dem Namen der Skriptdatei eintragen (**Abbildung 1.7**).

Sobald Sie das Eigenschaftenfenster über die *OK*-Schaltfläche schließen, übernimmt Windows die Parameter. Diese werden beim Aufruf der Verknüpfung automatisch an das Skript übergeben.

Gemäß den Ausführungen auf den vorhergehenden Seiten müssen Sie Parameter und Dateinamen, die Leerzeichen enthalten, in Anführungszeichen einfassen. Beim Anlegen der Verknüpfung beläßt Windows übrigens die Dateinamenerweiterung (z. B. *.vbs* oder *.js*), während die Dateinamenerweiterung bei anderen Verknüpfungsobjekten ausgeblendet wird. Die Ursache liegt darin, dass bei Skriptdateien zusätzliche Eigenschaften über WSH-Dateien definiert werden können (siehe ↗ Abschnitt »Skripteigenschaften festlegen«).

Hinweis

Skripte unter MS-DOS ausführen

Im MS-DOS-Fenster wird die Ausführung von Skriptdateien ebenfalls über das Programm *cscript.exe* unterstützt. Geben Sie in der MS-DOS-Befehlsebene die folgende Befehlszeile ein:

```
cscript pfad\Skriptname [Host-Optionen] [Skriptoptionen]
```

Als Skriptname ist der Name der betreffenden Skriptdatei mitsamt Laufwerk und Pfad anzugeben. An diesen Skriptnamen lassen sich Optionen für den Host und/oder für das Skript anhängen (**Abbildung 1.8**).

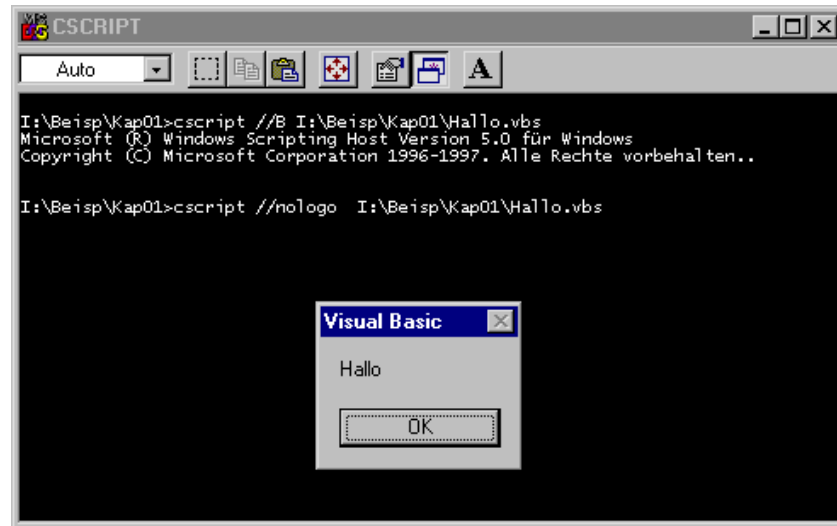
Die Host-Optionen ermöglichen es, bestimmte Windows Scripting Host-Funktionen zu- oder abzuschalten. Diesen Optionen sind immer zwei Slash-Zeichen // vorangestellt. Die nachfolgende Tabelle enthält eine Übersicht aller Host-Optionen:

Option	Bedeutung
//I	Schaltet den Interaktiv-Modus ein, der die Anzeige von Benutzermeldungen und Skriptfehlern zuläßt. Dies ist die Standardvorgabe, die durch //B deaktiviert wird.
//B	Aktiviert den Batch-Modus, die die Ausgabe aller Benutzermeldungen und Skriptfehler auf der MS-DOS-Ebene unterdrückt. Falls Ihr Skript beispielsweise ein

Tabelle 1.1:
Host-Optionen
beim Aufruf von
cscript.exe

	Dialogfeld über die <i>Echo</i> -Methode ausgibt, wird dieses in diesem Modus nicht angezeigt.
//T:nn	Setzt einen Time-out-Wert. Dieser gibt die maximale Zeit in Sekunden an, die das Skript ausgeführt werden darf. Terminiert das Skript innerhalb dieser Zeit nicht, wird es zwangsweise beendet.
//logo	Bewirkt die Anzeige eines Logos beim Aufruf des Skripts. Diese Anzeige wird mit //nologo unterdrückt.
//nologo	Unterdrückt die Anzeige eines Logos bei der Skriptausführung.
//H:Cscript //H:Wscript	Registriert <i>Cscript.exe</i> oder <i>Wscript.exe</i> als den Standard-Interpreter für die Ausführung eines Skripts. Fehlt diese Angabe, wird die Windows-Version der Script-Engine (<i>Wscript.exe</i>) zur Ausführung verwendet.
//S	Sichert die Befehlszeile zum Ausführen des Skripts für diesen Benutzer.
//?	Zeigt die Textseite mit den Host-Optionen.

Abbildung 1.8:
MS-DOS-Fenster
mit Aufrufen des
Windows Scripting
Host



Hinweis

Bevor Sie über die Bedeutung des Schalters *//logo* lange rätseln: unter dem Terminus »logo« verstehen die Microsoft-Entwickler die Textmeldung, die bei der Ausführung des Skripts auf der MS-DOS-Befehlszeilenebene ausgibt. Mit *//logo* wird dieser Text angezeigt. Sie sehen dies in **Abbildung 1.8** in der obersten Zeile. Beim zweiten Aufruf wurde diese Ausgabe dagegen mit *//nologo* unterdrückt.

Parameterübergabe beim Aufruf von *Cscript.exe*

Benötigt Ihr Skript bestimmte Parameter beim Aufruf (z. B. den Namen einer Datei)? Dies ist auf der MS-DOS-Befehlsebene sehr einfach möglich. Neben den Host-Optionen, die mit zwei vorangestellten Slash-Zeichen markiert sind, können Sie dem Skript bestimmte Parameter als zusätzliche Skriptoptionen übergeben. Den Skriptparametern kann ein einfaches Slash-Zeichen vorangestellt werden. Das bereits erwähnte Beispiel von der Begleit-CD-ROM läßt sich dann unter MS-DOS folgendermaßen aufrufen:

```
cscript G:\Beisp\Kap01\Param.js //S /"hallo" 1 2 3
```

Hierbei wird vorausgesetzt, dass das CD-ROM-Laufwerk dem Buchstaben *G:* zugeordnet ist. Sie können aber auch auf das Slash-Zeichen verzichten, wenn Sie die Host-Optionen vor dem Namen der Skriptdatei angeben. Dies ist in **Abbildung 1.9** zu sehen.

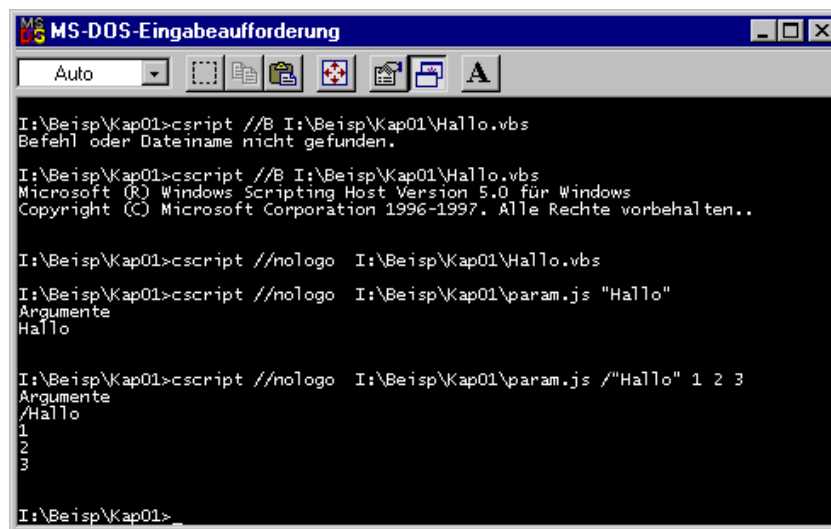


Abbildung 1.9:
Beispiel für die
Parameterübergab
e an das Skript in
MS-DOS

Beachten Sie: Da die einzelnen Parameter optional sind müssen Sie in der Befehlszeile mindestens den Namen der Skriptdatei angeben. Hinweis

In **Abbildung 1.9** wurde übrigens die JScript-Variante des Skripts aufgerufen. Diese Variante benutzt die *Echo*-Methode des *WScript*-Objekts zur Anzeige der Parameter. Die Folge: Anstelle eines Dialogfelds zeigt die *Echo*-Methode die Ausgaben direkt in der MS-DOS-Befehlszeile an. Einzelheiten können Sie dem nachfolgenden Listing entnehmen.

```
//*****  
// File:   Param.js (WSH-Beispiel in JScript)  
// Autor:  (c) G. Born  
//  
// Anzeige des übergebenen Parameters in einem
```

Listing 1.4:
Parameterauswert
ung in JScript

```
// Dialogfeld.
//*****

var objArgs;
var text = "Argumente \n";

var objArgs = WScript.Arguments;      // Objekt erzeugen

for (var i=0; i < objArgs.length; i++) // über alle Argumente
    text = text + objArgs(i) + '\n';    // hole Argument

WScript.Echo (text); // zeige Argument per Echo
WScript.Quit();
//*****
//*** Ende -> WSH powered by Günter Born      ***
//*****
```

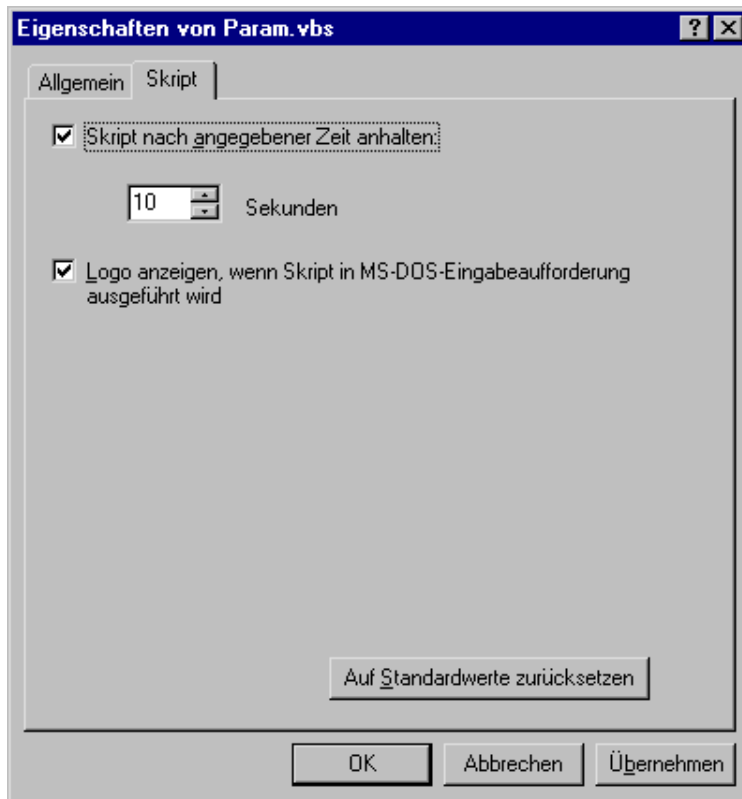
Hinweis

Im Vorgriff auf kommende Kapitel möchte ich an dieser Stelle noch auf eine weitere Besonderheit hinweisen. JScript unterstützt zwar die *Arguments*-Auflistung des *WScript*-Objekts. Sobald Sie jedoch die *Counts*-Eigenschaft des Aufzählungsobjekts abfragen wollen, löst dies einen Laufzeitfehler aus. In JScript müssen Sie daher auf die *length*-Eigenschaft ausweichen, die aus Kompatibilitätsgründen eingeführt wurde. Sie finden das Programm *Param.js* im Ordner *\Beisp\Kap01* auf der Begleit-CD-ROM.

Skripteigenschaften festlegen

Unter MS-DOS können Sie gemäß der obigen Beschreibung bestimmte Optionen beim Aufruf mitgeben. In Windows läßt sich das Skript zwar über das Dialogfeld *Ausführen* aufrufen. Meist wird man jedoch ein Skript direkt per Doppelklick auf die betreffende Datei (oder auf ein Verknüpfungssymbol dieser Datei) starten. Sie haben aber die Möglichkeit, unter Windows die Eigenschaften zur Ausführung einer Skriptdatei zu definieren.

Abbildung 1.10:
Die Registerkarte
Skript



1. Klicken Sie mit der rechten Maustaste auf das Dateisymbol der Skriptdatei, und wählen Sie im Kontextmenü den Befehl *Eigenschaften*.
2. Auf der Registerkarte *Skript* können Sie dann die Eigenschaften für die Skriptausführung einstellen (**Abbildung 1.10**).

Über das Kontrollkästchen *Skript nach angegebener Zeit anhalten* und das zugehörige Drehfeld läßt sich der Time-out-Wert zum Beenden des Skripts angeben. Stellt Windows fest, dass das Skript auch nach dieser Zeit noch aktiv ist, wird dieses abgebrochen.

Sobald Sie die Registerkarte schließen, legt Windows im Ordner der Skriptdatei eine neue Datei mit dem Namen der Skriptdatei, aber der Dateinamenerweiterung *.wsh* an. Rufen Sie diese Datei auf, wird auch unter Windows die Time-out-Zeit überwacht. Die WSH-Datei ist als Textdatei ähnlich wie eine INI-Datei aufgebaut und läßt sich mit einem beliebigen Texteditor öffnen:

```
[ScriptFile]
Path=C:\WINDOWS\Samples\WSH\showprop.vbs

[Options]
```

Listing 1.5:
Beispiel für den
Inhalt einer WSH-
Datei

```
Timeout=10
DisplayLogo=1
BatchMode=0
```

Die absolute *Path*-Angabe im Abschnitt [ScriptFile] identifiziert die auszuführende Skriptdatei. Die Schlüsselwörter im Abschnitt [Options] legen die Ablaufeigenschaften fest. *Timeout* gibt die im Eigenschaftenfenster auf der Registerkarte *Skript* eingestellte Time-out-Zeit an. *DisplayLogo=1* erzwingt die Anzeige eines Logos (d. h. der Textzeile beim Aufruf des Hosts gemäß **Abbildung 1.8**) im MS-DOS-Fenster.

Hinweis

Die Datei *Param.wsh* von der Begleit-CD-ROM wird auf Ihrem System wegen der absoluten Pfadangabe vermutlich nicht funktionieren. Kopieren Sie die Skriptdatei *Param.vbs* bzw. *Param.js* in einen Ordner der lokalen Festplatte, und legen Sie die WSH-Datei neu an (siehe den Abschnitt vor **Listing 1.5**).

Tip

Der Eintrag *BatchMode* läßt sich nicht per Registerkarte ändern und ist üblicherweise auf 0 gesetzt. Dies bewirkt, dass beim Ablauf des Skripts Benutzer- und Fehlermeldungen ausgegeben werden. Ändern Sie diesen Wert im Windows-Editor auf 1, läuft das Skript im Batch-Modus, d. h. alle Meldungen werden auch beim Aufruf von der Windows-Ebene unterdrückt. Dies ist beispielsweise hilfreich, wenn Sie ein Skript im Hintergrund oder beim Windows-Start ausführen möchten.

Hinweise zu den Windows-Skriptbeispielen

Windows 98 wird bereits mit einigen Skript-Beispielen ausgeliefert. Diese Beispiele finden sich nach der Installation im Windows-Ordner *\Samples\Wsh*. Wenn Sie diesen Ordner im Windows-Explorer öffnen, lassen sich die Beispiele testen. Die folgende Tabelle gibt Ihnen Hinweise zur Funktion der einzelnen Beispiele.

Tabelle 1.2:
*WSH-Beispiele in
Windows 98*

Datei	Bemerkung
<i>Chart.vbs</i> <i>Chart.js</i>	Demonstriert den Zugriff auf Microsoft Excel (auf die Chart-Funktion) per Windows Scripting Host.
<i>Excel.vbs</i> <i>Excel.js</i>	Das Beispiel demonstriert die Methode zum Ausführen anderer Anwendungen. Hier wird eine Excel-Tabelle mit Informationen erzeugt, in der Windows Scripting Host-Eigenschaften angezeigt werden
<i>Network.vbs</i> <i>Network.js</i>	Dieses Beispiel demonstriert die Verwendung des <i>WSHNetwork</i> -Objekts. Es ermittelt Netzwerkeigenschaften (Benutzername und Computernamen), stellt eine Verbindung zum Netzwerk her und listet die Netzlaufwerke auf. Anschließend wird

	diese Verbindung wieder beendet.
<i>Registry.vbs</i> <i>Registry.js</i>	Demonstriert den Zugriff auf die Registrierung. Es werden Einträge geschrieben und wieder gelöscht.
<i>Shortcut.vbs</i> <i>Shortcut.js</i>	Benutzt das <i>WSHShell</i> -Objekt, um eine Verknüpfung auf dem Desktop anzulegen.
<i>Showvar.vbs</i>	Das Beispiel ermittelt Systeminformationen und listet die Umgebungsvariablen der Maschine auf.

Zusätzliche Informationen zum Windows Scripting Host liefert die (englischsprachige) Hilfedatei *wshadmin.hlp* im Verzeichnis *\tools\reskit\scripting* auf der Windows 98-CD-ROM.

Ist der Windows-Ordner *\Samples\Wsh* mit den Beispielen nicht vorhanden? Klappt der Aufruf der WSH-Programme nicht? Öffnet Windows 98 beim Aufruf eines Skripts per Doppelklick das Dialogfeld *Öffnen mit*? Dann müssen Sie vermutlich erst den Windows Scripting Host als optionale Windows-Komponente installieren. Doppelklicken Sie in der Systemsteuerung auf das Symbol *Software*. Den Windows Scripting Host finden Sie unter Windows 98 in der Komponente *Zubehör* (Eintrag *Zubehör* markieren und auf die Schaltfläche *Details* klicken, dann im Dialogfeld *Zubehör* die Komponente *Windows Scripting Host* zum Installieren markieren).

Editoren zur Skriptentwicklung	30
Hilfen zur Skriptbearbeitung	38
Hilfen zur Skriptentwicklung	43
ActiveX-Module installieren/deinstallieren	53
Skriptdateien testen	60

Der Windows Scripting Host besitzt keine eigene Entwicklungsumgebung zur Erstellung der Skripte. Im Prinzip reicht ein einfacher Texteditor aus, um den Quellcode einzugeben. Um effizienter zu arbeiten, benötigen Sie aber Entwicklungshilfen. In diesem Kapitel lernen Sie verschiedene Werkzeuge und Techniken kennen, mit denen sich die Skripterstellung vereinfachen läßt.

- ◆ Lesen Sie nach, warum ein einfacher Editor zwar für die Skripteingabe ausreicht, aber sonst kaum Hilfestellung liefern kann.
- ◆ Erfahren Sie, wie sich die Skripterstellung mit geeigneten Programmen erheblich erleichtern läßt.
- ◆ Setzen Sie die restlichen in diesem Buch erwähnten Hilfsmittel ein, um die Entwicklung Ihrer Skripte auf eine professionelle Basis zu stellen.
- ◆ Schlagen Sie nach, um zu erfahren, wie sich Skriptprogramme auf Fehler testen und debuggen lassen.

Mit dem Wissen aus diesem Kapitel sollte die effiziente Entwicklung von Skripten kein größeres Problem mehr darstellen. Wer gerade in die Thematik der Programmerstellung für den WSH einsteigt, mag dieses Kapitel vorerst übergehen.

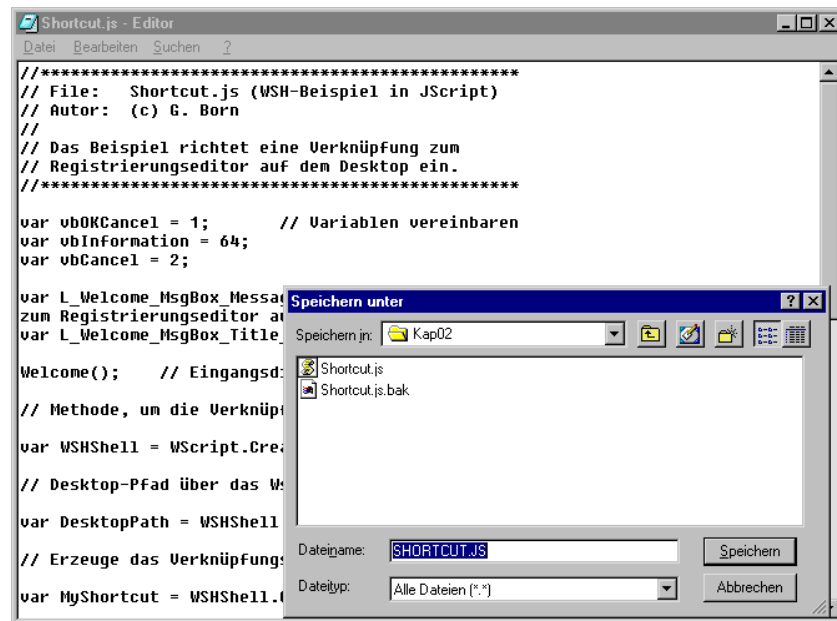
Editoren zur Skriptentwicklung

In diesem Abschnitt erfahren Sie, wie sich Skripte erstellen und ändern lassen. Sie lernen auch die Werkzeuge kennen, mit denen dieser Vorgang effizienter von der Hand geht.

Wie wird ein Skriptprogramm erstellt?

Ein Skript läßt sich direkt im Windows-Editor als Textdatei erstellen. Sie müssen lediglich die Anweisungen in der Syntax der benutzten Sprache im Editor eintippen.

Abbildung 2.1:
Speichern eines
Skripts im
Windows-Editor



Anschließend speichern Sie das Programm in einer Skriptdatei. Bei VBScript verwenden Sie die Dateinamenserweiterung *.vbs*, und bei JScript *.js* (**Abbildung 2.1**).

Tip

Stellen Sie den Dateityp beim Speichern im Dialogfeld *Speichern unter* auf *Alle Dateien (*.*)*. In diesem Fall werden Ihnen die bereits im Zielordner abgelegten Skriptdateien angezeigt.

Nachfolgend finden Sie das komplette in **Abbildung 2.1** als Ausschnitt gezeigte Programmlisting. Das Skript ist in JScript verfaßt und erzeugt bei der Ausführung eine Verknüpfung des Registrierungseditors auf dem Desktop.

Listing 2.1:
*JScript-Beispiel für
 der Windows
 Scripting Host*

```
//*****
// File:   Shortcut.js (WSH-Beispiel in JScript)
// Autor:  (c) G. Born
//
// Das Beispiel richtet eine Verknüpfung zum
// Registrierungseditor auf dem Desktop ein.
//*****

var vbOKCancel = 1;      // Variablen vereinbaren
var vbInformation = 64;
var vbCancel = 2;

var L_Welcome_MsgBox_Message_Text = "Dieses Skript erstellt eine
Verknüpfung zum Registrierungseditor auf Ihrem Desktop.";
var L_Welcome_MsgBox_Title_Text   = "Borns Windows Scripting Host-
Beispiel";

Welcome();    // Eingangsdialog zeigen

// Methode, um die Verknüpfung zu erzeugen

var WSHShell = WScript.CreateObject("WScript.Shell");

// Desktop-Pfad über das SpecialFolders-Eigenschaft ermitteln

var DesktopPath = WSHShell.SpecialFolders("Desktop");

// Erzeuge das Verknüpfungsobjekt auf dem Desktop

var MyShortcut = WSHShell.CreateShortcut(DesktopPath + "\\Borns
Regedit.lnk");

// Objekteigenschaften setzen, %windir% gibt den Windows-Ordner an

MyShortcut.TargetPath =
WSHShell.ExpandEnvironmentStrings("%windir%\\Regedit.exe");
MyShortcut.WorkingDirectory = WSHShell.ExpandEnvironmentStrings("%windir%");
MyShortcut.WindowStyle = 4;    // Fensterstil, als Logo
MyShortcut.IconLocation =
WSHShell.ExpandEnvironmentStrings("%windir%\\Regedit.exe, 0");
MyShortcut.Save();    // Speichern

WScript.Echo("Der Registrierungseditor wurde auf Ihrem Desktop
eingerrichtet");
```

```

////////////////////////////////////
/////
//
// Welcome
//
function Welcome() {
    var WSHShell = WScript.CreateObject("WScript.Shell");
    var intDoIt;

    intDoIt = WSHShell.Popup(L_Welcome_MsgBox_Message_Text,
                             0,
                             L_Welcome_MsgBox_Title_Text,
                             vbOKCancel + vbInformation );

    if (intDoIt == vbCancel) {
        WScript.Quit();
    }
}

//*****
//*** Ende -> WSH powered by Günter Born      ***
//*****

```

Die einzelnen Anweisungen müssen der Syntax der gewählten Skriptsprache entsprechen. In JScript sind beispielsweise alle Anweisungen mit einem Semikolon abzuschließen. Die geschweiften Klammern { } fassen Anweisungen zu Blöcken zusammen. Auf die Details der Realisierung komme ich zu einem späteren Zeitpunkt zurück.

Hinweis

Sie finden diese Beispieldatei *Shortcut.js* auf der Begleit-CD-ROM im Ordner *\Beisp\Kap02*.

So läßt sich eine Skriptdatei erstellen

Möchten Sie eine neue Skriptdatei erstellen, haben Sie zwei Möglichkeiten. Sie können auf die auf der Begleit-CD-ROM im Ordner *\Beisp\Kap01* enthaltenen Vorlagen *VBScript.vbs* und *JScript.js* zurückgreifen. Kopieren Sie die beiden Dateien in einen lokalen Ordner der Festplatte. Anschließend laden Sie die gewünschte Vorlage in den Editor, ergänzen die gewünschten Programmanweisungen und speichern das Ergebnis unter einem neuen Namen.

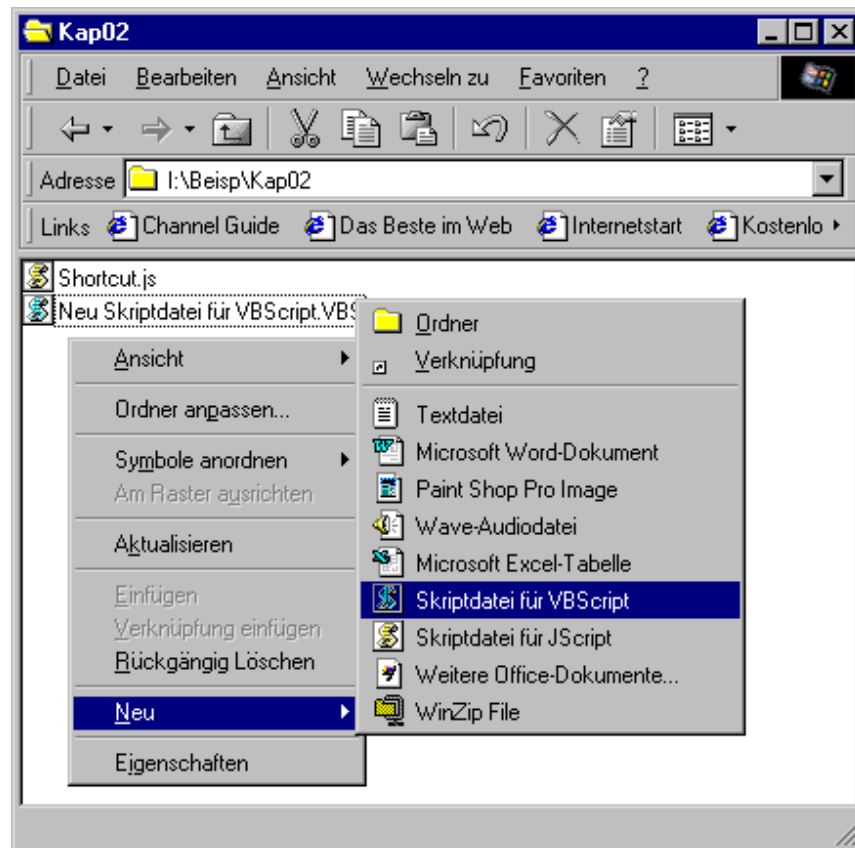
Die von der CD-ROM kopierten Dateien sind automatisch mit dem Attribut »Schreibgeschützt« versehen. Dies verhindert, dass Sie die Vorlagen irrtümlich überschreiben. Hinweis

Die Alternative besteht darin, dass Sie im Ordnerfenster mit der rechten Maustaste auf eine freie Stelle klicken und im Kontextmenü den Befehl *Neu/Textdatei* wählen. Anschließend benennen Sie die gerade erstellte Textdatei mit dem gewünschten Namen und der Dateinamenerweiterung *.vbs* oder *.js* (je nachdem welche Skriptsprache Sie im Programm verwenden). Die Windows-Warnung, dass sich durch das Umbenennen der Dateinamenerweiterung die Datei nicht mehr öffnen lässt, können Sie ignorieren. Anschließend können Sie die (leere) Textdatei in einen Editor laden und bearbeiten.

Eine Vorlage für Skriptdateien

Die beiden im obigen Abschnitt beschriebenen Verfahren sind nicht so ganz optimal. Was halten Sie davon, das Windows-Kontextmenü um zwei Einträge zu erweitern, über die sich automatisch VBScript- oder JScript-Dateien im Ordnerfenster (**Abbildung 2.2**) anlegen lassen? Im Grunde verfügen Sie bereits über alle Hilfsmittel, Sie müssen Windows nur noch so konfigurieren, dass das Betriebssystem die richtigen Vorlagen auch kennt.

Abbildung 2.2:
*Anlegen einer
neuen Skriptdatei
per Kontextmenü*



1. Öffnen Sie den Ordner *\Beisp\Kap01* von der Begleit-CD-ROM. Dieser Ordner enthält bereits die beiden Vorlagedateien. Da die Vorlagedateien auf der CD-ROM automatisch mit dem Schreibschutzattribut versehen werden, kopieren Sie die Dateien *VBScript.vbs* und *JScript.js* in einen lokalen Ordner auf der Festplatte. Anschließend löschen Sie das Schreibschutzattribut der beiden Dateien (Datei mit der rechten Maustaste anklicken, im Kontextmenü den Befehl *Eigenschaften* wählen und auf der Registerkarte *Allgemein* die Markierung des Kontrollkästchens für das Schreibschutzattribut löschen).
2. Haben Sie das Ordnerfenster mit den beiden Vorlagedateien geöffnet. Starten Sie jetzt das Hilfsprogramm Tweak UI (befindet sich in der Regel im Ordnerfenster der Systemsteuerung), und wählen Sie die Registerkarte *New*.
3. Ziehen Sie die beiden Vorlagedateien aus dem Ordnerfenster zur Registerkarte *New* (**Abbildung 2.3**). Sobald Sie die linke Maustaste loslassen, richtet Tweak UI einen Eintrag für die betreffende Datei auf der Registerkarte ein.

Wenn Sie die Registerkarte *New* über die *OK*-Schaltfläche schließen, übernimmt Tweak UI die Vorlagedateien in den Windows-Ordner *\ShellNew* und setzt auch die entsprechenden Registrierungseinträge (siehe /1, 2/ im Literaturverzeichnis). Öffnen Sie anschließend das Kontextmenü gemäß **Abbildung 2.2**, finden Sie bereits die neuen Befehle.

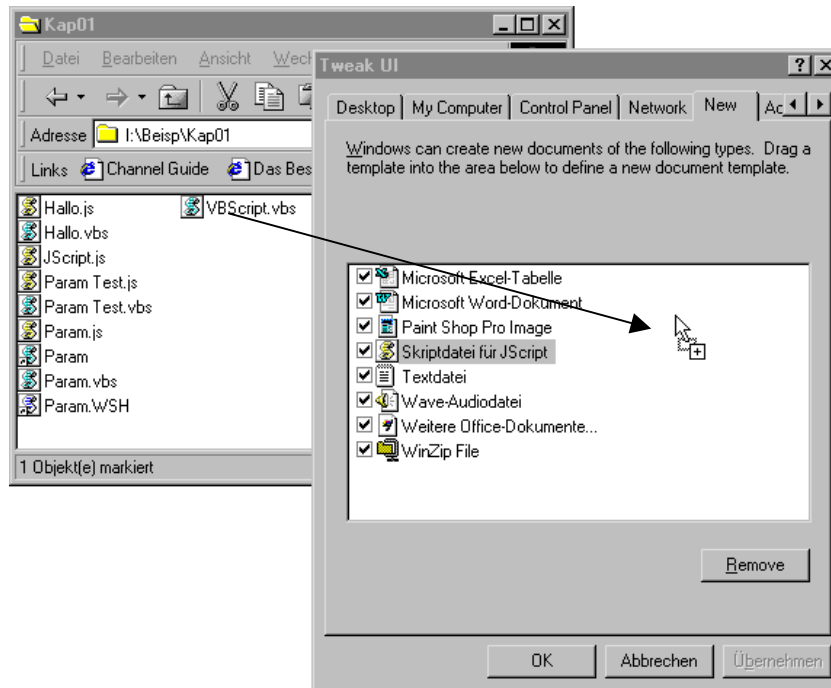


Abbildung 2.3:
Einrichten einer
Vorlage für
Skriptdateien

Das Hilfsprogramm Tweak UI müssen Sie explizit installieren. Windows 98- Benutzer finden dieses Modul auf der Windows-98-CD-ROM im Ordner *\tools\reskit\powertoy*. Windows 95- und Windows NT 4.0-Benutzer finden eine Version von Tweak UI auf der Begleit-CD-ROM zu diesem Buch im Ordner *\Tools\PowerToy\Tweakui*. Klicken Sie mit der rechten Maustaste auf die Datei *Tweakui.inf*, und wählen Sie im Kontextmenü den Befehl *Installieren*. Nach der Installation finden Sie das Tweak UI-Symbol im Ordnerfenster der Systemsteuerung. Hinweis

Haben Sie die Vorlagen für die Skriptdateien direkt von der Begleit-CD-ROM installiert, werden alle neuen Dateien mit dem Attribut »Schreibgeschützt« versehen. Über die Schaltfläche *Remove* der Registerkarte *New* können Sie einen Eintrag für die Vorlage wieder entfernen. Tweak UI löscht aber nicht die Vorlagedatei, die sich im Windows-Ordner *ShellNew* befindet. Wenn Sie anschließend die Vorlagedatei mit *Remove* entfernen und dann erneut installieren wollen, meldet Tweak UI einen Fehler (da der Zielordner noch die schreibgeschützte Datei aufweist). Sie müssen dann zunächst die schreibgeschützte Vorlagedatei aus dem Windows-Ordner *ShellNew* löschen, bevor Sie die Vorlage erneut installieren können. Wichtig

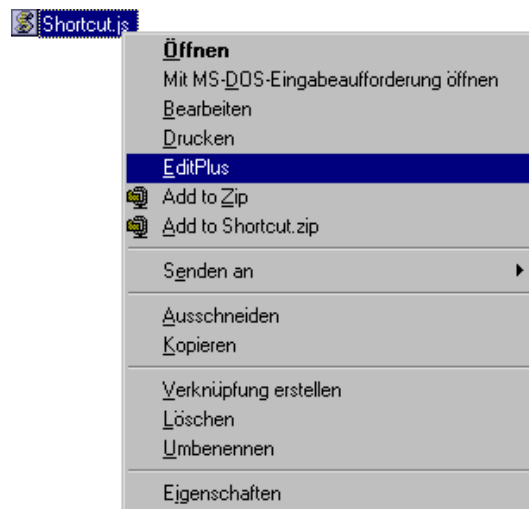
So können Sie Skriptdateien laden

Um eine bereits existierende Skriptdatei im Windows-Editor zu laden, klicken Sie mit der rechten Maustaste auf die Datei. Anschließend wählen Sie im Kontextmenü den Befehl *Bearbeiten*. Windows startet den Editor und lädt die Datei zum Bearbeiten. Bei der Registrierung des Windows Scripting Host wird der betreffende Befehl in der Windows-Registrierung eingetragen.

So richten Sie neue Befehle zum Bearbeiten ein

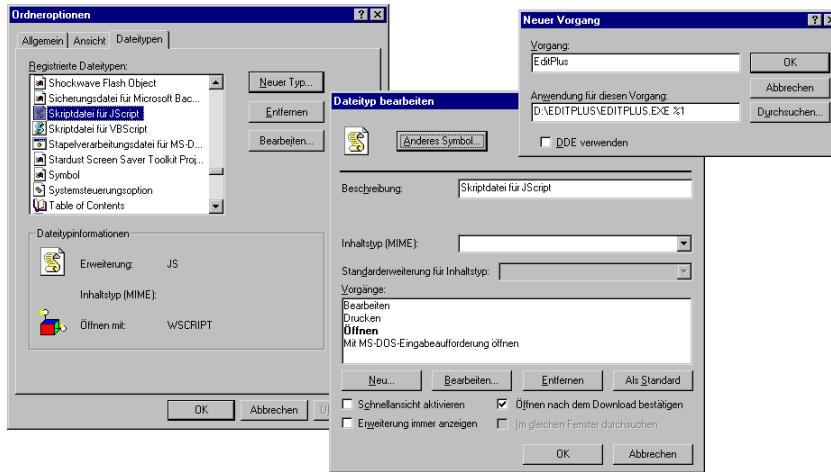
In den folgenden Abschnitten zeige ich Ihnen, warum sich der Windows-Editor nicht sonderlich zum Entwickeln von Skripten eignet. Weiterhin werden Ihnen einige Werkzeuge zur Skripterstellung vorgestellt. Allerdings stehen Sie vor der Frage, wie sich diese Werkzeuge im Kontextmenü der Skriptdateien eintragen lassen. Es soll ja nicht so sein, dass Sie erst das Programm starten und dann die Skriptdatei laden müssen.

Abbildung 2.4:
Kontextmenü zum
Bearbeiten der
Skriptdatei



In **Abbildung 2.4** sehen Sie im Kontextmenü den Befehl *EditPlus*, mit dem sich die Skriptdatei in dem nachfolgend vorgestellten Programm *EditPlus* zum Bearbeiten laden läßt. Um diese Erweiterung des Kontextmenüs vorzunehmen, gehen Sie in folgenden Schritten vor:

Abbildung 2.5:
Registrieren eines
Kontextmenübefehl
s



1. Wählen Sie im Ordnerfenster oder im Explorer-Fenster im Menü *Ansicht* den Befehl *Ordneroptionen*.
2. Suchen Sie auf der Registerkarte *Dateitypen* den Eintrag für die gewünschte Skriptdatei (**Abbildung 2.5**, links). Markieren Sie den Eintrag, und klicken Sie auf die Schaltfläche *Bearbeiten*.
3. Im Dialogfeld *Dateityp bearbeiten* (**Abbildung 2.5**, Mitte) wählen Sie die Schaltfläche *Neu*.
4. Anschließend tragen Sie im Dialogfeld *Neuer Vorgang* den Namen für den Kontextmenübefehl im Feld *Vorgang* ein (**Abbildung 2.5**, rechts).
5. Im Feld *Anwendung für diesen Vorgang* tragen Sie den ausführbaren Befehl (Pfad sowie Name der Anwendung) ein. Die Zeichen %1 stellen den Platzhalter für die aktuelle Datei dar.

Sobald Sie die Dialogfelder und Registerkarten schließen, wird der neue Dateityp registriert. Bei der nächsten Anwahl der betreffenden Dateien erscheint der neue Befehl im Kontextmenü.

Denken Sie daran, auch den jeweils zweiten Dateityp für Skriptdateien zu registrieren (d. h. *.vbs* und *.js*). Hintergrundinformationen zur Registrierung von Dateitypen finden Sie in den im Literaturverzeichnis unter /1, 2/ aufgeführten Titeln. Hinweis

Über den Kontextmenübefehl *Drucken* können Sie den Quellcode der Skriptdatei auf dem Drucker ausgeben. Sofern Sie einen der nachfolgend erwähnten Editoren verwenden, wird dieser Ausdruck mit Zeilennummern versehen, was beim Testen recht hilfreich ist. Tip

Hilfen zur Skriptbearbeitung

Das Erstellen von Skriptdateien im Windows-Editor ist eigentlich kein Problem. Wenn man von den spartanischen Möglichkeiten des Windows-Editors einmal absieht, können Sie die Anweisungen recht einfach eintragen, anpassen und wieder speichern. Die Probleme kommen jedoch beim Testen des Skripts. Der Windows Scripting Host liest den Quellcode des Skripts und parst die einzelnen Anweisungen. Wird eine fehlerhafte Anweisung gefunden, oder tritt ein Laufzeitproblem auf, erhalten Sie lediglich einen Fehlerdialog angezeigt, in dem die Zeilennummer der fehlerhaften Anweisung sowie ein Fehlerhinweis aufgeführt sind (**Abbildung 2.21**). Dann müssen Sie die Skriptdatei erneut im Windows-Editor laden und die fehlerhafte Anweisung korrigieren. Im Windows-Editor ist die Ermittlung dieser Zeile jedoch schwierig. Sie müssen von Hand die Zeilen zählen, was schnell zu Fehlern führt. Auf den folgenden Seiten möchten ich Ihnen daher einige Alternativen zur Skriptbearbeitung vorstellen, die Ihnen die Aufgabe des Zeilenzählens abnehmen.

Hinweis

Als »parsen« wird der Vorgang bezeichnet, bei dem der Interpreter der WSH-Sprach-Engine den Quellcode auf syntaktische Korrektheit analysiert. In einem zweiten Durchlauf werden dann die Anweisungen erneut geparkt, in einzelne Befehle zerlegt und ausgeführt.

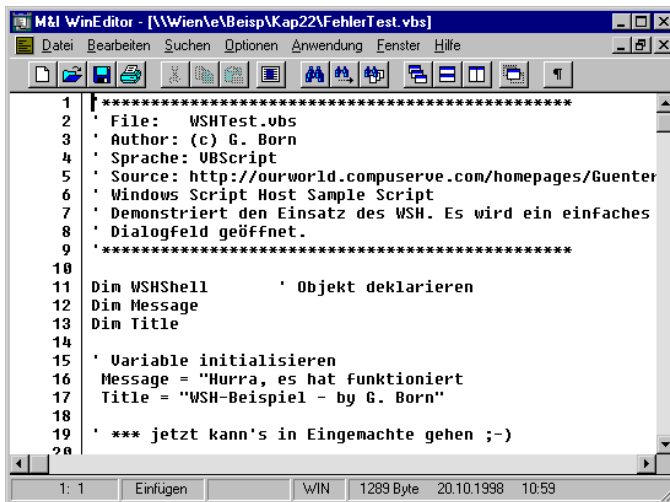
M&I WinEditor

Eine sehr elegante Lösung zur Skriptentwicklung bietet der M&I WinEditor von Ingo Paleit. Dieser deutschsprachige Editor erlaubt nicht nur die Bearbeitung mehrerer Dateien, sondern bietet neben vielen Funktionen (wie beispielsweise die Anzeige von Dateien im Hexadezimalmodus) eine zuschaltbare Zeilennummerierung (**Abbildung 2.6**). Sobald Sie also eine Skriptdatei mit diesem Editor öffnen, können Sie die fehlerhaften Anweisungen direkt anhand der Zeilennummer identifizieren.

Hinweis

Sie finden die Shareware-Version (30 Tage-Testversion) dieses recht kostengünstigen Programms auf der Begleit-CD-ROM im Ordner `\Tools\MIEdit`. Persönlich hat mir dieses Programm beim Einstieg in die WSH-Programmierung gute Dienste geleistet.

Abbildung 2.6:
*M&I WinEditor
mit
Zeilennummerierung
zum Erstellen eines
Skripts*



EditPlus

Eine englischsprachige Alternative zur Skriptbearbeitung bietet das Programm *EditPlus* (Abbildung 2.7).

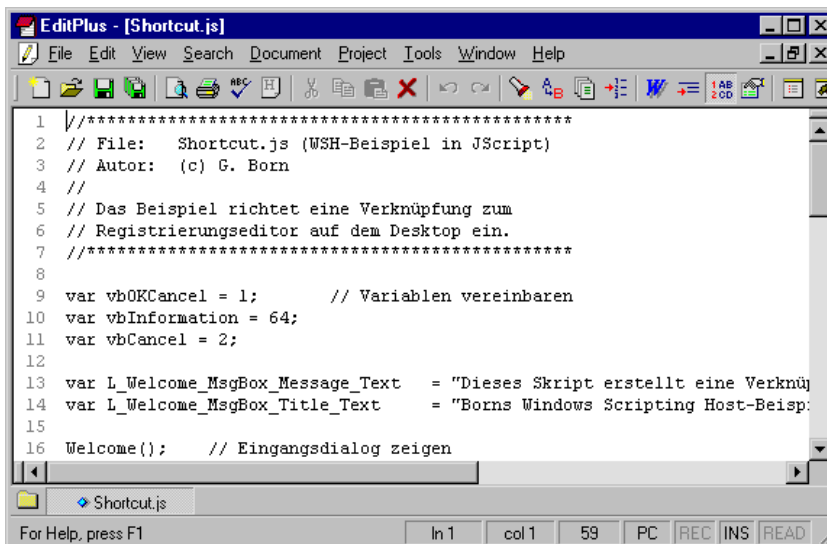


Abbildung 2.7:
*EditPlus mit
Zeilennummerierung
im Skript*

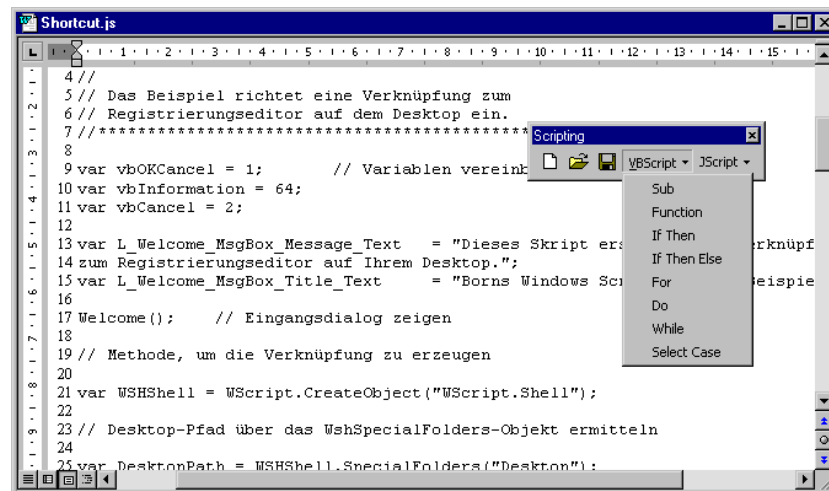
Bei diesem Editor können Sie ebenfalls eine Nummerierung zuschalten. Der Editor besitzt zusätzlich den Vorteil, dass Sie Vorlagen für verschiedene Skriptsprachen nutzen und selbst definieren können.

Sie finden die Shareware-Version (30 Tage-Testversion) dieses englischsprachigen Programms auf der Begleit-CD-ROM im Ordner `\Tools\EditPlus`.

Microsoft Word 97 als Skripteditor

Verfügen Sie über Microsoft Word, können Sie dieses Programm ebenfalls zur Skriptentwicklung einsetzen. In **Abbildung 2.8** sehen Sie ein Textfenster mit einem geladenen Skript, dessen Anweisungen mit Zeilennummern versehen sind.

Abbildung 2.8:
Word zur
Bearbeitung von
Skriptdateien



Eine Skriptdatei läßt sich in Word wie jede andere Textdatei laden, bearbeiten und wieder speichern. Die Anzeige der Zeilennummerierung nehmen Sie in Word 97 beispielsweise folgendermaßen vor:

1. Wählen Sie im Menü *Datei* den Befehl *Seite einrichten*.
2. Auf der Registerkarte *Seitenlayout* klicken Sie auf die Schaltfläche *Zeilennummern*.
3. Anschließend lassen sich die Optionen zur Zeilennummerierung im Dialogfeld *Zeilennummern* festlegen.

Beachten Sie aber, dass die Zeilennummern nur in der Seiten-Layout-Ansicht angezeigt werden.

Die Dokumentvorlage *Script.dot*

Sie können sich das Leben sogar noch etwas vereinfachen, indem Sie die auf der Begleit-CD-ROM im Ordner *\Tools\Template* enthaltene Datei *Script.dot* als Add-In in Microsoft Word 97 laden:

1. Hierzu öffnen Sie im Word-97-Fenster das Menü *Extras* und wählen den Befehl *Vorlagen und Add-Ins*. Word öffnet das Dialogfeld *Dokumentvorlagen und Add-Ins*.
2. Über die Schaltfläche *Hinzufügen* öffnen Sie ein Dialogfeld zur Auswahl der Add-In-Datei. Wählen Sie in diesem Dialogfeld das Laufwerk und den Pfad zur Vorlagedatei *Script.dot*. Dann markieren Sie die Vorlagedatei und schließen das Dialogfeld.

Im Dialogfeld *Dokumentvorlagen und Add-Ins* wird jetzt das Add-In *Script.dot* angezeigt, das zugehörige Kontrollkästchen ist markiert. Sie können jetzt das Dialogfeld schließen. Beim nächsten Start von Word 97 wird das Add-In automatisch geladen.

Sobald die Vorlagedatei als Add-In geladen ist, stehen die Funktionen der VBA-Prozeduren in der Datei *Script.dot* zur Verfügung. Interessant ist vor allem die *Scripting*-Symbolleiste. Diese läßt sich über den Befehl *Symbolleisten* im Menü *Ansicht* ein- oder ausblenden (wählen Sie den Befehl *Scripting* im Untermenü an). Auf der eingeblendeten Symbolleiste finden Sie drei Schaltflächen (**Abbildung 2.8**), um ein neues Skript zu erstellen, eine Skriptdatei zu öffnen oder eine Dokumentseite zu speichern:

- ♦ Die Schaltfläche *Neu* legt ein neues Dokument an. Bei diesem Schritt wird automatisch die Zeilennummerierung und die Layout-Ansicht eingeschaltet.
- ♦ Die Schaltfläche *Öffnen* blendet das Dialogfeld *Öffnen* ein. Gleichzeitig wird als Dateityp *Textdateien (*.txt)* eingestellt sowie das Suchmuster für Dateien auf **.vbs* und **.js* gesetzt. Dies erlaubt Ihnen direkt die Anzeige und Auswahl der Skriptdateien. Beim Laden wird dann die Zeilennummerierung eingeschaltet.
- ♦ Die Schaltfläche *Speichern* erlaubt Ihnen das Sichern der Skriptdateien. Eine geladene Skriptdatei wird in der Regel ohne weitere Nachfrage als Textdatei unter dem alten Namen gespeichert. Liegt ein neues Skriptdokument vor, müssen Sie nach Auswahl der Schaltfläche *Speichern* die Dateinamenerweiterung im Dialogfeld *Speichern unter* allerdings manuell von *.txt* auf *.vbs* oder *.js* umsetzen.

Die beiden Menüs *VBScript* und *JScript* enthalten Befehle, mit denen Sie bestimmte Sprachkonstrukte wie IF-Schleifen direkt im Quellcode einfügen können. Klicken Sie auf den Menüeintrag, und wählen Sie anschließend den

gewünschten Befehl aus (**Abbildung 2.8**). Word fügt dann die komplette Anweisung an der aktuellen Stelle im Skript ein.

Hinweis

Ich habe den Aufbau des Add-Ins bewußt einfach gehalten. Für interessierte Leser: die Datei *Script.dot* enthält mehrere Makros zur Realisierung dieser Funktionalität. Da der Quellcode nicht geschützt ist, haben Sie jederzeit die Möglichkeit, die Datei *Script.dot* direkt zu laden, das Fenster der Entwicklungsumgebung über die Funktionstasten Alt+F11 zu öffnen und die Makros zu erweitern bzw. anzupassen.

Tip

Die Verwendung von Word 97 als Skripteditor besitzt noch einen weiteren Vorteil: Erstellen Sie VBScript-Programme, können Sie jederzeit die Entwicklungsumgebung für VBA-Programme öffnen. Diese Entwicklungsumgebung läßt sich aber auch zum Erstellen von VBScript-Programmen »mißbrauchen«. Vor allem steht Ihnen der Objektkatalog zur Inspektion der Schnittstellen anderer Automatisierungsobjekte zur Verfügung. Auf diesen Sachverhalt komme ich auf den folgenden Seiten im Abschnitt ↗ »Hilfen zur Skriptentwicklung« in diesem Kapitel zu sprechen.

Weitere Editiermöglichkeiten

Neben dem Windows-Editor und den oben erwähnten Werkzeugen können Sie natürlich auch auf andere Programme zur Bearbeitung des Skriptquellcode ausweichen. So besitzt der nachfolgend vorgestellte Microsoft Script-Debugger (siehe ↗ Abschnitt »Skriptdateien testen« in diesem Kapitel) eine Funktion zur Bearbeitung des Quellcodes. Bei Anwahl einer Zeile wird auch die Zeilennummer in der Statusleiste eingeblendet. Persönlich finde ich es aber einfacher, wenn die Zeilennummern direkt am linken Rand im Editierfenster sichtbar sind.

Arbeiten Sie mit Visual Studio 6.0 (oder einer früheren Version)? Dann steht Ihnen wahrscheinlich das Programm *Visual InterDev* als Web-Entwicklungsumgebung zur Verfügung. In diesem Fall bietet es sich an, diese Entwicklungsumgebung zur Erstellung der Skripte zu verwenden. Über Makros können Sie sich die notwendigen Funktionen zur Eingabe der Befehle selbst schaffen.

Einige Editoren für Webseiten unterstützen direkt die Eingabe von JavaScript-Code. Dem Produkt Hot Metal Pro 4.0 liegt beispielsweise Acadia Infuse ScriptBuilder 1.0 bei (eine Entwicklungsumgebung für JavaScript). Mit diesen Erweiterungen lassen sich ggf. auch Skripte in der Sprache JScript sehr elegant erstellen.

Arbeiten Sie vorzugsweise mit VBScript, können Sie auf die nachfolgend beschriebene Entwicklungsumgebung für VB/VBA aufsetzen und verschiedenen Funktionen nutzen.

Hilfen zur Skriptentwicklung

Im vorhergehenden Abschnitt haben Sie einige Programme kennengelernt, mit denen die Eingabe des Quellcodes für Skripte unterstützt wird. Kern- und Angelpunkt war dabei die Anzeige einer Zeilennummerierung, um die Fehleranalyse zu erleichtern. Aber bei der Entwicklung von Skripten ist die Eingabe des Quellcodes nur die halbe Miete. Sie benötigen zusätzliche Informationen über die im Betriebssystem unterstützten Objekte. Weiterhin wäre es schön, wenn ggf. Informationen über die Objektschnittstellen, die verfügbaren Methoden und Eigenschaften etc. angezeigt werden. Nachfolgend möchte ich Ihnen noch einige Hilfsmittel und Techniken vorstellen, die bei der Skripterstellung recht hilfreich sind. Vorab: Sofern Sie allerdings nur ein oder zwei kleinere Skripte mit wenigen Zeilen schreiben, können Sie den folgenden Abschnitt übergehen. Der Aufwand zum Einstieg in die Technik lohnt sich nicht. Wer sich jedoch bereits mit Visual Basic auskennt oder mit Automatisierungsobjekten arbeiten möchte, wird die folgenden Ausführungen sicherlich mit Interesse zur Kenntnis nehmen. Welche der Werkzeuge bzw. Techniken Sie nutzen, bleibt dabei Ihnen überlassen.

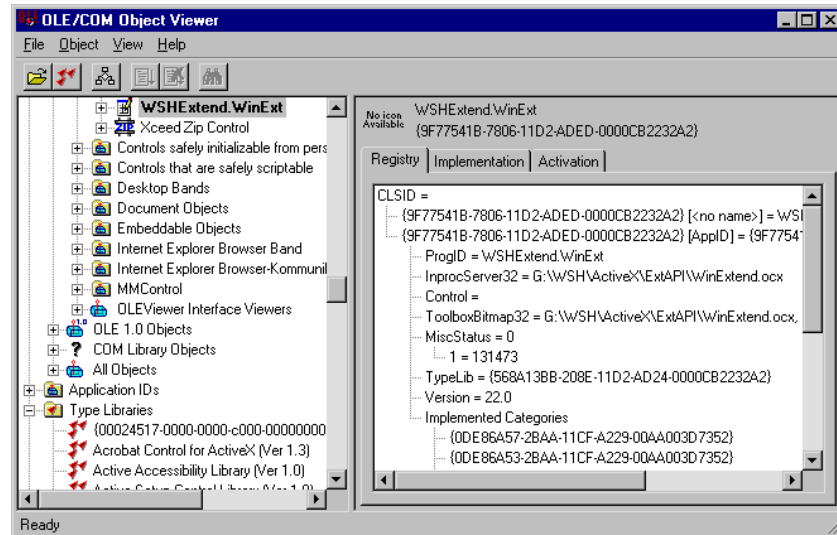
OLE/COM Object Viewer

Sobald Sie in einem Skript auf externe Automatisierungsobjekte zugreifen möchten, benötigen Sie Informationen über deren Schnittstellen. Sie müssen beispielsweise die Programm-Identifikationsbezeichnung (kurz auch *ProgID* genannt) kennen, um Objektreferenzen mit *GetObject* oder *CreateObject* anzulegen. Weiterhin ist es wichtig zu wissen, ob das Automatisierungsobjekt überhaupt unter Windows registriert ist.

Diese Informationen stellt Ihnen der Microsoft OLE/COM Object Viewer zur Verfügung. Dieses von den Microsoft Entwicklern geschriebene Programm analysiert die Windows-Registrierung und zeigt Ihnen die Objektklassen, die Anwendungs-IDs, die Typenbibliotheken (Type Libraries) sowie die (hier weniger interessierenden) Interface-Definitionen auf dem aktuellen System an (**Abbildung 2.9**). Sie können das Programm dabei ähnlich wie den Windows Explorer handhaben. Beim Aufruf zeigt der Viewer Ihnen im linken Fenster die verfügbaren Kategorien an Registrierungseinträgen an. Doppelklicken Sie auf ein Symbol, öffnet das Programm den Zweig mit den unterlagerten Informationen. Existieren keine Unterzweige mehr, blendet das Programm bei Anwahl eines Eintrags im linken Fenster die zugehörigen Registrierungseinträge im rechten Fenster ein. Der Viewer wertet hierzu lediglich die Registrierungseinträge im Zweig *HKEY_CLASSES_ROOT* aus. Für den Skriptentwickler sind diese Informationen aber recht interessant.

Sie erfahren beispielsweise die *ProgID*, unter der die betreffende Komponente unter Windows registriert ist. Diese Angabe benötigen Sie zum Zugriff auf das Objekt. Weiterhin zeigt Ihnen der Viewer den Pfad für den jeweiligen Inprocess-Server der Komponente an. Hierbei handelt es sich in der Regel um eine EXE-, DLL- oder OCX-Datei, aus der die Funktionen der Komponente geladen werden. Anhand der Pfadangabe können Sie sehr leicht herausfinden, ob die betreffende Komponente noch benötigt wird und wo die zugehörigen Dateien liegen. Dies ist zum Deinstallieren von ActiveX-Komponenten beispielsweise sehr hilfreich (↗ siehe die folgenden Seiten). Weiterhin benötigen Sie den Pfad zu einer OCX-Datei, falls Sie im Objektkatalog Informationen über die Automatisierungsschnittstelle aufrufen wollen.

Abbildung 2.9:
Anzeige des
OLE/Com Object
Viewer



Hinweis

Sie finden den OLE/COM Object Viewer auf der Begleit-CD-ROM zu diesem Buch im Ordner `\Tools\OleView`. Sie können das Programm *Oleview.exe* direkt von der CD-ROM laden.

Unterstützung bei der VBScript-Codeeingabe

Sofern Sie VBScript-Code zur Erstellung der Skripte verwenden, können Sie noch einige Tricks nutzen. In der VBScript-Laufzeitumgebung sind bereits eine Reihe benannter Konstanten definiert. Um beispielsweise die Schaltflächen und Symbole im Dialogfeld beim Aufruf der *MsgBox*-Funktion zu setzen, können Sie die folgende Konstante verwenden:

```
MsgBox "Hallo", 1+64, "WSH by Günter Born"
```

Diese Anweisung bewirkt, dass ein Dialogfeld mit dem vorgegebenen Titel und dem Text *Hallo* angezeigt wird. Dieses Dialogfeld besitzt weiterhin die

beiden Schaltflächen *OK* und *Abbrechen* sowie das *Info*-Symbol (**Abbildung 2.10**). Hätten Sie bei der Inspektion der obigen Anweisung sofort gewußt, welche Auswirkungen die Konstanten 1+64 besitzen? Ich muß jedenfalls ständig in der Dokumentation nachsehen.



Abbildung 2.10:
Beispiel für ein
Dialogfeld

Wesentlich einfacher geht es, wenn Sie mit symbolischen Konstanten arbeiten. Dann sieht die Anweisung folgendermaßen aus:

```
MsgBox "Hallo", vbInformation + vbOKCancel, "WSH by Günter Born"
```

Gleiches Ergebnis, aber ein wesentlich besser lesbarer Code. Auch ohne tiefgreifende VBScript-Kenntnisse dürfte klar sein, dass im Meldungsfeld ein *Information*-Symbol erscheinen soll. Weiterhin ist die Schaltflächenkombination *OK* und *Cancel* gewünscht. Es erfordert dabei vermutlich kaum Phantasie, den Begriff *Cancel* mit der deutschen Schaltflächenbeschriftung *Abbrechen* gleichzusetzen.

Bleibt noch die Frage, wie sich diese Konstante eingeben lassen. Meist ahnt man ja den Namen der Konstante, kennt aber die genaue Schreibweise nicht. Folglich sind Laufzeitfehler und Fehlfunktionen vorprogrammiert. Aber dies muß nicht unbedingt sein. Vieles läßt sich (dank der Microsoft-Entwickler) wesentlich intelligenter per Mausklick abrufen.

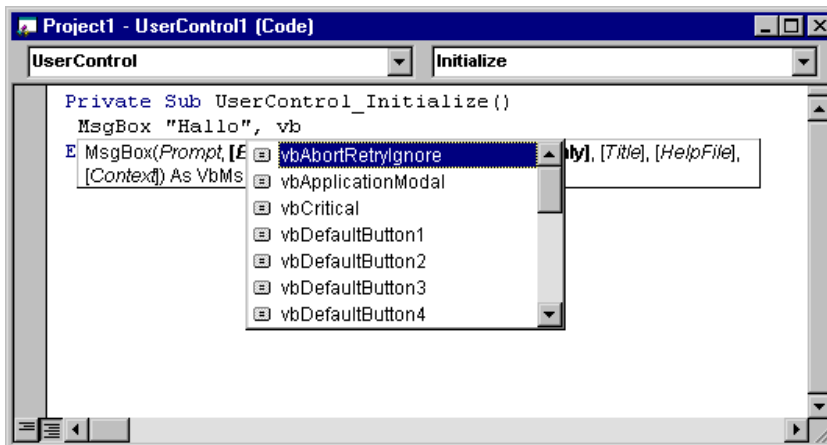


Abbildung 2.11:
Unterstützung bei
der VBScript-
Codeeingabe

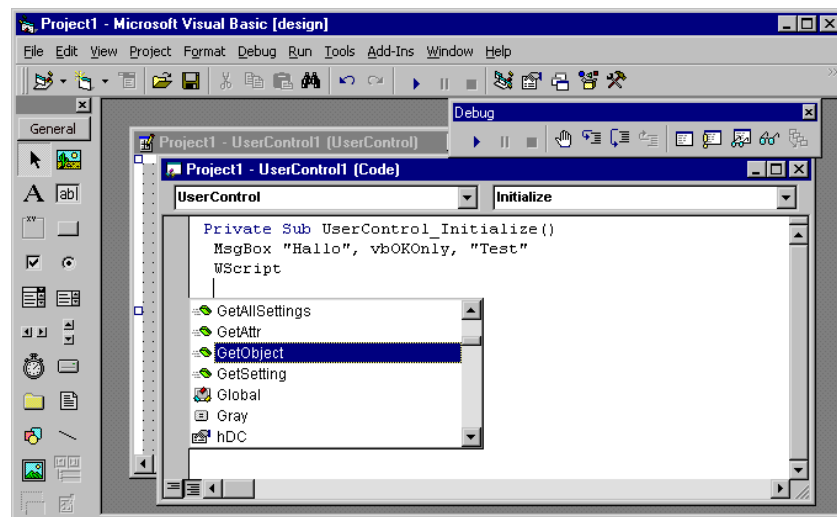
Sobald Sie eine der Microsoft-Entwicklungsumgebungen zur Unterstützung der Visual Basic-Dialekte verwenden, können Sie sehr viele Informationen über diese Sprache direkt abrufen. Nehmen wir an, Sie kennen sich mit den

Konstanten und der Syntax der Funktionsaufrufe noch nicht sonderlich gut aus. In **Abbildung 2.11** sehen Sie das Fenster zur Codeeingabe in einer solchen Entwicklungsumgebung. Im Codefenster wurde der Befehl *MsgBox* eingegeben. Sobald Sie dieses Schlüsselwort eintippen, blendet die Entwicklungsumgebung bereits ein Beispiel zur Nutzung der Funktion in einem QuickInfo-Fenster ein (allerdings sollten Sie wissen, dass in VBScript die beiden letzten Parameter *Helpfile* und *Context* entfallen). In **Abbildung 2.11** befindet sich die Anweisungszeile gerade in der Phase, in der die Konstanten einzugeben sind. Die Entwicklungsumgebung blendet dann die in Visual Basic definierten Konstanten in einem zweiten QuickInfo-Fenster ein. Sie brauchen dann nur noch einige Buchstaben einzutippen, um die richtige Konstante im Fenster anzuzeigen.

Tip

Wird das QuickInfo-Fenster nicht automatisch geöffnet, drücken Sie nach Eingabe des Befehls die Tastenkombination Strg+Leer. Verfügt die Entwicklungsumgebung über Informationen zu dem jeweiligen Thema, blendet diese das QuickInfo-Fenster ein. Alternativ können Sie die Stelle mit der rechten Maustaste anklicken und per Kontextmenü Informationen abrufen. Haben Sie die gewünschten Informationen erhalten bzw. die Konstante ausgewählt, läßt sich diese durch Drücken der Leer-Taste im Quellcode einfügen. Dies ermöglicht Ihnen, sehr schnell den benötigten Code einzutragen.

Abbildung 2.12:
Anzeige von
Methoden und
Eigenschaften in
der VB-
Entwicklungsum-
gebung



Aber es geht noch weiter. In einigen Fällen liefert Ihnen die Entwicklungsumgebung sogar noch Informationen zu den Methoden und Eigenschaften, die ein bestimmtes Objekt betreffen. In **Abbildung 2.12** ist ein solches QuickInfo-Fenster zu sehen. Die mit einem grünen Block gekennzeichneten Einträge stehen für Methoden, die stilisierte Karteikarte mit der Hand symbolisiert eine Eigenschaft.

Leider gibt es noch einen kleinen, aber gravierenden Haken: Diese Anzeigen der Entwicklungsumgebung gelten nur für Programme, die in dieser Umgebung erstellt werden. Folglich werden auch Methoden oder Eigenschaften angezeigt, die in VBScript nicht verfügbar sind. Aber wenn ich einmal voraussetze, dass Sie schon eine Ahnung haben, welche Objekte vorhanden sind, dann dürfte es auch ein leichtes sein, die gewünschten Einträge in den QuickInfo-Fenstern zu selektieren.

Vielleicht stellt sich Ihnen jetzt die Frage, wie Sie die oben gezeigten Techniken nutzen können, wo Sie an eine entsprechende Entwicklungsumgebung herankommen und wie der so erstellte Code in einem Skript genutzt werden kann. An dieser Stelle nochmals der Hinweis: Sofern Sie nur eine oder zwei kleinere Skripte mit wenigen Zeilen schreiben, brauchen Sie den ganzen Aufwand nicht zu betreiben. Auch wer täglich mit VBScript programmiert, entwickelt vermutlich andere Techniken, um Code Teile aus bestehenden Skripten in neue Programme zu übernehmen. Persönlich greife ich aber häufiger auf die Möglichkeiten der VB-Entwicklungsumgebungen zurück, um VBScript-Code mit bestimmten Konstruktionen schneller einzugeben.

Sie haben anschließend die Möglichkeit, den Programmcode über die Exportfunktion der jeweiligen Entwicklungsumgebung in eine Textdatei mit der Dateinamenerweiterung *.vbs* zu exportieren. In diesem Fall müssen Sie aber die Kopfzeile, die von der Entwicklungsumgebung automatisch hinzugefügt wird, wieder in einem Editor entfernen. Die zweite Möglichkeit besteht darin, den Quellcode der jeweiligen Prozeduren in der Entwicklungsumgebung einzugeben und dann per Zwischenablage in den Editor einzufügen, zu speichern und zu testen. Dies ist zwar nicht sonderlich optimal, aber immer noch besser, als stundenlang Bücher zu wälzen, um bestimmte Konstanten herauszufinden.

Letzte Frage: Wo bekommen Sie die Entwicklungsumgebungen her? Wer mit Visual Basic arbeitet, verfügt automatisch über eine solche Entwicklungsumgebung. Nutzen Sie Microsoft Word 97 (oder spätere Versionen)? Die mit Office 97 ausgelieferten Programme unterstützen Visual Basic für Applikationen (VBA). Verwenden Sie Word 97 zur Skriptentwicklung, genügt das Drücken der Tastenkombination Alt+F11, um zur Entwicklungsumgebung zu gelangen. Verfügen Sie weder über Word noch über Visual Basic? Auch dann geht die Welt nicht unter. Microsoft hat die Visual Basic 5 Control Creation Edition (VB 5 CCE) geschaffen, um die Entwicklung von ActiveX-Steuerelementen zu forcieren. Und diese VB 5 CCE befindet sich für die Leser dieses Buches zur freien Nutzung auf der Begleit-CD-ROM (Ordner *\Tools\Cce*). Bei der VB 5 CCE handelt es sich im Grund genommen um die Visual Basic 5.0-Entwicklungsumgebung, bei der lediglich die Funktion zum Erstellen von EXE-Programmen gesperrt wurde. Sobald Sie also die VB 5 CCE installieren und aufrufen, verfügen Sie über die betreffende Entwicklungsumgebung. Diese Umgebung werden wir in

diesem Buch noch nutzen, um ActiveX-Steuerelemente zur Erweiterung des Windows Scripting Host zu erstellen.

Wo gibt's Infos zu Automatisierungsobjekten

Mein größtes Problem beim Einstieg in die WSH-Skriptprogrammierung bestand in den fehlenden Informationen zu den Möglichkeiten. Nach etwas Suchen fand ich auch die in diesem Buch vorgestellten Informationen über die Objekte, die der Windows Scripting Host automatisch liefert. Aber dies ist ja nur die halbe Miete. Vielfach müssen Sie auf andere Automatisierungsobjekte zugreifen, um bestimmte Funktionen zu realisieren. Aber wo bekommen Sie diese Informationen her? An dieser Frage wäre ich beim Einstieg in die WSH-Skriptprogrammierung fast verzweifelt. Glücklicherweise verfügte ich über sehr umfangreiche Microsoft-Dokumentation zu Visual Basic und zur Visual Basic-Programmierung in Office 97. Weiterhin lagen Erfahrungen aus eigenen Buchprojekten zu diesen Themen vor. Aber trotzdem stand ich ständig vor der Frage: Wo gibt es Informationen zu einer bestimmten Schnittstelle, welche Parameter sind zu übergeben, welche Schnittstellen unterstützt ein Objekt etc.? Da ich in diesem Buch das Ziel verfolge, Ihnen möglichst viele Informationen zum Umfeld der Skriptentwicklung zu vermitteln, möchte ich Ihnen nachfolgend noch zeigen, wie sich mit dem Objektkatalog der Entwicklungsumgebung arbeiten und die Schnittstellen offenlegen. Profis, die Visual Basic bzw. VBA kennen, dürfen den folgenden Abschnitt daher gerne überspringen.

Der Objektkatalog, das Fenster zu Automatisierungsobjekten

Das gesamte Windows-Automatisierungskonzept für OLE, COM, ActiveX etc. beruht auf dem Ansatz, dass die Automatisierungsobjekte untereinander über bestimmte Schnittstellen kommunizieren. Ohne jetzt die Einsteiger mit allzuvielen Details zu langweilen, läßt sich doch eine Schlußfolgerung ziehen: Irgendwo muß es unter Windows Informationen geben, mit denen die betreffenden Automatisierungsmodule (allgemein als Objekte bezeichnet) Informationen über die Schnittstellen beschaffen. Die Einträge in der Registrierung sind hier nur die halbe Miete. Weiter oben wurde aber bereits gezeigt, dass das Programm OLE/COM Object Viewer Informationen aus der Registrierung ausliest. Inspizieren Sie einen solchen Eintrag, finden Sie Verweise auf die Dateien des betreffenden Inprocess-Servers (OLE-Server etc.). Nutzt eine Komponente ein Automatisierungsobjekt, greift es über den Registrierungseintrag auf den Inprocess-Server zu und handelt mit diesem die Schnittstellenmodalitäten aus.

Wenn Sie also ein anderes Automatisierungsobjekt im WSH-Skript verwenden wollen, müssen Sie die Schnittstellendefinitionen kennen. Diese Definitionen sind in verschiedenen Dateien des Servers hinterlegt. Sie

brauchen jetzt noch ein Werkzeug, welches diese Schnittstellendefinitionen automatisch erkennt und Ihnen möglichst die benötigten Informationen im Klartext anzeigt. Und genau dieses Werkzeug liefert Ihnen die VB/VBA/CCE-Entwicklungsumgebung in Form des Objektkatalogs. Dieser Objektkatalog läßt sich über die gleichnamige Schaltfläche der Symbolleiste ein- und ausblenden (**Abbildung 2.13**). Lassen Sie sich in **Abbildung 2.13** nicht von der Titelzeile des Objektkatalogs verwirren. Die in dieser Abbildung benutzte VB 5 CCE liegt nur in englisch vor, und dort heißt die Funktion nun mal »Object Browser«. Falls Sie jedoch mit VBA oder Visual Basic arbeiten, wird die betreffende Titelzeile »Objektkatalog« lauten.

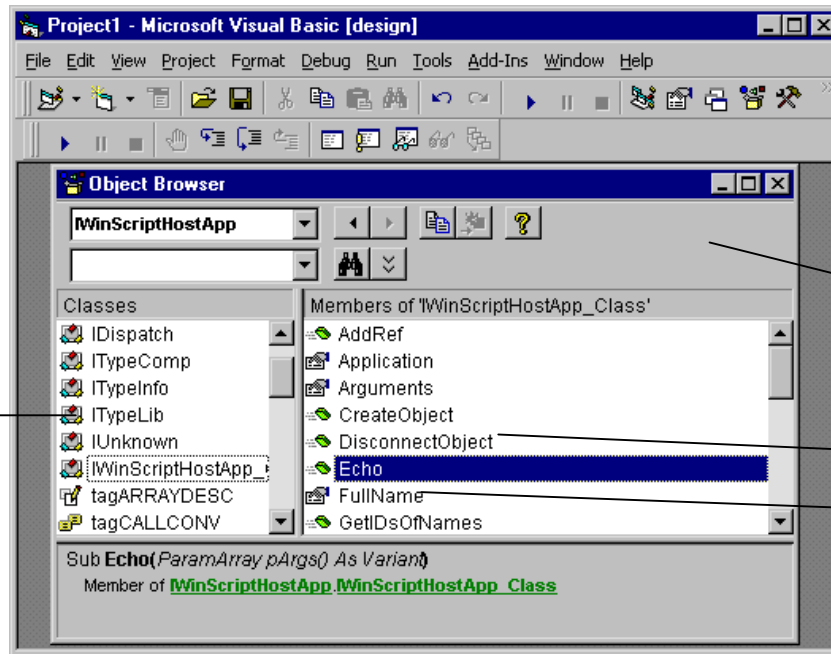


Abbildung 2.13:
VB 5 CCE mit
geöffnetem
Objektkatalog

Klassen

Objektkatalog

Methode

Eigenschaft

Der Objektkatalog listet im linken Fenster die aktuell im Projekt verfügbaren Klassen (Objekte) auf. Zu jeder Klasse sehen Sie dann im rechten Fenster die vom Objekt unterstützten Methoden (werden mit einem grünen Symbol markiert) und Eigenschaften (werden mit einer stilisierten Hand dargestellt).



Abbildung 2.14:
Auswahl der
Bibliotheken

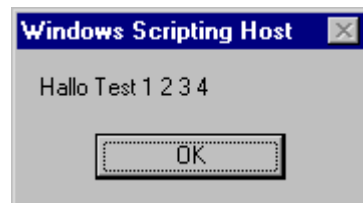
Welche Objekte im Fenster *Classes* angezeigt werden, läßt sich über das Listenfeld des Objektkatalogs einstellen (**Abbildung 2.14**). Wählen Sie den Eintrag »All Libraries«, zeigt der Objektkatalog alle Objekte an, die der Entwicklungsumgebung aktuell bekannt sind. Sie haben aber die Möglichkeit, die Auswahl auf eine bestimmte Bibliothek zu begrenzen, indem Sie den betreffenden Eintrag im Listenfeld wählen. Wie Sie Verweise auf zusätzliche Bibliotheken einrichten, wird im folgenden Abschnitt erläutert.

Klicken Sie auf ein Objekt, zeigt der Objektkatalog die Eigenschaften und Methoden an. Wählen einen solchen Eintrag aus, sehen Sie in der Fußzeile des Objektkatalogs die betreffende Definition des Eintrags. Bei einer Eigenschaft erhalten Sie Informationen, ob diese nur gelesen oder auch verändert werden kann, welcher Datentyp vorliegt und zu welcher Klasse die Eigenschaft gehört. Bei einer Methode liefert der Objektkatalog zusätzlich Hinweise zur Aufrufchnittstelle samt dem benötigten Parametern. In **Abbildung 2.13** wird beispielsweise die Schnittstelle der *Echo*-Methode angezeigt. Dort sehen Sie, dass die *Echo*-Methode als Parameter ein Feld (Parameterarray) akzeptiert. Dies wurde von mir gleich ausprobiert. Die folgende Anweisung:

```
WScript.Echo "Hallo" "Test" 1 2 3 4
```

wird vom Windows Scripting Host in das in **Abbildung 2.15** gezeigte Dialogfeld umgesetzt. Sie finden die Beispieldatei *EchoTest.vbs* im Ordner *\Beisp\Kap02* auf der Begleit-CD-ROM.

Abbildung 2.15:
Ausgabe mehrerer
Parameter im
Meldungsfeld



So beeinflussen Sie die Anzeige des Objektkatalogs

Der Objektkatalog zeigt Ihnen nur die Klassen (Objekte sowie deren Methoden und Eigenschaften) an, die aktuell in der Entwicklungsumgebung registriert sind.

Hinweis

Die Entwicklungsumgebung benötigt die Informationen über die im aktuellen Projekt benutzten Bibliotheken primär, um die Modulschnittstellen bereitzustellen und anschließend bei der Übersetzung der Module die Verweise auf diese externen Bibliotheken aufzulösen. Beim Erstellen eines VB-Programms läßt sich durch Nutzung der Objektvariable festlegen, ob die Anbindung an die externen Module bereits bei der Übersetzung oder erst zur Laufzeit erfolgt. Experten sind diese Mechanismen unter den Stichworten »Early Binding« und »Late Binding« geläufig (siehe auch /3/ im Literaturverzeichnis).

Unter Windows können aber wesentlich mehr Automatisierungsobjekte registriert sein. Gegebenenfalls müssen Sie daher vor dem Abrufen des Objektkatalogs die für die Skriptentwicklung erforderlichen Bibliotheken angeben. Hierzu bietet die Entwicklungsumgebung eine entsprechende Funktion.

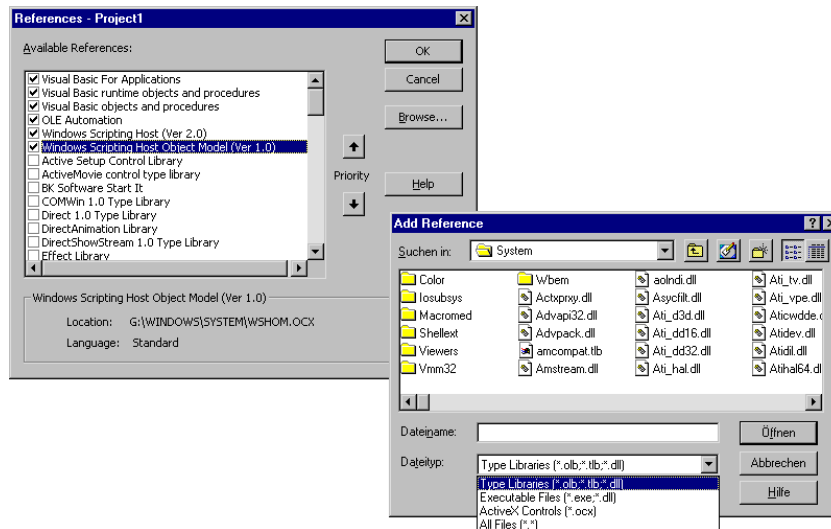


Abbildung 2.16:
Auswahl neuer
Verweise auf
Bibliotheken

1. Öffnen Sie das Dialogfeld *Verweise* (bzw. *References* in der englischen Version). Der Befehl *References* zum Aufruf des Dialogfelds befindet sich in der VB 5 CCE im Menü *Project*. In der VBA-Entwicklungsumgebung wählen Sie dagegen den Befehl *Verweise* im Menü *Extras*.
2. In **Abbildung 2.16** sehen Sie links oben das geöffnete Dialogfeld *Verweise* (bzw. *References*) in der englischen Fassung. In der Liste *Verfügbare Verweise* (bzw. *Available References*) finden Sie die Namen der Bibliotheken, die der Entwicklungsumgebung bekannt sind. Ein markiertes Kontrollkästchen vor dem Namen der Bibliothek signalisiert, dass deren Verweise im Projekt einbezogen werden und folglich auch im Objektkatalog angezeigt werden. Enthält die Liste eine benötigte Bibliothek, die nicht im Objektkatalog angezeigt wird, markieren Sie das zugehörige Kontrollkästchen.
3. Fehlt die Bibliothek in der Liste, ist noch nicht alles verloren. Klicken Sie auf die Schaltfläche *Suchen* (bzw. *Browse*) des Dialogfeldes. Die Entwicklungsumgebung öffnet ein zweites Dialogfeld (**Abbildung 2.16**, rechts), in dem Sie die gewünschte Bibliotheksdatei auswählen.

Sobald Sie das Dialogfeld schließen und das Kontrollkästchen im Dialogfeld *Verweise* markieren, zeigt der Objektkatalog die in der Bibliothek enthaltenen Klassen, Methoden und Eigenschaften an.

Hinweis Beginn

Für interessierte Leser schließt sich hier der Kreis. Weiter oben hatte ich im **7** Abschnitt »OLE/COM Object Viewer« bereits erwähnt, dass dieses Werkzeug den Pfad zum Inprocess-Server einer registrierten Automatisierungskomponente anzeigt. Diese Datei enthält aber die Beschreibung aller exportierten Schnittstellen. Sobald Sie also wissen, wo der Inprocess-Server der registrierten Automatisierungskomponente gespeichert ist, können Sie den Pfad zur betreffenden Datei angeben. Hierbei gibt es mehrere Dateien, aus denen die Entwicklungsumgebung die Schnittstellenbeschreibungen herausziehen kann:

- ◆ EXE, DLL: Dies sind die Dateien mit dem ausführbaren Programmcode der Inprocess-Server.
- ◆ OCX: In diesen Dateien werden ActiveX-Module gespeichert.
- ◆ TLB, OLB, DLL: Standardmäßig bezieht die Entwicklungsumgebung die Schnittstelleninformationen aus den Type Libraries. Diese werden in drei Dateitypen gehalten: *Technical Libraries* (TLB), *Object Libraries* (OLB) und *Dynamic Link Libraries* (DLL).

Die betreffenden Dateitypen lassen sich über den Dateityp im Listenfeld auswählen. Die einzige Schwierigkeit besteht darin, dass Sie den Pfad zu den betreffenden Dateien kennen müssen. Weiterhin liefert nicht jede Datei die benötigten Informationen in aller Ausführlichkeit. Trotzdem empfinde ich diese Funktion als äußerst hilfreich beim Zugriff auf Objekte und deren Schnittstellen.

[Hinweis Ende](#)

Auf der Windows-98-CD-ROM finden Sie beispielsweise im Ordner `\tools\reskit\sysfiles` eine Reihe von OCX-Dateien mit den Schnittstellen zu verschiedenen Systemfunktionen. Die Datei *Win.tlb* enthält beispielsweise alle Schnittstellen zum Zugriff auf das Windows-API. Die einzige Problematik bei der Nutzung dieser Schnittstellen besteht darin, dass die in VBScript bzw. JScript unterstützten Datentypen nicht mit den spezifizierten Parametertypen übereinstimmen. Daher empfiehlt es sich, die API-Aufrufe über ActiveX-Module zu kapseln.

ActiveX-Module installieren/deinstallieren

Ein komplexes Thema stellen ActiveX-Objekte dar, über die Sie die Funktionalität des Windows Scripting Host erweitern können. Sobald eine ActiveX-Komponente auf dem System registriert ist, können Sie aus einem Skript auf die von dieser Komponente exportierten Objekte, Methoden und

Eigenschaften zugreifen. Nachfolgend möchte ich zeigen, wie sich solche ActiveX-Komponenten installieren und wieder deinstallieren lassen.

ActiveX-Objekte installieren

Um die Objekte einer ActiveX-Komponente per Skript nutzen zu können, muß die ActiveX-Komponente auf dem lokalen Rechner vorliegen und registriert sein. Wie kommt man nun an solche ActiveX-Komponenten und wie lassen sich diese installieren? ActiveX-Objekte werden in OCX-Dateien hinterlegt. Sobald sich die OCX-Datei auf der Festplatte befindet, läßt sich diese zur Nutzung registrieren. Bei diesem Schritt werden die durch den bereits weiter oben erwähnten OLE/COM Object Viewer angezeigten Informationen in der Registrierung eingetragen.

Installation per Webseite

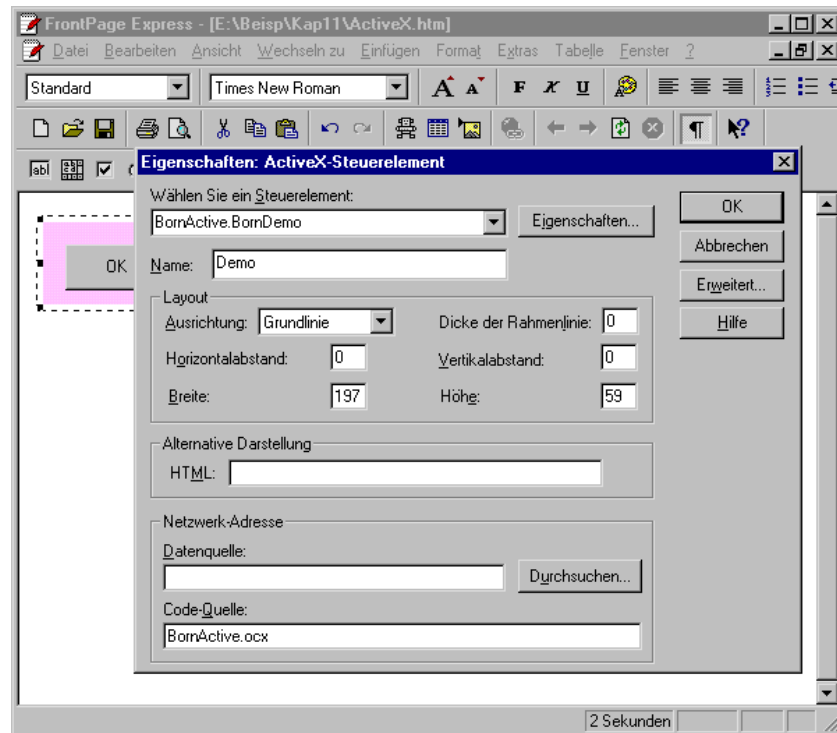
Die gebräuchlichste Variante zur Verteilung von ActiveX-Komponenten besteht darin, dass diese in Webseiten angeboten werden. Der Autor der Webseite bindet die ActiveX-Komponente als Objekt ein (siehe /6/ im Literaturverzeichnis sowie im folgenden Abschnitt) und spezifiziert einen Pfad zur OCX-Datei auf dem Webserver. Erkennt der Browser beim Zugriff auf die Seite das fehlende ActiveX-Element, wird dieses automatisch auf den lokalen Rechner geladen und registriert. Auf diese Weise kommen Sie ggf. auch an ActiveX-Komponenten mit Erweiterungen für den Windows Scripting Host. Wenn Sie eine Webseite mit solchen ActiveX-Objekten aufrufen, erscheint oft die Frage, ob die betreffenden Objekte installiert werden sollen. Alternativ bieten einige Webseiten die Möglichkeit, eine Archivdatei durch Anklicken eines Links herunterzuladen und diese dann per Installationsprogramm einzurichten.

So sorgt der Webautor für die Installation des ActiveX-Moduls

ActiveX-Komponenten wurden ursprünglich entworfen, um auf Webseiten aktive Inhalte präsentieren zu können. Je nach Objekt werden dabei sichtbare oder unsichtbare Inhalte dargestellt. Dass wir diese Technik zur Erweiterung des WSH nutzen, ist nur eine andere Seite der Medaille. Sobald eine Seite mit dem ActiveX-Modul vorliegt, können Sie dem Browser dessen Installation überlassen. Wie läßt sich dies im HTML-Code der Webseite realisieren? Nehmen wir an, Sie haben eine ActiveX-Komponente entwickelt und möchten diese jetzt per Web vertreiben. Erstellen Sie die betreffende Webseite mit der Komponente in FrontPage Express, gehen Sie folgendermaßen vor:

1. Öffnen Sie die zu erstellende Seite über FrontPage Express, und markieren Sie die Zeile, in der das Objekt einzufügen ist.
2. Wählen Sie im Menü *Einfügen* den Befehl *Andere Komponenten/ActiveX-Steuerelement*. FrontPage Express öffnet das Dialogfeld *Eigenschaften: Active-Steuerelement* (**Abbildung 2.17**).
3. Öffnen Sie das Listenfeld *Wählen Sie ein Steuerelement*, und wählen Sie eines der ActiveX-Steuerelemente aus der Liste aus. Mit diesem Schritt legt FrontPage bereits das ActiveX-Objekt samt CLSID-Code fest.

Abbildung 2.17:
Einfügen eines
ActiveX-Objekts in
einer Webseite mit
FrontPage Express



4. Tragen Sie eine Bezeichnung im Feld *Name* ein. Unter dieser Bezeichnung läßt sich das Objekt ggf. im HTML-Dokument ansprechen.
5. Geben Sie im Feld *Code-Quelle* die URI zur OCX-Datei der betreffenden ActiveX-Komponente an. Befindet sich die OCX-Datei im gleichen Ordner wie das HTML-Dokument, genügt die Eingabe des Dateinamens.
6. Schließen Sie das Eigenschaftenfenster über die *OK*-Schaltfläche. Anschließend passen Sie die Größe des ActiveX-Steuerelements an.

Sobald die Seite fertiggestellt wurde, speichern Sie diese lokal oder auf einem Webserver. FrontPage Express generiert für jedes im Dokument enthaltene ActiveX-Modul den folgenden HTML-Code.

```
<object id="Demo" name="Demo"
classid="clsid:37DDEAA9-5235-11D2-ADED-0000CB2232A2"
codebase="BornActive.ocx" align="baseline" border="0" width="197"
height="59"></object>
```

Listing 2.2:
HTML-Code zum
Einbinden eines
ActiveX-
Steuerelements

Das Steuerelement wird durch den <OBJECT>-Tag im Dokument eingebunden. Das *classid*-Attribut gibt den CLSID-Code der betreffenden ActiveX-Komponente an. Dieser CLSID-Code wird von FrontPage Express bei der Auswahl des Steuerelements über das Feld *Wählen Sie ein Steuerelement* aktuell aus der Registrierung abgerufen und eingesetzt.

ActiveX-Objekte werden in Dateien mit der Erweiterung *.ocx* gespeichert. Um die Steuerelemente in FrontPage Express beim Erstellen der Webseite abrufen zu können muß die OCX-Datei auf dem Rechner vorhanden und als ActiveX-Komponente registriert sein. Haben Sie das Objekt selbst auf dem Rechner erstellt, führt die Entwicklungsumgebung dies automatisch für Sie durch. Verfügen Sie lediglich über eine OCX-Datei einer fertigen ActiveX-Komponente, verwenden Sie die nachfolgend im **7** Abschnitt »So können Sie eine lokale OCX-Datei auch registrieren« gezeigte Technik zum Registrieren. Wichtig

Bleibt noch die Frage: Was passiert, wenn ein Benutzer die Seite aus dem Web lädt? Wird die ActiveX-Komponente angezeigt oder erscheint lediglich ein Platzhalter? Ist das ActiveX-Steuerelement bereits beim Benutzer installiert, findet der Browser dieses in der Registrierung und benutzt es. Meist wird die Komponente jedoch nicht installiert sein. In diesem Fall würde der Benutzer lediglich einen Platzhalter sehen (ähnlich wie bei der Grafikanzeige, wenn die Datei fehlt). Sie müssen daher als Webautor dafür sorgen, dass die fehlende OCX-Datei auf den Rechner des Benutzers geladen und als ActiveX-Objekt registriert wird. Dies erfolgt durch die Angabe der URI-Adresse zur OCX-Datei im Feld *Code-Quelle* (**Abbildung 2.17**). FrontPage fügt diese URI als Attributwert in *CodeBase* ein. Die Anweisung:

```
codebase="BornActive.ocx"
```

besagt, dass das Objekt aus dem gleichen Verzeichnis wie das HTML-Dokument zu laden ist. Rufen Sie das HTML-Dokument später auf, erkennt der Browser das fehlende ActiveX-Objekt. Anhand der *CodeBase*-Angabe wird die OCX-Datei geladen und anschließend automatisch als DCOM-Objekt in der Registrierung angezeigt.

ActiveX-Komponente per Entwicklungsumgebung registrieren

Sobald Sie ActiveX-Objekte mit einer Entwicklungsumgebung (z. B. mit Visual Basic oder mit der auf der Begleit-CD-ROM enthaltenen Control Creation Edition) selbst erstellen, wird die betreffende Komponente als OCX-Datei in einem Ordner auf der lokalen Festplatte hinterlegt. Dies passiert, sobald Sie das Projekt in eine OCX-Datei übersetzen. Gleichzeitig übernimmt

die Entwicklungsumgebung automatisch die Registrierung des Moduls (dabei wird der dem ActiveX-Modul zugewiesene ClassID-Code in die Registrierung eingetragen). Diesen Ansatz nutzen Sie beispielsweise, wenn Sie ActiveX-Komponenten mit der auf der Begleit-CD-ROM mitgelieferten VB 5 CCE erstellen.

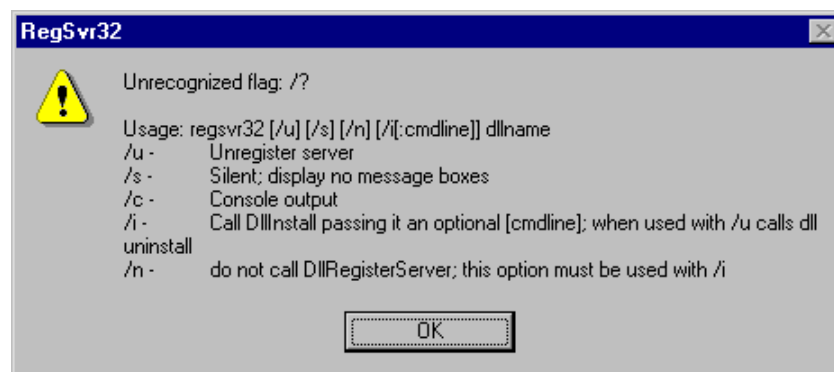
So können Sie eine lokale OCX-Datei auch registrieren

Verfügen Sie über eine fertige ActiveX-Datei mit der Erweiterung *.ocx*? Um diese Datei in FrontPage Express oder als ActiveX-Steuerelement im WSH abrufen zu können, muß das Steuerelement als Objekt registriert sein. Wie läßt sich so etwas aber bewerkstelligen, speziell wenn Sie über keine Entwicklungsumgebung verfügen? Dies ist beispielsweise der Fall, wenn die fertige OCX-Datei mit den Skriptdateien per Diskette weitergegeben wird. Eine Möglichkeit besteht darin, dass Sie von der Entwicklungsumgebung eine Installationsdatei erstellen lassen. Oder Sie nutzen die in den vorherigen Abschnitten beschriebenen Techniken. Aber es gibt eine wesentlich einfachere Lösung: Hier kommt das Programm *RegSvr32.exe* zum Tragen. Sie können das Programm mit der Anweisung:

RegSvr32.exe /?

aufgerufen. Das Programm meldet zwar einen Fehler, der Schalter */?* wird nicht erkannt, aber es zeigt ein Dialogfeld mit den verfügbaren Optionen an (**Abbildung 2.18**).

Abbildung 2.18:
Schalter und
Optionen des
Programmes
RegSvr32



Um den Inhalt der OCX-Datei als ActiveX-Komponente zu registrieren, reicht es aus, wenn Sie beispielsweise den Befehl:

RegSvr32.exe C:\Beisp\Kap02\BornTest\BornActive.ocx

eingeben. Das Programm sucht dann die Datei im angegebenen Pfad, wertet deren Inhalt aus und trägt alle erforderlichen Informationen in die Registrierung ein. Sie können anschließend die betreffenden ActiveX-Komponenten direkt im WSH-Skript nutzen.

RegSvr32.exe ist Bestandteil aller Microsoft Entwicklungsumgebungen und erlaubt das Registrieren von OCX-Modulen. Glücklicherweise wird das Programm aber auch direkt unter Windows 98 im Systemordner installiert. Dies bedeutet, Sie können das Programm direkt aufrufen. Hinweis

Die Registrierung für eine OCX-Datei entfernen

In den vorherigen Abschnitten wurde gezeigt, wie der Internet Explorer eine in einem HTML-Dokument enthaltene ActiveX-Komponente automatisch installiert und in der Registrierung einträgt. Sie haben auch erfahren, dass die Entwicklungsumgebung (z. B. VB 5 CCE) beim Erstellen einer solchen Komponente diese automatisch registriert. Daher stellt sich die Frage, wie Sie später die OCX-Komponente wieder entfernen können. Es genügt sicherlich nicht, wenn Sie lediglich die OCX-Datei löschen. Spätestens wenn Sie etwas im Internet gesurft sind oder einige VB-Beispiele angesehen haben, wimmelt Ihre Registrierung von nicht mehr benötigten Einträgen für ActiveX-Komponenten.

ActiveX-Komponenten entfernen: Methode 1

Haben Sie eine Entwicklungsumgebung (z. B. Visual Basic) für ActiveX-Objekte installiert? Wimmelt Ihre Registrierung von »toten« Einträgen für solche DCOM-Objekte (die aus diversen Tests zurückgeblieben sind)? In den Anfangstagen habe ich diese »Leichen« in mühevollen Handarbeit aus der Registrierung entfernt. Dies ist zwar auch heute noch in einigen wenigen Fällen notwendig. Zwischenzeitlich wende ich aber intelligentere Techniken an. Ein Blick auf **Abbildung 2.18** zeigt bereits die erste Lösung. Sie benötigen hierzu das Programm *RegSvr32.exe*. Geben Sie unter Windows den folgenden Befehl ein:

```
RegSvr32.exe /u C:\Beisp\Kap02\BornTest\BornActive.ocx
```

sucht das Programm das im Pfad angegebene OCX-Modul. Der Schalter */u* veranlaßt das Programm, die OCX-Datei auszuwerten und alle Registrierungsinformationen aus der Registrierung zu löschen. Anschließend können Sie die OCX-Datei löschen, es bleiben keine »Leichen« im System zurück.

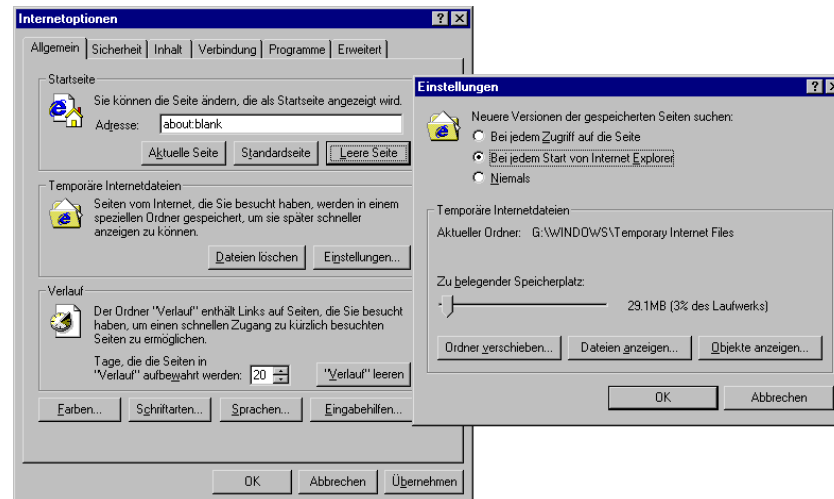
Möchten Sie häufiger eine OCX-Datei registrieren und wieder deinstallieren? Tip
Dann empfiehlt es sich, den betreffenden Befehl in einer Stapelverarbeitungsdatei abzulegen und die BAT-Datei im Verzeichnis der OCX-Datei zu speichern. Mit dem Befehl *Start RegSvr32.exe BornActive.ocx* wird das aktuelle Modul registriert. Der Befehl *Start* bewirkt den Aufruf des Windows-Programms. Der Pfad kann entfallen, da die BAT-Datei im gleichen Ordner wie die OCX-Datei vorliegt. Sie finden im Ordner *\Beisp\Kap02\BornTest* der Begleit-CD-ROM die beiden Dateien

RegisterOcx.bat und *UnRegisterOcx.bat*, mit denen Sie die OCX-Datei *BornActive.ocx* registrieren und wieder aus der Registrierung austragen können.

ActiveX-Komponenten entfernen: Methode 2

Wurde die ActiveX-Komponente durch Anwahl einer Webseite vom Internet Explorer installiert? In diesem Fall lädt Windows 98 (bzw. der Internet Explorer) die zugehörige Datei in den Ordner *Temporäre Internetdateien*. Gleichzeitig wird das betreffende Modul in der Registrierung als DCOM-Objekt eingetragen. Möchten Sie ein auf diesem Weg registriertes ActiveX-Modul entfernen? Dann gehen Sie in folgenden Schritten vor:

Abbildung 2.19:
Einstellen der
Internetoptionen



1. Starten Sie den Internet Explorer. Anschließend wählen Sie im Menü *Ansicht* den Befehl *Internetoptionen*.
2. Wählen Sie auf der Registerkarte *Allgemein* die Schaltfläche *Einstellungen* der Gruppe *Temporäre Internetdateien* (**Abbildung 2.19**, links).
3. Windows öffnet ein zweites Dialogfeld *Einstellungen*, auf dem Sie die Schaltfläche *Objekte anzeigen* anklicken (**Abbildung 2.19**, rechts).
4. Jetzt öffnet Windows das Ordnerfenster mit den Inhalt des Ordners *Downloaded Program Files* (**Abbildung 2.20**). Dieser Ordner enthält die Objekte, die bei Internetsitzungen heruntergeladen und anschließend in die Registrierung eingetragen wurden.
5. Klicken Sie mit der rechten Maustaste auf das zu entfernende Objekt. Windows öffnet das Kontextmenü mit den auf das Objekt anwendbaren Befehlen. Wählen Sie den Befehl *Entfernen*.

Nach einer Sicherheitsabfrage, die Sie mit der *Ja*-Schaltfläche bestätigen, entfernt Windows bzw. der Browser das betreffende Objekt. Dies bedeutet, dass die Datei gelöscht wird und gleichzeitig auch die Registrierungseinträge dieses Objekts entfernt werden.

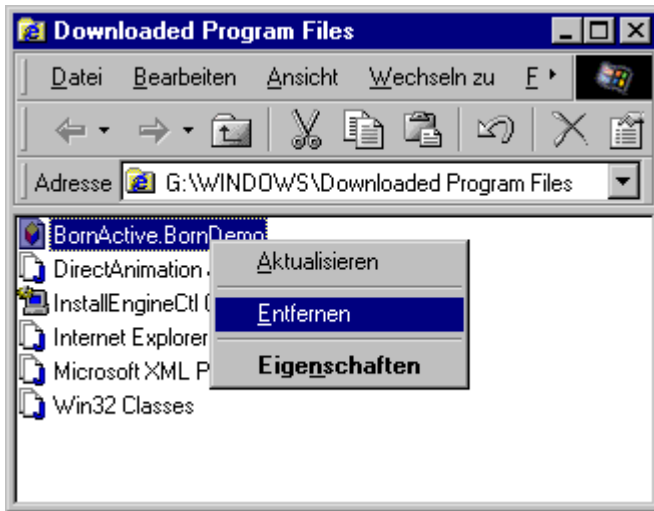


Abbildung 2.20:
Ordner
Downloaded
Program Files

Bei diesen Objekten muß es sich nicht ausschließlich um ActiveX-Objekte in Form von OCX-Dateien handeln. In diesem Ordner finden Sie auch Java-Klassenmodule in Form von CAB-Archiven. Sie können die Registrierungseinträge anschließend direkt mit dem COM/OLE Object Viewer kontrollieren. Hinweis

Einige ActiveX-Objekte werden aber mit einem eigenen Installationsprogramm geliefert. Dann finden Sie meist einen Eintrag zum Deinstallieren der Komponente auf der Registerkarte *Installieren/Deinstallieren* (Symbol *Software* in der Systemsteuerung).

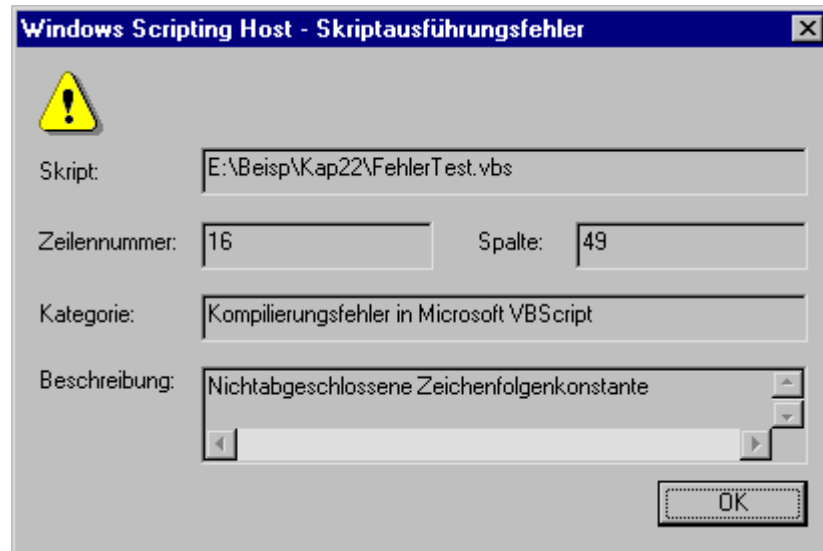
Skriptdateien testen

Das Testen von WSH-Skriptdateien ist ohne Hilfsmittel eine recht mühsame Angelegenheit. Der nachfolgende Abschnitt zeigt Ihnen einige Möglichkeiten, mit denen sich Skriptdateien testen lassen. Dabei gilt es sowohl Syntaxfehler als auch logische Fehler oder Laufzeitfehler aufzudecken.

So zeigt der WSH auftretende Fehler an

Starten Sie ein Skript, welches einen Fehler enthält, meldet der Windows Scripting Host dies in einem Fehlerdialog (**Abbildung 2.21**). Das betreffende Dialogfeld enthält den Pfad und den Namen des Skripts sowie einen Hinweis auf die Fehlerkategorie samt Fehlerbeschreibung. Es wird teilweise sogar die Zeilennummer und die Spalte gemeldet, in der der Fehler auftrat.

Abbildung 2.21:
Fehlerdialog bei
der
Skriptausführung



Beim Testen wird dabei noch unterschieden, ob es sich um Laufzeitfehler handelt, oder ob eine fehlerhafte Anweisung vorliegt, die bereits bei der Übersetzung zu dem in **Abbildung 2.21** gezeigten Dialog führt. In beiden Fällen heißt es für Sie als Entwickler, die Skriptdatei im Editor zu laden und die fehlerhafte Zeile zu korrigieren.

Bei Syntaxfehlern in den Anweisungen führt kein Weg an der obigen Vorgehensweise vorbei. Sie müssen das Skript im WSH starten, damit dieser beim Parsen des Quellcodes die Syntax prüft und ggf. die Fehler anzeigt. Sofern Sie die Skriptdateien mit dem weiter oben erwähnten M&I WinEditor oder einem der anderen Werkzeuge bearbeiten, erhalten Sie über die Zeilennummern zumindest einen schnellen Hinweis auf die fehlerhafte Zeile. Am besten positionieren Sie das Ordnerfenster und das Fenster des Editors nebeneinander.

Auf der Begleit-CD-ROM finden Sie beispielsweise die Datei *FehlerTest.vbs* im Ordner *\Beisp\Kap02*. In dieser Datei wurde der in **Abbildung 2.21** in Zeile 16 gezeigte Fehler bewußt eingebaut. Die Zeichenfolgenkonstante in dieser Zeile wurde nicht mit einem Anführungszeichen abgeschlossen. Rufen Sie das Skript auf, löst der WSH den Fehler aus. Sie müssen dann den Fehler korrigieren, das Ergebnis speichern und das Skript erneut ausführen.

Programmablauf verfolgen

Sobald das WSH-Skript fehlerfrei geladen wird, kann mit dem Funktionstest begonnen werden. Selten tritt der Idealfall auf, dass das Skript das erwartete Ergebnis liefert. Sofern Sie nur sehr einfache Skripte haben, kann eine Ausgabe von Meldungen und Zwischenergebnissen in Dialogfeldern recht hilfreich sein. Das folgende Listing zeigt diese Technik.

Listing 2.3:
*Skript mit
Testausgaben*

```
'*****
' File:   WSHDemo.vbs (WSH-Beispiel in VBScript)
' Autor:  (c) Günter Born
'
' Zweck:  Zeige, wie sich der Programmablauf mit
' Meldungen verfolgen läßt.
'*****
' Die folgenden Zeilen schalten die Ausgabe der
' Testmeldungen ein oder aus. Sie müssen nur den
' Kommentar in die entsprechende Zeile setzen.

' DebugFlag = false   ' Testausgabe abgeschaltet
DebugFlag = true      ' Testausgabe zulassen

j = 0
debug "Start", 0, 0

For i = 1 to 10      ' versuche 10 Durchläufe
  debug "Durchlauf: ", i, j
  j = j + i          ' Addiere alle Zahlen
Next

debug "Ende", i, j

WScript.Echo "Ergebnis: ", j

WScript.Quit

Sub debug (text, count, val)
  If DebugFlag Then   ' Debug-Modus aktiviert?
    WScript.Echo text, i , "Teilergebnis: ", j
  End if
End sub
'*****
```

```

'***           Ende -> WSH-VBScript           ***
'*****

```

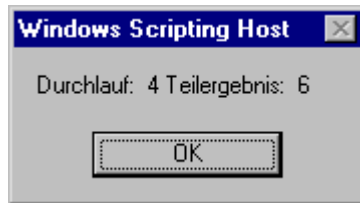
Die eigentliche Programmfunktion ist recht trivial: Das Programm berechnet in einer Schleife die Summe der Zahlen von 1 bis 10. Dieses Ergebnis wird anschließend mittels der *Echo*-Methode angezeigt.

Für Testzwecke möchte Sie aber wissen, wann das Programm in die Schleife eintritt und wieviele Schleifendurchläufe wirklich ausgeführt werden. Hierzu läßt sich in der Schleife oder an den betreffenden Stellen ebenfalls die *Echo*-Methode aufrufen. Um den Aufbau des Programmes nicht zu stark durch diese Ausgabeanweisungen zu »verwässern«, habe ich hier einen besonderen Ansatz gewählt. Alle Testausgaben wurden in eine eigene Prozedur mit dem Namen *debug* ausgelagert. Daher braucht im Programm zur Testausgabe nur noch die Anweisung:

```
debug "Start", 0, 0
```

hinterlegt zu werden. Der erste Parameter definiert den (links erscheinenden) Text, der im Meldungsfeld erscheinen soll. Die beiden anderen Parameter können Zahlen aufnehmen, die ebenfalls im Meldungsfeld ausgegeben werden. Wenn Sie das obige Listing ansehen, werden Sie die Testanweisungen an verschiedenen Stellen sehen. Innerhalb der Schleife werden beispielsweise der Schleifenindex *i* und der berechnete Zwischenwert angegeben (**Abbildung 2.22**).

Abbildung 2.22:
Meldung beim
Testen



Nun kommt aber noch eine Besonderheit zum Tragen: Solange Sie das Programm testen, sollen die Fehlerwerte gemäß **Abbildung 2.22** angezeigt werden. Sobald aber das Programm fehlerfrei läuft, stören diese Ausgaben. Sie sind dann gezwungen alle Testanweisungen aus dem Programm zu entfernen. Ändern Sie später etwas am Programm, müßten Sie die Testanweisungen erneut einbauen.

Diesen ganzen Weg können Sie sich sparen, indem Sie die Ausgabe der Testanweisungen in Abhängigkeit von einer Option steuern. In obigem Listing wird in der Prozedur *debug* eine Abfrage der Art:

```
If DebugFlag Then ' Debug-Modus aktiviert?
```

ausgeführt. Nur wenn der Wert der globalen Variable *DebugFlag* auf *true* gesetzt ist, werden die *Echo*-Ausgaben angezeigt. Diese Variable wird im Kopf des Skripts auf den Wert *true* gesetzt.

```
' DebugFlag = false ' Testausgabe abgeschaltet
```

```
DebugFlag = true      ' Testausgabe zulassen
```

An dieser Stelle finden Sie aber auch die mit einem Kommentar versehene Anweisungen, um die Variable auf den Wert *false* zu setzen. Dieser Wert unterdrückt die Ausgabe.

Sie finden das Testprogramm *WSHDemo.vbs* (bzw. *WSHDemo.js*) im Ordner *\Beisp\Kap02* auf der Begleit-CD-ROM. JScript 3.0 unterstützt im Internet Explorer 4.0 zusätzlich die Möglichkeit der bedingten Übersetzung. Hinweise zu den jeweiligen Anweisungen liefert die JScript-Dokumentation auf der Begleit-CD-ROM zu diesem Buch im Ordner *\Infos\Jscript*. Bisher ist es mir aber im WSH nicht gelungen, diese bedingte Übersetzung zu nutzen. Hinweis

Test mit dem Microsoft Script Debugger

Gehen wir einen Schritt weiter. In den vorherigen Abschnitten haben Sie erfahren, wie sich einfache Skripte per Editor und durch Ausgaben von Testanweisungen testen lassen. Diese Vorgehensweise wird aber bei umfangreicheren Skripten recht aufwendig und ist bei manchen Laufzeitfehlern kaum noch einsetzbar. Dann ist guter Rat teuer. Wie läßt sich feststellen, was das Programm macht und welche Werte bestimmte Variable besitzen. Es gibt aber einen trickreichen Ausweg, mit dem Sie sogar kostenlos an einen Debugger gelangen. Von Microsoft wird der Microsoft Script Debugger zum Testen von Skripten angeboten. Dieses Programm wurde zwar ursprünglich zum Testen von Skripten in HTML-Dateien oder in Active Server Pages (ASP-Dateien) entwickelt. Es läßt sich aber mit einigen Tricks auch zum Testen von WSH-Skripten verwenden.

Sie finden den Microsoft Script Debugger auf der Begleit-CD-ROM im Ordner *\Tools\IE4Debug*. Starten Sie die Datei *ie401dbg.exe*. Der Debugger wird anschließend unter Windows installiert. Sie finden das Programm im Ordner *\Programme\Microsoft Script Debugger*. Falls Sie häufiger mit dem Programm arbeiten, sollten Sie ein Verknüpfungssymbol auf dem Desktop einrichten. Hinweis

Vorbereiten des Skripts zum Debuggen

Das Problem beim Testen mit dem Microsoft Script Debugger besteht darin, das Skript zur Ausführung zu bringen. Im Debugger selbst habe ich keine Möglichkeit gefunden, um das Skript zu laden und dann auszuführen. Sie müssen also dafür sorgen, dass das Skript bereits läuft, wenn Sie den Ablauf im Debugger verfolgen möchten. Ohne besondere Vorkehrungen ist das Skript vielfach aber bereits beendet, bevor der Debugger die Kontrolle übernehmen kann.

Hinweis

Dieses Verhalten ist auch zu erklären. Der Debugger wurde als Add-On zum Internet Explorer entwickelt. Ist der Debugger installiert, und laden Sie eine HTML-Datei mit einem Skript, zeigt der Internet Explorer den neuen Befehl *Skriptdebugger* im Menü *Ansicht*. Über diese Befehle können Sie das Fenster des laufenden Debuggers in den Vordergrund holen und das Skript testen.

Um den Debugger dennoch zum Testen eines WSH-Skripts zu nutzen gibt es zwei Möglichkeiten. Nachfolgend erläutere ich erst die Methode, die ich lange Zeit verwendet habe – und die in allen WSH-Versionen funktionieren sollte. Sie benutzen folgenden kleinen Trick: Am Anfang eines Skripts wird ein Aufruf zur Anzeige eines Dialogfelds eingebaut. Dieses Dialogfeld (**Abbildung 2.23**) fordert Sie zum Aktivieren des Debuggers auf.

Abbildung 2.23:
Dialogfeld mit
Aufforderung zum
Starten des
Debuggers



Hinweis

Da der Ablauf des Skripts während der Anzeige des Dialogfelds unterbrochen wird, haben Sie anschließend Gelegenheit zum Aufrufen des Debuggers. Erst wenn Sie das Dialogfeld über die *OK*-Schaltfläche schließen, wird der Ablauf fortgesetzt. Vorher können Sie aber dafür sorgen, dass der Debugger die Kontrolle über den Ablauf erlangt.

Bei einem VBScript-Programm sieht die Anweisung zum Öffnen des Dialogfelds folgendermaßen aus:

```
MsgBox "Debugger starten"
```

Möchten Sie das Dialogfeld in einem JScript-Programm öffnen, können Sie die *Echo*-Methode des *Wscript*-Objekts gemäß folgender Anweisung benutzen:

```
Wscript.Echo ("Debugger starten");
```

Plazieren Sie die betreffende Anweisung an den Anfang des jeweiligen Skripts. Dann wird das Dialogfeld direkt nach dem Start angezeigt und das Skript angehalten. Sie können dann die auf den folgenden Seiten aufgeführten Schritte durchführen.

Hinweis

Die beiden Testdateien *FehlerTest1.vbs* und *FehlerTest1.js* im Ordner *\Beisp\Kap02* auf der Begleit-CD-ROM sind aber bereits mit solchen Anweisungen zum Debuggen vorbereitet.

Trick: So geht's noch einfacher

Die Methode, ein Meldungsfeld am Skriptanfang zu öffnen, um den Ablauf anzuhalten, ist zwar recht pfiffig. Sie müssen aber den Debugger manuell

starten und das Skript zum Testen auswählen (➤ siehe die folgenden Abschnitte). Bei der Entwicklung der ersten Skripte verfolgte ich diesen Weg.

Irgendwann stieß ich dann bei meinen Recherchen auf eine effektivere Möglichkeit. Diese besteht darin, den Quellcode des Skripts mit einem Befehl zum direkten Aufrufen des Debuggers auszustatten. Leider unterscheidet sich dieser Befehl zwischen VBScript und JScript:

- ◆ In VBScript ist der Befehl *stop* im Quellcode einzubauen.
- ◆ In JScript muß die Anweisung *debugger* im Quellcode eingebaut werden.

In beiden Fällen wird der Ablauf des Skripts unterbrochen, sobald der Interpreter den Befehl ausführt. Gleichzeitig startet der Interpreter den Debugger (der natürlich installiert sein muß) und lädt die Codeseite zum Testen.

Sie finden im Ordner *\Beisp\Kap02* der Begleit-CD-ROM die beiden Dateien *FehlerTest2.js* und *FehlerTest2.vbs*. Hinweis

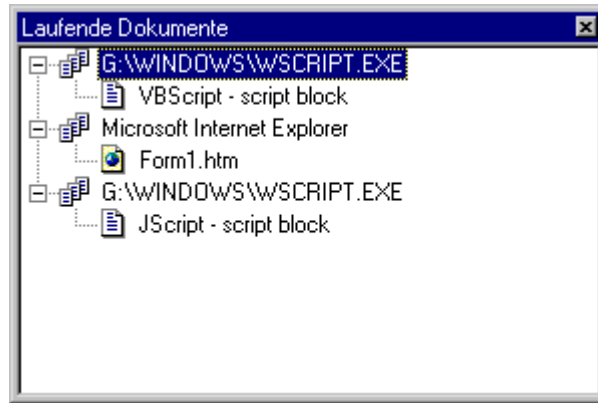
Diese beiden Befehle sind aber sehr gut in den Sprachreferenzen versteckt. Wichtig
Bei VBScript ist beispielsweise anzumerken, dass der WSH den in der Sprachreferenz aufgeführten Befehl *debug.print* nicht unterstützt. Bei JScript ist der Befehl *debugger* als reserviertes Schlüsselwort für zukünftige Erweiterungen reserviert. Den Hinweis auf das Schlüsselwort *debugger* habe ich zufällig in einem anderen Dokument gefunden (Sie finden dieses Dokument in der Datei *JScript32.htm* im Ordner *\Infos\EcmaScript* auf der Begleit-CD-ROM). Es kann daher sein, dass die beiden Befehle in diversen WSH-Versionen nicht funktionieren. Bei meinen Test unter Windows 98 klappte der Aufruf aber ohne Probleme.

Das Skript zum Testen starten

Enthält das Skriptprogramm keine Anweisungsfehler mehr (d. h. es werden keine Übersetzungsfehler gemeldet) und wurde es für die Testausgabe gemäß den obigen Ausführungen vorbereitet, können Sie mit dem Testen im Microsoft Script Debugger beginnen. Zum Ausführen des Skripts reicht ein Doppelklick auf das Symbol der Datei. Findet sich eine Anweisung zum Aufrufen des Debuggers im Quellcode, wird dieser automatisch gestartet und die Seite mit dem Code geladen. Sie können direkt mit dem Test beginnen.

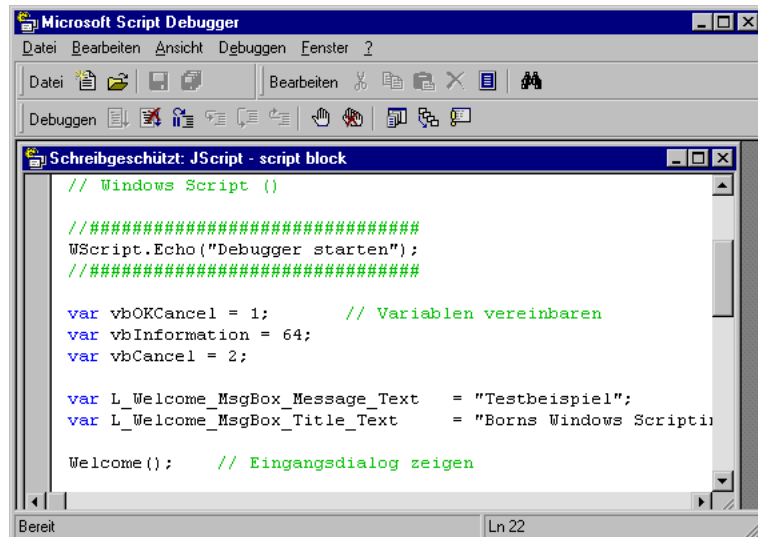
Falls Sie aber die Anweisung zur Anzeige des Meldungsfelds (**Abbildung 2.23**) verwendet haben, hält der WSH das Skript lediglich an und wartet auf das Anklicken der *OK*-Schaltfläche. Sie haben dann die Möglichkeit, den Debugger manuell zu starten und den Code im Debug-Fenster zu laden.

Abbildung 2.24:
Fenster Laufende
Dokumente



1. Starten Sie nun den Microsoft Script Debugger. Dieser öffnet ein leeres Programmfenster. Zum Testen der Skriptanweisungen müssen Sie jetzt auf jeden Fall das laufende Dokument laden.
2. Wählen Sie im Menü *Ansicht* des Script Debuggers den Befehl *Laufende Dokumente*. Oder Sie klicken in der Symbolleiste auf die gleichnamige Schaltfläche (**Abbildung 2.26**). Der Debugger blendet dann das Fenster mit den laufenden Dokumenten ein.
3. In **Abbildung 2.24** sind zur Zeit drei Dokumente aktiv, wobei eines zum Internet Explorer gehört, während zwei Instanzen des Windows Scripting Host ausgeführt werden. Sie müssen nun die Dokumentinstanz per Doppelklick wählen, deren Quellcode im Debugger auszuführen ist.

Abbildung 2.25:
Fenster des Script
Debuggers mit zum
Test geladenem
Skript



Der Script Debugger zeigt dann das Fenster mit dem Quellcode an. Die ausführbaren Anweisungen werden in grüner Schrift angezeigt. Im Gegensatz

zu einem eventuell geöffneten Editierfenster mit dem Quelltext der Skriptdatei lassen sich die Anweisungen im Debug-Fenster jedoch nicht bearbeiten. Sie erkennen dies an dem Vermerk »Schreibgeschützt« in der Titelzeile des Fensters. **Abbildung 2.25** zeigt ein solches Beispiel. In dieser Abbildung enthält das linke Fenster die Anweisungen zum Testen des Skripts. Das rechte Fenster wurde dagegen mit den oben beschriebenen Anweisungen zum Bearbeiten des Quellcodes geöffnet. Jetzt können Sie den Debugger zum Testen des Skripts vorbereiten, indem Sie beispielsweise Haltepunkte setzen.

Der Debugger erlaubt Ihnen auch den Quelltext des Skripts zum Bearbeiten Hinweis im Editierfenster zu laden. Hierzu wählen Sie im Menü *Datei* den Befehl *Öffnen* (oder klicken Sie auf die Schaltfläche *Öffnen*). Im Dialogfeld *Öffnen* ist der Dateityp auf *Alle Dateien (*.*)* zu setzen, um die Skriptdateien anzuzeigen. Dann wählen Sie die Skriptdatei und schließen das Dialogfeld. Der Debugger zeigt den Quellcode in einem eigenen Editierfenster an. Im Gegensatz zum Debug-Fenster können Sie die Anweisungen im Editierfenster jederzeit ändern. Denken Sie aber daran, dass vor dem Ausführen des Tests der geänderte Quellcode gespeichert werden muß. Leider besitzt der Script Debugger keine Funktion zum Zurücksetzen des Skripts im Windows Scripting Host. Sie müssen daher die obigen Schritte erneut ausführen, d. h. die aktuelle Debug-Sitzung abbrechen, das Skript neu starten und das laufende Dokument erneut im Debugger öffnen (einfacher geht es leider nicht).

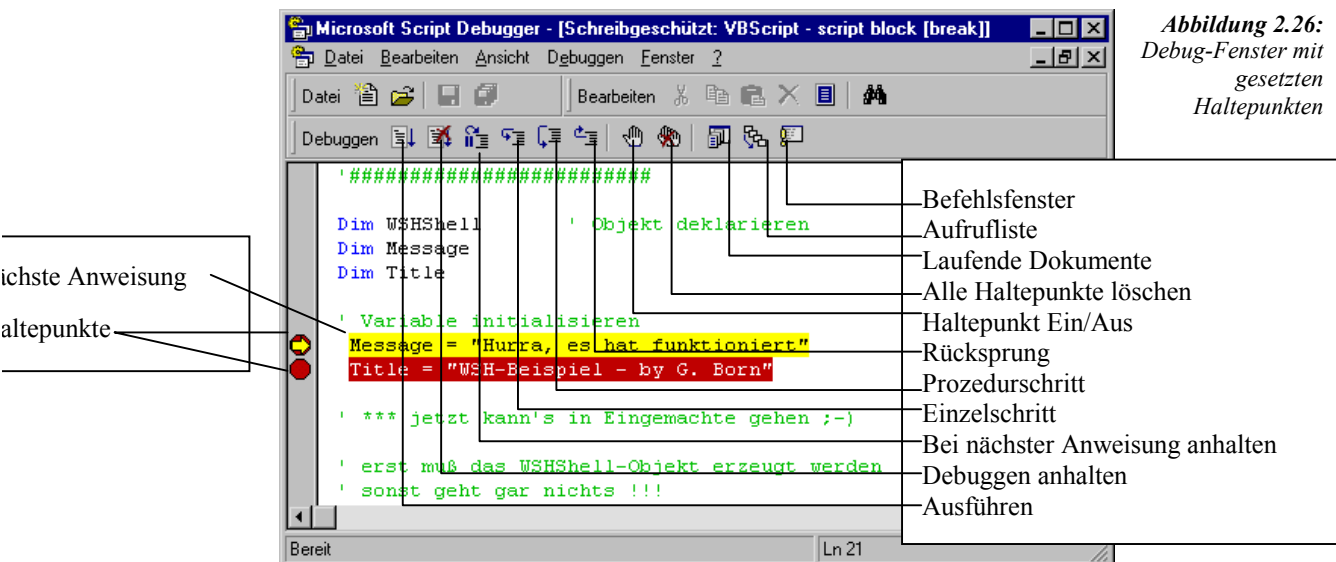


Abbildung 2.26:
Debug-Fenster mit
gesetzten
Haltepunkten

Nach diesen Schritten sind Sie endlich am Ziel. Sie können nun den Ablauf des Skripts im Debugger steuern und verfolgen. Die Schaltflächen der Symbolleiste *Debuggen* rufen die betreffenden Funktionen auf (**Abbildung 2.26**).

1. Klicken Sie jetzt auf die Schaltfläche *Bei nächster Anweisung anhalten*. Dies stellt sicher, dass das Skript nicht in einem Stück ausgeführt wird.
2. Dann wechseln Sie zum geöffneten Dialogfeld *Debugger starten* und schließen dieses über die *OK*-Schaltfläche.

Jetzt besitzt der Debugger die Kontrolle über das Skript. Die als nächstes auszuführende Zeile wird im Debug-Fenster gelb hervorgehoben. Um das Skript auszuführen, können Sie jetzt die Befehle im Menü *Debuggen* oder die Schaltflächen der gleichnamigen Symbolleiste verwenden.

Befehle des Debuggers

Neben dem Befehl *Bei nächster Anweisung anhalten* besitzt der Debugger weitere Befehle, die sich per Menü *Testen* oder über die Schaltflächen der Symbolleiste abrufen lassen (**Abbildung 2.26**):

- ◆ *Einzelschritt*: Wählen Sie diese Schaltfläche oder die Funktionstaste F8, um die nächste Anweisung des Skripts auszuführen.
- ◆ *Prozedurschritt*: Diese Schaltfläche bzw. der Befehl bzw. die Tastenkombination Umschalt+F8 bewirkt, dass ein Prozeduraufruf komplett bis zum Rücksprung in das übergeordnete Modul ausgeführt wird. Liegt kein Prozeduraufruf vor, wirkt der Befehl wie der Einzelschritt-Modus.
- ◆ *Rücksprung*: Dieser Befehl führt alle Anweisungen innerhalb einer Prozedur bis zum Rücksprung zum übergeordneten Modul aus.
- ◆ *Haltepunkte Ein/Aus*: Klicken Sie auf eine ausführbare Anweisung, lassen sich mit diesem Befehl (bzw. mit der Funktionstaste F9) Haltepunkte setzen bzw. gesetzte Haltepunkte aufheben. Haltepunkte im Skript werden am linken Fensterrand mit einem braunen Punkt markiert. Bei der Ausführung des Skripts unterbricht der Debugger den Ablauf beim Erreichen eines solchen Haltepunkts.
- ◆ *Alle Haltepunkte löschen*: Der Befehl löscht alle im Debugger gesetzten Haltepunkte.

Über die Schaltfläche *Befehlsfenster* läßt sich das gleichnamige Fenster öffnen. In diesem Fenster können Sie direkt Befehle eingeben und ausführen lassen. Die Schaltfläche *Aufrufliste* öffnet ein Fenster, in dem die Aufrufliste zur Aktivierung verschiedener Prozeduren angezeigt wird. Diese Liste ist leer, wenn das Skript einen linearen Ablauf ohne Prozeduraufrufe besitzt.

Um das gesamte Skript ablaufen zu lassen, wählen Sie den Befehl *Ausführen*. Der Ablauf wird lediglich durch gesetzte Haltepunkte unterbrochen.

Mit dem Befehl *Debuggen anhalten* läßt sich der Debug-Prozeß beenden. Der Windows Scripting Host zeigt ein Dialogfeld mit dem Hinweis an, dass das Skript abgebrochen wurde. Schließen Sie dann das noch geöffnete Testfenster mit dem angezeigten Quellcode.

Die Anweisungen schrittweise ausführen

Beim Testen ist häufig ein schrittweiser Ablauf des Programmes erforderlich. Durch Anklicken der Schaltfläche *Einzelschritt* oder durch Drücken der Funktionstaste F8 wird die nächste Anweisung als *Einzelschritt* ausgeführt.

Die nächste auszuführende Anweisung wird dabei im Code-Fenster durch einen gelben Pfeil und eine farbliche Markierung hervorgehoben (**Abbildung 2.26**).

Den Ablauf gezielt unterbrechen

Verwenden Sie bereits ausgetestete Prozeduren, ist es im Einzelschrittmodus recht umständlich, wenn jede Anweisung der Prozedur unterbrochen wird. Sie können jedoch die Schaltfläche *Prozedurschritt* oder die Tastenkombination Umschalt+F8 verwenden. Enthält die Anweisung einen Prozeduraufruf, wird die Prozedur ohne Unterbrechung ausgeführt. Das Programm wird dann bei der Anweisung hinter dem Prozeduraufruf unterbrochen. Bei anderen Anweisungen wirkt der Befehl wie die Schaltfläche *Einzelschritt*.

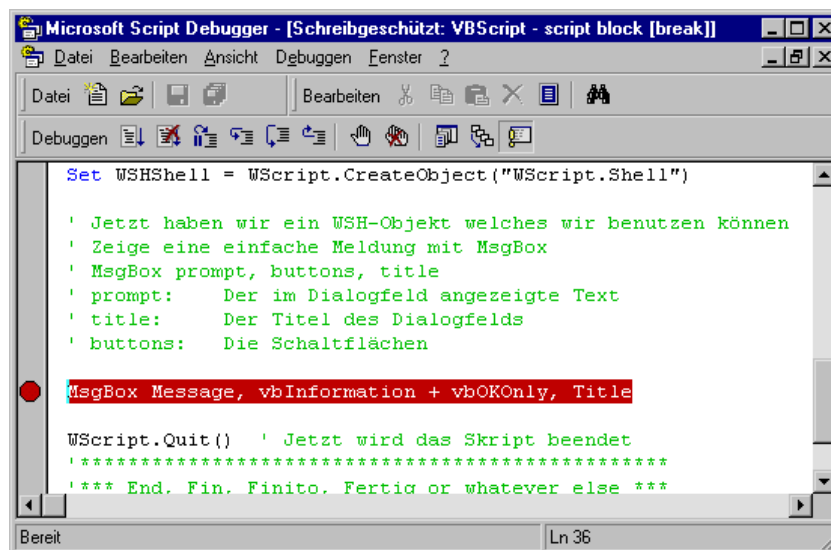


Abbildung 2.27:
Code-Fenster mit
einer Anweisung
zum
Prozeduraufruf

Wurde der Ablauf des Programms innerhalb einer Prozedur unterbrochen, läßt sich die Schaltfläche *Rücksprung* (oder die Tastenkombination

Strg+Umschalt+F8) benutzen. Dann führt das System alle Anweisungen der Prozedur aus und unterbricht den Programmablauf am Prozedurende.

Haltepunkte setzen

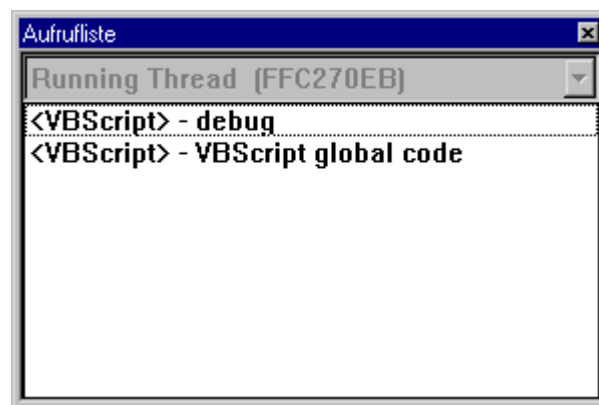
Um die Ausführung des Programmes in einer bestimmten Anweisungszeile zu unterbrechen, wählen Sie die betreffende Anweisungszeile per Mausklick an. Dann setzen Sie einen Haltepunkt über die Schaltfläche *Haltepunkt Ein/Aus* bzw. durch Drücken der Taste F9. Sie können hierbei mehrere Haltepunkte pro Skript setzen. Wenn Sie anschließend den Programmablauf über die Schaltfläche *Ausführen* starten, unterbricht der Debugger den Programmablauf, sobald ein Haltepunkt erreicht wird. Haltepunkte werden im Code-Fenster durch einen braunen Punkt in der *Kennzeichenleiste* und eine farbliche Hinterlegung in der Anweisungszeile markiert (**Abbildung 2.27**).

Um Haltepunkte zu entfernen, klicken Sie erneut auf die betreffende Anweisungszeile und wählen wiederum die Funktion zum Setzen des Haltepunkts. Der Befehl *Alle Haltepunkte löschen* im Menü *Testen* (bzw. die gleichnamige Schaltfläche oder die Tastenkombination Strg+Umschalt+F9) entfernen alle gesetzten Haltepunkte aus dem aktuellen Modul.

Hinweis

Im Gegensatz zu den anderen Entwicklungsumgebungen lassen sich die Haltepunkte im Debugger nicht durch Anklicken der *Kennzeichenleiste* der Zeile direkt setzen oder entfernen.

Abbildung 2.28:
Anzeige der
Aufrufeliste



Anzeige der Aufrufeliste

Die Reihenfolge der Prozeduraufrufe läßt sich über den Befehl *Aufrufeliste* im Menü *Testen* oder über die entsprechende Schaltfläche einblenden. Das System zeigt dann das Fenster aus **Abbildung 2.28** mit den Namen der aufgerufenen Funktionen und Prozeduren an. In diesem Beispiel wurde

beispielsweise gerade die Prozedur *debug* aus dem Hauptprogramm aufgerufen.

Werte anzeigen

Wird ein Programm im Ablauf unterbrochen, können Sie sich auf recht einfache Weise die aktuellen Werte einer Variablen (oder einer Eigenschaft) anzeigen lassen. Öffnen Sie das Fenster *Befehlsfenster*. Bei VBScript geben Sie anschließend ein Fragezeichen gefolgt von dem Sie interessierenden Variablennamen ein. Der Debugger gibt dann den Wert in einer neuen Zeile aus (**Abbildung 2.29**).

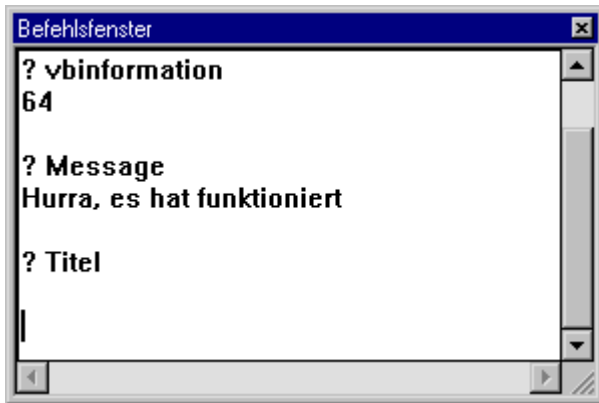


Abbildung 2.29:
Anzeige von
Werten

Bei JScript-Programmen müssen Sie eine Anweisung zur Ausgabe einer Meldung (z. B. *WScript.echo variable*) angeben. Dann führt der Debugger den Befehl aus, das Meldungsfeld mit der angegebenen Variable *variable* wird angezeigt.

Möchten Sie den Wert einer Variablen gezielt setzen, öffnen Sie das Hinweis Direktfenster und geben anschließend die Anweisung mit der Wertzuweisung an die Variable ein. Im *Befehlsfenster* können Sie natürlich auch Programmanweisungen ausprobieren (dies wurde beim Einsatz der *WScript.Echo*-Methode ja bereits genutzt). Die in der Sprachreferenz aufgeführte Möglichkeit in VBScript mit *debug.print* Ausgaben zu tätigen, funktioniert nach meinen Erkenntnissen beim WSH leider nicht.

Mit den obigen Funktionen lassen sich Skripte sehr komfortabel testen. Allerdings kommt der Debugger in seiner Funktionalität nicht an die Debugger der Microsoft-Entwicklungsumgebungen heran. Beachten Sie auch, dass der Microsoft Debugger nicht benutzt werden darf, falls Sie die Entwicklungsumgebung Visual Interdev installiert haben. Diese Entwicklungsumgebung besitzt einen eigenen Debugger, der beim Auftreten eines Fehlers im (HTML- bzw. ASP-) Skript automatisch aufgerufen wird.

Anweisungen, Folgezeilen und Kommentare	74
Variablen und Konstanten	78
Operatoren	88
Kontrollstrukturen	93
Schleifen	96
Prozeduren und Funktionen	99

In diesem Kapitel erhalten Sie eine kurze Einführung in die Sprache VBScript. Wer von Visual Basic oder Visual Basic für Applikationen umsteigt, wird sich mit VBScript sofort zurecht finden.

- ◆ Lernen Sie, wie ein VBScript-Programm aufgebaut ist und wie Sie mit Variablen oder Konstanten umgehen.
- ◆ Erfahren Sie, welche Konstrukte VBScript zur Programmsteuerung bietet.
- ◆ Lesen Sie nach, wie Prozeduren und Funktionen in VBScript gehandhabt werden.
- ◆ Weiterhin finden Sie als VB-Umsteiger Hinweise zu den Restriktionen, die bei VBScript gelten.

Mit diesen Informationen sollten Sie sehr schnell eigene Skripte erstellen können. Wer die Sprache VBScript sehr gut beherrscht, kann dieses Kapitel problemlos übergehen.

Anweisungen, Folgezeilen und Kommentare

VBScript ist die Microsoft-Implementierung von Visual Basic (VB) für den Internet Bereich. Aus dem VB-Funktionsumfang wurden verschiedene Komponenten entfernt, die im Internetbereich weniger relevant sind. Sofern Sie mit VB oder VBA umgehen können, dürfte die Programmierung von VBScript keine größeren Probleme aufwerfen. Nachfolgend finden Sie einige prinzipielle Hinweise zur Gestaltung von VBScript-Programmen.

VBScript-Anweisungen

VBScript-Anweisungen sind im Editor gemäß den jeweils geltenden Syntaxregeln einzugeben. Die nachfolgenden Zeilen enthalten gültige Anweisungen:

```
Wert = 10
Wert = Wert + 10
If Wert > 100 Then Wert = 100
Mwst = 0.1 : Ust = Wert * Mwst
```

Sie können auch mehrere Anweisungen in einer Zeile aufnehmen, sofern Sie diese durch Doppelpunkte : trennen. Dies wurde beispielsweise in der letzten Zeile genutzt. Aus Gründen der Übersichtlichkeit und Programmentransparenz sollten Sie jedoch auf diese Möglichkeit verzichten.

Tip

Bei der Eingabe der Befehle wird in VBScript (im Gegensatz zu JScript) keine Unterscheidung zwischen Groß- und Kleinschreibung getroffen. Dezimalzahlen sind im VBScript-Code mit einem Punkt und nicht mit einem Komma anzugeben.

Fortsetzungszeilen

Bei sehr langen Anweisungszeilen leidet die Übersichtlichkeit erheblich. Die folgende Anweisung:

```
MsgBox "Sie haben schon wieder mal einen falschen Namen eingegeben",
vbOkOnly, "Eingabefehler"
```

zeigt einen Text in einem Meldungsfeld an. Die Anweisung ist zu lang für eine Zeile. Enthält Ihr Programm viele solcher Anweisungen, leidet die Handhabbarkeit und Lesbarkeit des Listings erheblich. VBScript bietet Ihnen daher die Möglichkeit, lange Anweisungen auf mehrere Zeilen aufzuteilen.

Sie müssen am Ende einer Zeile jeweils ein Leerzeichen gefolgt von einem Unterstrich _ einfügen.

```
MsgBox "Sie haben schon wieder mal einen falschen Namen eingegeben", _  
      vbOkOnly, _  
      "Eingabefehler"
```

Der Unterstrich am Zeilenende signalisiert dem WSH-Interpreter, dass die Anweisung in der folgenden Zeile fortgesetzt wird. Dieses Fortsetzungszeichen darf durchaus in mehreren Zeilen auftreten. Lediglich die letzte Zeile der gesamten Anweisung enthält kein solches Zeichen.

Beachten Sie, dass hinter einer Fortsetzungszeile kein Kommentar folgen darf. Bei Fortsetzungszeichen in Zeichenketten müssen diese mit einem Anführungszeichen abgeschlossen und mit einem &- bzw. +-Zeichen verknüpft werden. Das Fortsetzungszeichen muß weiterhin durch mindestens ein Leerzeichen von dem letzten Zeichen der Anweisung getrennt werden. Hinweis

Kommentare

Möchten Sie, dass VBScript eine Zeile oder einen Teil einer Zeile nicht als Anweisung interpretiert? Dann müssen Sie diese Anweisung als Kommentar markieren. Kommentare werden in VBScript durch ein Hochkomma (Zeichen ') eingeleitet. Der VBScript-Interpreter überliest dann den gesamten folgenden Text bis zum Zeilenende. Die beiden nachfolgenden Zeilen enthalten jeweils Kommentare:

```
' Dies ist eine komplette Kommentarzeile  
Wert = Wert * Mwst      ' ermittle Steuersatz
```

In der zweiten Zeile beginnt der Kommentar erst hinter der Anweisung, d. h. VBScript führt die am Zeilenanfang befindliche Anweisung noch aus. Dies erlaubt Ihnen, Kommentare am Zeilenende hinter einzelnen Anweisungen einzufügen.

Hinweise zur Struktur eines VBScript-Programms

Ein VBScript-Programm kann aus Kommentaren und den bereits oben erwähnten Anweisungen bestehen. Die in diesem Buch benutzten Skriptdateien besitzen jedoch eine besondere Struktur, die in dem folgenden Listing gezeigt wird.

```
' *****  
' File:   WSHTest.vbs (VBScript WSH)  
' Autor: (c) G. Born  
'  
' Demonstriert den Einsatz des WSH. Es wird ein einfaches
```

Listing 3.1:
*VBScript-
Programm*

```

' Dialogfeld geöffnet.
'*****

Dim WSHShell      ' Objekt deklarieren
Dim Message
Dim Title

' Variable initialisieren
Message = "Hurra, es hat funktioniert"
Title = "WSH-Beispiel - by G. Born"

' *** Jetzt kann's ins Eingemachte gehen ;- )

' Erst muß das WSHShell-Objekt erzeugt werden,
' sonst geht (meistens) gar nichts !!!
Set WSHShell = WScript.CreateObject("WScript.Shell")

' Jetzt haben wir ein WSH-Objekt, welches wir benutzen können
' Zeige eine einfache Meldung mit MsgBox an
' MsgBox prompt, buttons, title
' prompt:      Der im Dialogfeld angezeigte Text
' title:       Der Titel des Dialogfelds
' buttons:     Die Schaltflächen

MsgBox Message, _
    vbInformation + vbOKOnly, _
    Title

WScript.Quit() ' Jetzt wird das Skript beendet
'*****
'*** Ende, WSH powered by Günter Born ***
'*****

```

Das obige Skript besitzt eine ganze Reihe von Kommentaren. Um bestimmte Funktionen zu realisieren, benötigen Sie diese Kommentarzeilen zwar nicht. Es entspricht aber einer guten Programmierpraxis, jedes Skript am Programmanfang mit einem sogenannten Kommentarkopf zu versehen. Als erstes ist der Name der Skriptdatei sowie die Sprachvariante einzutragen. Vielleicht sehen Sie dies als überflüssig an, Sie wissen ja, dass Sie gerade ein Skript mit dem Namen xy erstellen. Nun, der obige Ansatz wurde durch die Erfahrungen des Autors bei der Realisierung großer, teilweise international geführter Softwareprojekte geprägt. Nehmen wir einmal an, Sie erhalten einen Stapel Ausdrucke mit dem Quellcode verschiedener Programme. Sofern der Editor beim Drucken den Dateinamen nicht mit ausgibt, beginnt die

Raterei, in welcher Datei der Code wohl zu suchen ist. Und die Angabe der Sprachversion kommt auch aus dieser Ecke. Gelangt ein fremdes Skript in Ihre Hände, sind Sie sicherlich dankbar, wenn der Autor im Programmkopf vermerkt hat, für welche Sprachvariante der Code geschrieben wurde. Es könnte sich ja auch um VB, VBS oder VBScript für den Internet Explorer handeln. Und diese Skripte laufen vermutlich nicht im WSH.

Äußerst hilfreich ist es auch, wenn sowohl der Autor vermerkt ist als auch der Zweck des Skripts angegeben wird. Gerade bei der Entwicklung umfangreicher Skripte wird die Nachwelt solche Angaben dankbar zur Kenntnis nehmen.

Dass der Quellcode bei den Skripten des Autors mit einem abschließenden Kommentar versehen wird, entspricht einer Marotte, die aber einen tieferen Sinn hat. VBScript erfordert nicht zwingend eine Ende-Anweisung am Programmende. In einem der früheren Projekte (ich glaube, es war seinerzeit dBASE II) arbeitete ich mit einem Editor, der bei bestimmten Befehlen einfach den Rest des Programmes abschnitt. Den anschließend gespeicherten Quellcodes fehlten einige Zeilen. Der Fehler fiel leider lange Zeit nicht auf. Erst als jemand die Software ausgiebig testete, stellten sich Fehlfunktionen ein. Ein erster Blick zeigte keine Fehler. Erst die Analyse ergab, dass einige Zeilen am Programmende fehlten. Da die Applikation aus mehreren hundert Dateien bestand, war es ein riesiger Aufwand, die fehlerhaft gespeicherten Dateien zu identifizieren und anhand der Listingausdrucke zu korrigieren. Seit dieser Zeit signiere ich das Ende aller Programmdateien (insbesondere wenn diese interpretiert werden) mit einem abschließenden Kommentar. Eigentlich reicht der Kommentar:

```
' *** Ende
```

für diesen Zweck. Aber Hand aufs Herz, sieht der folgende Kommentar nicht besser aus?

```
' *****  
' *** End, Fin, Finito, Fertig or whatever else ***  
' *** WSH powered by Günter Born ***  
' *****
```

Inwieweit Sie diese Empfehlungen übernehmen, bleibt Ihnen überlassen. Persönlich pflege ich meine Marotten und bin spätestens nach einigen Wochen dankbar, wenn die Ausdrucke der Programme mit Kommentarkopf und weiteren Erläuterungen versehen sind. In den Beispielen dieses Buches wird von der Kommentierung ausführlich Gebrauch gemacht.

Sie finden das obige Beispielprogramm *WSHTest.vbs* im Ordner *\Beisp\Kap03* auf der Begleit-CD-ROM. VBScript-Programme lassen sich auch in HTML-Dokumente einbinden und im Internet Explorer 4.0 ausführen. Es gibt aber geringfügige Abweichungen im Hinblick auf die zulässigen Funktionen zwischen VBScript-Programmen im WSH und den VBScript-Anweisungen in einem HTML-Dokument. So ist die komplette

Hinweis

Ereignisverwaltung, die VBScript-Skripte in HTML-Dokumenten unterstützen müssen, im WSH-Skript weitgehend ohne Bedeutung. Hinweise zum Einbinden von VBScript-Anweisungen in HTML-Dokumente sowie Beispiele finden Sie in dem im Literaturverzeichnis unter /6/ aufgeführten Titel.

Variablen und Konstanten

In VBScript können Sie wie in anderen Sprachen Variablen und Konstanten zum Speichern von Werten verwenden.

Konstanten

In den VBScript-Anweisungen lassen sich feste Werte direkt als Konstanten angeben. Die folgende Anweisung verwendet eine solche Konstante:

```
Betrag = Preis * Menge + 100.0
```

Der Wert 100.0 ist eine Konstante, die in die Berechnung eingeht. Häufig besteht jedoch der Wunsch, eine Konstante zentral zu vereinbaren und als Bezeichner im VBScript-Programm zu verwenden. Hierzu muß die Konstante im Deklarationsteil des Moduls explizit definiert werden.

```
Const Zuschlag = 100.0
```

Mit dem Schlüsselwort *Const* wird eine Konstantendeklaration eingeleitet. Daran schließt sich der Name der Konstanten an. Weiterhin wird in der Anweisungszeile der Wert der Konstanten gesetzt. Die Konstante läßt sich anschließend unter ihrem Namen im VBScript-Programm benutzen.

```
Betrag = Preis * Menge + Zuschlag
```


Der Vorteil besteht darin, dass Sie den Wert der Konstanten zentral im Deklarationsteil ändern können. In einer Zeile lassen sich auch mehrere Konstanten definieren:

```
Const MwSt = 0.16, Zuschlag = 10
```

VBScript weist der Konstanten neben dem Wert den betreffenden Datentyp zu. VBScript verwendet allerdings (abweichend von VB bzw. VBA) nur den Datentyp *Variant* (↗ siehe auch die nachfolgenden Ausführungen). Benannte Konstante sind standardmäßig als *public* (öffentlich) deklariert. Verwenden Sie eine benannte Konstante in einer Prozedur, besitzt diese eine lokale Gültigkeit. Sie können die Gültigkeit aber durch Schlüsselwörter *Public* und *Private* angeben (↗ siehe auch die folgenden Seiten).

Tip

Numerische Konstanten werden üblicherweise im Dezimalsystem angegeben (z. B. 10.14). Sie können aber auch Konstanten im Hexadezimalsystem in der

Form *&Hxxx* verwenden, wobei *xxx* für eine Hexadezimalzahl steht. Der Wert *&H0C* entspricht beispielsweise der Zahl 12 (siehe  Abschnitt »Vergleichs-Operatoren«).

Vordefinierte Konstanten


VBScript kennt eine Reihe vordefinierter Konstanten (z. B. *vbOkOnly* etc.), die Sie direkt angeben können. Ich hatte ja bereits in Kapitel 2 das Beispiel beim Aufruf der *MsgBox*-Funktion gebracht. Die Anweisung:

```
MsgBox "Hallo", 0 + 64, "Test"
```

ist sicherlich wesentlich kryptischer, als wenn Sie mit den in VBScript vordefinierten Konstanten arbeiten. In diesem Fall sieht die Anweisung beispielsweise folgendermaßen aus:

```
MsgBox "Hallo", vbOkOnly + vbInformation, "Test"
```

Die einzige Schwierigkeit besteht darin, dass Sie beim Schreiben des Codes die Namen der jeweiligen Konstanten kennen sollten. Hier hilft Ihnen aber der nachfolgende Tip weiter.

Einmal finden Sie auf der Begleit-CD-ROM im Ordner *\Infos\VBScript* die VBScript-Sprachreferenz, in der auch die jeweiligen Konstanten aufgeführt sind. Alternativ können Sie die VB-Entwicklungsumgebungen (z. B. die VB 5 Control Creation Edition) benutzen, um die Konstantennamen direkt im Code-Fenster (siehe  Kapitel 2) abzurufen. Eine Liste der Konstanten erhalten Sie auch, indem Sie in der VB 5 CCE (oder in der VBA-Entwicklungsumgebung von Word 97) den Objektkatalog öffnen, die Bibliothek VBA wählen und dann auf die Klasse *Constants* (bzw. *Konstanten*) klicken. Allerdings sollten Sie in diesem Fall bedenken, dass VBScript nicht alle (VBA- bzw. anwendungsspezifischen) Konstanten unterstützt. Für einen schnellen Zugriff z. B. auf Konstanten der *MsgBox*-Funktion etc. bevorzuge ich diese Variante gegenüber dem Nachschlagen in der Sprachreferenz. Tip

Variablen

Variablen sind Platzhalter, die im Programm in einzelnen Anweisungen auftreten können. Dabei kann der Wert der Variablen (im Gegensatz zu Konstanten) im Programm verändert werden. Eine Variable kann direkt im Programm auftreten:

```
Preis = 45      ' Preis festlegen  
MwSt = 17
```

Sobald der Name einer Variablen erstmals im Programm vorkommt, legt *VBScript* die zugehörige Variable an. Der Datentyp für eine VBScript-

Variable wird dabei immer auf *Variant* gesetzt (d. h. die Variable kann verschiedene Werte wie Zahlen, Texte etc. aufnehmen).

Hinweise zu VBScript-Datentypen

Abweichend von Visual Basic oder VBA kennt VBScript nur den Datentyp *Variant*. Dies ist ein spezieller Datentyp, der verschiedene Arten von Informationen aufnehmen kann. Das Format der in der Variablen gespeicherten Daten richten sich dabei nach dem Kontext des zugewiesenen Wertes. Weisen Sie der Variablen eine Zahl zu, wird ein numerischer Wert gespeichert. Erhält die Variable einen Datumswert zugewiesen, speichert der Interpreter diesen im Datumsformat in der Variablen. Ähnliches gilt für Zeichenketten, die als Text zugewiesen werden.

Hinweis

Da *Variant* der einzige in VBScript unterstützte Datentyp ist, liefern Funktionsaufrufe ebenfalls *Variant*-Werte zurück. Dies kann zu einigen Problemen beim Aufruf von API-Funktionen führen.

Bei Operationen auf Variablen versucht VBScript die am logischsten erscheinende Operation durchzuführen. Addieren Sie den Inhalt zweier Variablen, die numerische Werte aufweisen, wird auch das Ergebnis numerisch. Der Befehl:

```
Summe = Preis + 15.0
```

liefert ein numerisches Ergebnis in der Variablen *Summe* ab. Die folgende Anweisung:

```
Result = "Wert " + Summe
```

birgt jedoch ein Problem: Der erste Teil des Ausdrucks stellt eine Zeichenkette dar, während die Variable *Summe* einen numerischen Wert enthält. Der VBScript-Interpreter kann dann keine Zuweisung vornehmen, sondern liefert eine Fehlermeldung (**Abbildung 3.1**). In diesem Fall müssen Sie eine explizite Konvertierung des Subtyps (➦ siehe den folgenden Absatz) vornehmen.

```
Result = "Wert " + CStr(summe)
```

In diesem Fall übernimmt die VBScript-Funktion *CStr()* die Aufgabe, den numerischen Wert von *summe* in einen *Variant*-Datentyp vom Format Zeichenkette umzuwandeln. Das Ergebnis lässt sich dann mit der zweiten Zeichenkette *"Wert "* zu einer Gesamtzeichenkette verknüpfen (was auch als Konkatination bezeichnet wird).

Sie können auch Zahlen als Strings behandeln, falls diese als Konstante vorliegen. Hierzu müssen Sie die Konstante in Anführungszeichen setzen. Die Anweisung:

```
Text = "15" + "30"
```

Wird beispielsweise nicht als Wert 45 sondern als "1530" in *Text* gespeichert!
VBScript bietet weiterhin verschiedene Funktionen zur Typkonvertierung an.

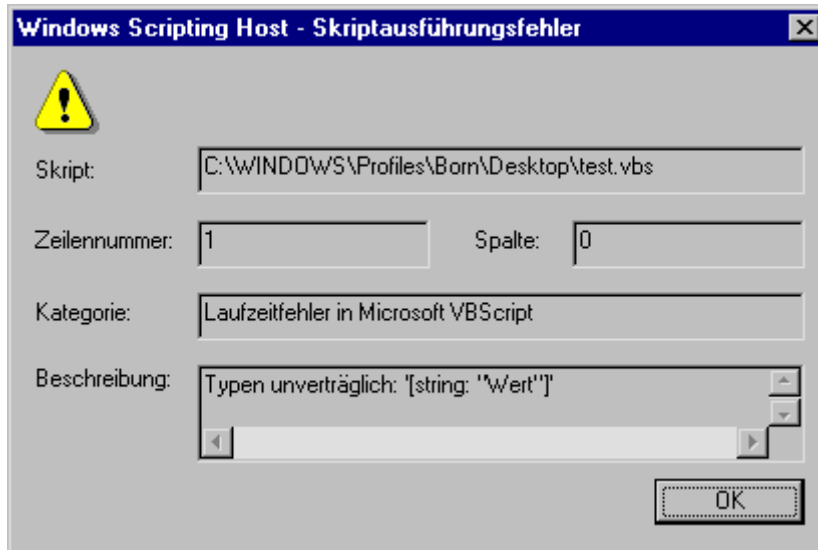


Abbildung 3.1:
Fehlerabbruch
wegen
Typenunverträglichkeiten

Variant-Subtypen

Der VBScript-Interpreter kann im *Variant*-Datentyp zusätzliche Subtypen unterscheiden. Sie haben bereits im vorherigen Abschnitt gelernt, dass der Interpreter den Inhalt einer Variablen nach numerischen Werten oder Zeichenketten unterscheidet. Aber es gibt noch mehr Varianten. Sie können beispielsweise ein Datum oder eine Uhrzeitangabe in eine Variable speichern. Der Wert ist zwar numerisch, benötigt aber ein besonderes Format. Andere Datentypen können logische Werte wie *wahr* oder *falsch* annehmen. Oder es werden Ganzzahlen bzw. Kommazahlen verwendet.

Letztendlich liegt zwar immer der gleiche Datentyp *Variant* vor. Sie können sich dies aber so vorstellen, dass innerhalb der Variablen ein bestimmtes »Format« zur Speicherung des eigentlichen Werts verwendet wird. Und dieses Format sieht bei einer Zeichenkette halt anders als bei einem numerischen Wert oder bei einem Datumswert aus. Hinweis

Die Unterscheidung erfolgt über die Subtypen des *Variant*-Datentyps. Vom VBScript-Interpreter werden intern die in der folgenden Tabelle aufgeführten *Variant*-Subtypen unterschieden.

Subtyp	Bemerkung
Empty	Uninitialisierte <i>Variant</i> -Variable. Der Wert ist 0 für numerische Variablen oder es wird eine leere Zeichenkette ("") für Zeichenkettenvariablen benutzt.

Tabelle 3.1:
Datentypen für
Variant

Null	Die <i>Variant</i> -Variable enthält unbeabsichtigter Weise keine gültigen Daten.
Boolean	Logische Variable, die die Werte <i>True</i> oder <i>False</i> (bzw. <i>Wahr</i> oder <i>Falsch</i>) annehmen kann.
Byte	Enthält einen Integer-Wert im Bereich 0 bis 255.
Integer	Enthält einen Integer-Wert im Bereich –32.768 bis 32.767.
Currency	Währungswert im Bereich –922.337.203.685.477,5808 bis 922.337.203.685.477,5807.
Long	Enthält einen Integer-Wert im Bereich –2.147.483.648 bis 2.147.483.647.
Single	Enthält einen einfach genauen Gleitkommawert im Bereich –3,402823E38 bis –1,401298E–45 für negative Zahlen und 1,401298E–45 bis 3,402823E38 für positive Zahlen.
Double	Enthält einen Gleitkommawert doppelter Genauigkeit im Intervall –1,79769313486232E308 bis –4,94065645841247E–324 und 4,94065645841247E–324 bis 1,79769313486232E308.
Date (Time)	Enthält eine Zahl, die ein Datum oder eine Zeit ausgehend vom 1. Januar 100 bis zum 31. Dezember 9999 repräsentiert.
String	Enthält eine Zeichenkette variabler Länge (bis zu 2 Milliarden Zeichen).
Object	Enthält ein Objekt.
Error	Enthält eine Fehlernummer.

Hinweis

Im Gegensatz zu VB oder VBA können Sie nicht explizit festlegen, welchen Datentyp die Variable besitzt. Der VBScript-Interpreter nimmt automatisch die Klassifizierung in die Untertypen vor. Sie können aber die VBScript-Konvertierfunktionen *Asc*, *CBool*, *CByte*, *CCur*, *CDate*, *Cdbl*, *Chr*, *CInt*, *CLng*, *CSng*, *CStr*, *Hex* und *Oct* verwenden. Weiterhin lässt sich über die *VarType*-Funktion der Subtyp abfragen.

Variablendeklaration mit *Option Explicit* erzwingen

Die implizite Vereinbarung von Variablen birgt eine große Gefahr: Sobald Sie sich vertippen, legt der VBScript-Interpreter eine neue Variable an. Dies führt zu Fehlern im Programm. Stellen Sie sich vor, Sie haben versehentlich in einer Anweisung *Pris* statt *Preis* eingetippt.

MwSt = 16.0


```

Preis = 0
Netto = 115.00
Pris = Preis + Netto * (1.0 + MwSt/100)
MsgBox Preis, vbOkOnly, "Preis"

```

Das obige Programm wird nicht funktionieren. Eigentlich sollte in der Variablen *Preis* der Bruttopreis hinterlegt und anschließend in einem Meldungsfeld ausgegeben werden. Der Tippfehler in der Zuweisung führt jedoch dazu, dass die neue Variable *Pris* vom Interpreter angelegt wird. Das Meldungsfeld gibt immer den Wert 0 aus. Solche Fehler sind schnell passiert, aber sehr schwer zu finden. Den obigen Fehler können Sie noch ohne Aufwand finden, da das Programm sehr trivial und recht kurz ist. Aber bei umfangreicheren Skripten sieht dies anders aus. Fast jeder Basic-Programmierer kennt die Fälle, wo solche Fehler stundenlang gesucht wurden.

Bei JScript tritt dieses Dilemma nicht auf, da alle Variablen zwangsweise deklariert werden müssen. Glücklicherweise hilft Ihnen VBScript ebenfalls aus diesem Dilemma. Sie können erreichen, dass Tippfehler in einer Variablen von *VBScript* erkannt werden, indem Sie in der ersten Zeile des Skripts die Anweisung *Option Explicit* einfügen. Dann müssen alle Variablen explizit im Modulkopf oder im Kopf der Prozedur mit *Dim* deklariert werden (siehe unten). Vergessen Sie dies bei einer Variablen, meldet der WSH beim Übersetzen des Skripts alle nicht deklarierten Variablen als Fehler. Auf diese Weise werden auch falsch geschriebene Variablenamen gefunden.

Sie können dies testen, indem Sie die auf der Begleit-CD-ROM im Ordner *\Beisp\Kap03* enthaltene Datei *FehlerTest2.vbs* aufrufen.

```

Option Explicit
'*****

' File:   FehlerTest2.vbs   (VBScript WSH)
' Autor:  (c) G. Born
'
' Demonstriert den Einsatz des Debuggers zum Testen
' eines WSH-Skripts sowie die Option Explicit-Wirkung.
'*****

'#####
stop ' Debugger aufrufen
'#####

Dim WSHShell      ' Objekt deklarieren
Dim Message
Dim Title

' Variable initialisieren

```

Listing 3.2:
FehlerTest2.vbs

```

Message = "Hurra, es hat funktioniert"
Title = "WSH-Beispiel - by G. Born"
Titel = "Hallo"
' *** Jetzt kann's ins Eingemachte gehen ;-))

' Erst muß das WSHShell-Objekt erzeugt werden,
' sonst geht gar nichts !!!
Set WSHShell = WScript.CreateObject("WScript.Shell")

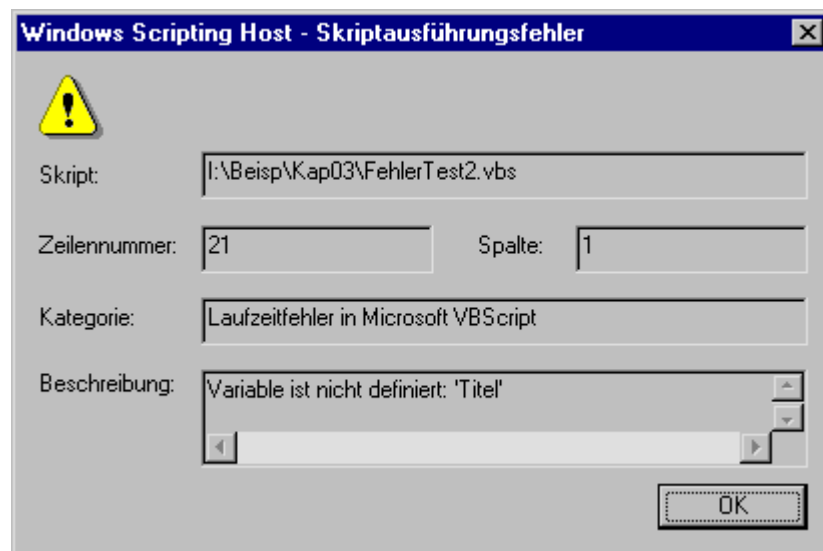
' Jetzt haben wir ein WSH-Objekt, welches wir benutzen können
' Zeige eine einfache Meldung mit MsgBox
' MsgBox prompt, buttons, title
' prompt:    Der im Dialogfeld angezeigte Text
' title:     Der Titel des Dialogfelds
' buttons:   Die Schaltflächen

MsgBox Message, vbInformation + vbOKOnly, Title

WScript.Quit() ' Jetzt wird das Skript beendet
'*****
'*** End, Fin, Finito, Fertig or whatever else ***
'*** WSH powered by Günter Born ***
'*****

```

Abbildung 3.2:
Meldung einer
nicht definierten
Variablen im WSH



In diesem Skript ist eine nicht deklarierte Variable *Titel* enthalten. Starten Sie das Skript, aktiviert dieses zuerst den Debugger. Anschließend können Sie

das Skript schrittweise ausführen. Sobald Sie zur Zuweisung an die Variable *Titel* kommen, merkt der WSH-Interpreter, dass die Variable noch nicht deklariert ist. Dann löst er den in **Abbildung 3.2** gezeigten Fehlerdialog aus. Programmfehler durch nicht deklarierte Variable fallen also sofort auf.

Auch wenn die vielen Fehlermeldungen beim ersten Start eines Skripts (oder bei der nachträglichen Einführung der *Option Explicit*-Anweisung) stören, verwenden Sie die Option. Dies erspart Ihnen nachträglich viel Zeit bei der Fehlersuche. Hinweis

Einsatz der Dim-Anweisung

In VBScript-Programmen stoßen Sie früher oder später auf *Dim*-Anweisungen. Dieses Schlüsselwort wurde beispielsweise bereits in dem vorhergehenden Abschnitt erwähnt. Das *Dim*-Schlüsselwort erlaubt zweierlei:

- ◆ Die explizite Definition von Variablen (auf Prozedur-, Funktions- oder Skriptebene).
- ◆ Die Definition von Feldvariablen (Arrays).

Sofern Sie die Anweisung *Option Explicit* verwenden, müssen die Variablen vor deren erstem Auftreten mit *Dim* explizit deklarieren. Dies kann beispielsweise im Programmkopf oder am Anfang einer Prozedur/Funktion (ähnlich wie eine Konstante) erfolgen:

```
Dim text
Dim x
Dim Preis, MwSt
```

Weiterhin beeinflussen Sie mit der *Dim*-Anweisung die Gültigkeit der Variablen. Falls Sie eine Variable innerhalb einer Funktion/Prozedur deklarieren, beschränkt sich deren Gültigkeit auf diesen Bereich. Variablen, die im Kopf des Skripts mit *Dim* deklariert werden, gelten dagegen in allen Prozeduren des Skripts.

Eine Variable wird bei der Definition automatisch mit einem Wert initialisiert. Bei numerischen Variablen ist dies der Wert 0, während bei Zeichenketten ein Leerstring ("") hinterlegt wird. Hinweis

Public und Private in der Variablendeklaration

Es wurde bereits im vorherigen Abschnitt erwähnt: Eine im Skript mit *Dim* global vereinbarte Variable ist global auch in allen Prozeduren gültig. Eine in einer Prozedur oder Funktion mit *Dim* vereinbarte Variable ist dagegen nur innerhalb der Prozedur gültig. Eigentlich wäre damit bereits alles geklärt. Die VBScript-Sprachreferenz kennt aber noch die beiden Schlüsselwörter *Public* und *Private*, um die Gültigkeit von Variablen explizit zu begrenzen.

Public Test
Public Preis, MwSt

Das Schlüsselwort *Private* bewirkt, dass die Variable innerhalb des Skripts bekannt ist. Verwenden Sie dagegen *Public*, ist die Variable in allen Skripten und Prozeduren bekannt.

Wichtig

Die Anweisungen *Public xWert* oder *Private xWert* sind innerhalb einer Prozedur/Funktion nicht zulässig und führen zu einem Übersetzungsfehler. Arbeiten Sie mit der *Option Explicit*-Anweisung müssen Sie die *Dim*-Anweisung zur Deklaration der Variablen verwenden. Sie können dies testen, indem Sie die im Ordner *\Beisp\Kap03* enthaltene Datei *WSHPublic.vbs* aufrufen. Diese Datei löst beim Übersetzen einen Fehler aus.

Hinweis

Vielleicht fragen Sie sich nun, was für einen Sinn *Private* und *Public* machen? Eigentlich reicht ja die *Dim*-Anweisung aus, um die Gültigkeit zwischen Prozeduren/Funktionen und im Skript eindeutig festzulegen. VBScript wurde aber ursprünglich für HTML-Seiten entwickelt. Und ein solches Dokument kann im HTML-Code mehrere mit dem `<SCRIPT>`-Tag vereinbarte Skripte enthalten. Mit dem Schlüsselwort *Public* erreichen Sie, dass eine Variable auch außerhalb des lokalen Skripts in anderen Skripten gültig ist. Wir werden dies in späteren Kapiteln noch nutzen.

Felder mit Dim deklarieren

Weiterhin können Sie Felder in der Variablendeklaration vereinbaren. Hierzu wird die Zahl der Feldelemente in der Deklaration angegeben.

```
Dim wert (10)
wert (1) = 11
```

Die erste Zeile definiert ein Feld mit 11 Elementen (die Untergrenze des Feldes wird immer mit dem Index 0 belegt). Das erste Element lässt sich mit *wert(0)* ansprechen. Die zweite Anweisung in obigem Beispiel setzt folglich den zweiten Feldwert auf 11. Bei der *Dim*-Anweisung können Sie dabei mehrdimensionale Felder vereinbaren. Die Anweisung:

```
Dim Wert (10,10)
```

vereinbart ein Feld mit zwei Dimensionen. Mit *Wert (0,0)* greifen Sie auf den ersten Feldwert in Zeile 0 und Spalte 0 zurück. VBScript lässt bis zu 60 Dimensionen in der Felddeklaration zu. Die Untergrenze für den Feldindex liegt immer bei 0, d. h. Sie können in der Felddeklaration immer nur die Obergrenze für den Feldindex vereinbaren.

Tip

Sie können die *Dim*-Anweisung mit leeren Klammern einsetzen (z. B. *Dim Wert ()*), um dynamische Felder zu vereinbaren. Sie können dann mittels der *ReDim*-Anweisung die Dimension des Feldes innerhalb einer Prozedur neu definieren. Wurde ein Feld dagegen in der *Dim*-Anweisung mit festen

Dimensionen vereinbart, löst die Redefinition der Felddimensionen einen Laufzeitfehler aus.

Variablennamen

Die Namen für Variable dürfen bis zu 255 Zeichen umfassen. Sie können den Namen weitgehend frei wählen, sofern dieser den nachfolgenden Kriterien genügt:

- ♦ Der Variablenname muß immer mit einem Buchstaben beginnen (*Test* ist ein gültiger Name, während *123* ungültig ist).
- ♦ Innerhalb des Namens sind Leerzeichen, Punkte, Kommas und einige Sonderzeichen (z. B. !, -, +) unzulässig.

Sie dürfen die deutschen Umlaute im Variablennamen verwenden. Beachten Sie aber, dass die Namen von Schlüsselwörtern wie *Sub*, *If*, *End*, *Dim* etc. nicht für Variablennamen benutzt werden dürfen.

Es empfiehlt sich, sprechende Variablennamen zu verwenden, die im Hinblick auf den Tippaufwand nicht länger als 10 bis 15 Zeichen sind.

Operatoren

In VBScript werden verschiedene Typen von Operatoren (arithmetische Operatoren für mathematische Berechnungen, logische Operatoren für logische Vergleiche, Vergleichsoperatoren für sonstige Vergleiche und Verkettungsoperatoren zur Verkettung von Zeichenfolgen) unterstützt. Nachfolgend finden Sie eine kurze Übersicht über die jeweiligen Operatoren.

Arithmetische Operatoren

Die nachfolgende Tabelle enthält eine Aufstellung der in VBScript verfügbaren arithmetischen Operatoren.

Operator	Erklärung
\wedge	Potenzieren ($x = y^{\text{Exponent}}$)
+	Addition ($x = a + b$)
-	Subtraktion oder negatives Vorzeichen ($x = -10$ oder $x = a - 100$)

Tabelle 3.2:
*Arithmetische
Operatoren*

*	Multiplikation ($x = b * 30$)
/	Gleitkomma-Division ($x = a / b$)
\	Integer-Division ($x = a \setminus b$)
Mod	Modulo, Rest aus einer Division ($x = a \text{ Mod } b$)

Hinweis

Beachten Sie, dass sich der `+`-Operator auch zur Verknüpfung von Zeichenketten einsetzen läßt (z. B. `Name = "Müll" + "erin"`). Alternativ kann aber der `&`-Operator zur Verkettung von Zeichenketten verwendet werden. Enthält ein Operand den Wert *Null* oder *Empty*, ist das Ergebnis der Operation ebenfalls *Null* (ungültig) oder *Empty* (leer). Weitere Hinweise entnehmen Sie der VBScript-Hilfe.

Zuweisung von Objektreferenzen mit Set

Der *Set*-Operator besitzt in VBScript noch eine besondere Bedeutung. Zum Zugriff auf Objekte benötigen Sie häufig eine Referenz auf dieses Objekt. Diese Referenzen lassen sich mit dem *Set*-Operator realisieren. Im Vorgriff auf spätere Kapitel möchte ich hier kurz einen solchen Befehl zeigen. Mit der Anweisung:

```
Set objAdr = WScript.Arguments
```

erzeugen Sie eine Objektvariable *objAdr*. Die Variable besitzt in VBScript zwar immer den Datentyp *Variant*, aber der Subtyp wird hier auf Objekt gesetzt. Hier wird konkret auf die *Arguments*-Eigenschaft des *WScript*-Objekts zugegriffen. Die *Arguments*-Eigenschaft liefert ein Objekt, welches den Zugriff auf die Aufrufparameter des Skripts erlaubt. Der *Set*-Operator bewirkt, dass das Objekt ermittelt und eine Referenz auf dieses Objekt in der Objektvariablen *objAdr* hinterlegt wird. Sie können dann später die Objektvariable im Programm verwenden:

```
MsgBox objAdr.Item(0)
```

Diese Anweisung gibt das erste Argument in einem Dialogfeld aus.

Hinweis

An dieser Stelle noch eine kurze Erläuterung zu dem Punkt, der in vielen Namen auftritt. Der Punkt trennt die Namen von Objekten, Methoden oder Eigenschaften voneinander. In der letzten Anweisung bildet *objAdr* den Objektnamen, während *Item* eine Eigenschaft darstellt. Durch den Punkt weiß der Interpreter, wo der Objektname aufhört und wo der nächste Name beginnt. Dies ist auch der Grund dafür, dass Sie im Variablennamen keinen Punkt verwenden dürfen.

Logische Operatoren

VBScript stellt auch einige logische Operatoren zur Auswertung von Ausdrücken zur Verfügung. Diese logischen Operatoren sind in **Tabelle 3.3** aufgeführt.

Operator	Bedeutung
Not	NEGATION ($x = \text{Not } y$)
And	UND ($x = a \text{ And } b$)
Or	ODER ($x = a \text{ Or } b$)
Xor	EXKLUSIV ODER ($x = a \text{ Xor } b$)
Eqv	ÄQUIVALENZ ($x = a \text{ Eqv } b$)
Imp	IMPLIKATION ($x = a \text{ Imp } b$)

Tabelle 3.3:
*Logische
Operatoren*

Logische Operatoren werden häufig in Verzweigungen benutzt. Mit der Anweisung:

```
If a > 100 And a < 1000 Then
```

werden zwei Vergleiche auf $a > 100$ und $a < 1000$ durchgeführt, die beide einen Wert *true* oder *false* liefern können. Anschließend werden beide Ergebnisse mit *And* verknüpft. Nur wenn beide Ergebnisse zutreffen (a liegt im Bereich zwischen 101 und 999) wird die IF-Anweisung ausgeführt.

Sie können die Operatoren *Not*, *And*, *Or* und *Xor* auch verwenden, um Bitoperationen auf Ganzzahlen (Byte, Integer) auszuführen. Für die *Not*-Operation gilt folgendes:

Not	Bit
0	1
1	0

Tabelle 3.4:
Not-Operation

Die Funktion liest das jeweilige Bit und invertiert dessen Wert. Aus einer 0 wird eine 1 und aus einer 1 wird eine 0. Solche Operatoren lassen sich am besten im Binär- oder Hexadezimalsystem demonstrieren. Die Dezimalzahl 3 läßt sich beispielsweise als Binärzahl folgendermaßen darstellen:

0011

Hier wurden nur 4 Bit zur Darstellung verwendet. Wenden Sie die *Not*-Operation auf diesen Wert an, ergibt sich die Darstellung:

1100

was der Hexadezimalzahl 0CH oder der Dezimalzahl 12 entspricht. Beim *And*-Operator werden zwei Bits miteinander verglichen. Nur wenn beide Werte 1 sind, ist auch das Ergebnis 1. Dies lässt sich an folgender Tabelle ablesen, in der die beiden Eingangswerte in den Spalten *Bit* stehen, während das Ergebnis der Verknüpfung in der linken Spalte *And* auftaucht.

Tabelle 3.5:
And-Operation

And	Bit	Bit
0	0	0
0	0	1
0	1	0
1	1	1

Die folgende *And*-Verknüpfung:

MsgBox (3 And 7)

liefert beispielsweise den Wert 3 im Meldungsfeld. In dieser Anweisung werden die beiden Zahlen bitweise mit *And* verknüpft. In der Binärdarstellung besitzt 3 den Wert 0011 und 7 den Wert 0111. Die *And*-Operation liefert folglich das Ergebnis 0011, was dem Wert 3 entspricht.

Der *Or*-Operator verknüpft die einzelnen Bits mit *Oder*, d. h. sobald eines der Bits gesetzt wird, erscheint im Ergebnis der Wert 1. Dies lässt sich in folgender Tabelle ablesen.

Tabelle 3.6:
Or-Operation

Or	Bit	Bit
0	0	0
1	0	1
1	1	0
1	1	1

Die *Or*-Verknüpfung:

MsgBox (3 Or 7)

liefert beispielsweise den Wert 7 im Meldungsfeld. In der Binärdarstellung besitzt 3 den Wert 0011 und 7 den Wert 0111. Die *Or*-Operation liefert folglich das Ergebnis 0111, was dem Wert 7 entspricht. Der letzte häufig benutzte Binäroperator ist *Xor*. Hier wird das Ergebnis 1, falls beide Eingangswerte unterschiedlich sind.

Tabelle 3.7:
Xor-Operation

Xor	Bit	Bit
0	0	0

1	0	1
1	1	0
0	1	1

Vergleichsoperatoren

VBScript stellt auch einige Vergleichsoperatoren zur Verfügung. Den ersten Operator haben Sie bereits im vorherigen Abschnitt in einem Beispiel kennengelernt. Die Vergleichsoperatoren erlauben den Vergleich von Ausdrücken (die auch Zahlen und Zeichenketten beinhalten können). Die nachfolgende Tabelle enthält die in VBScript verfügbaren Vergleichsoperatoren.

Operator	Erklärung
<	Kleiner als ($a < b$)
>	Größer als ($a > b$)
=	Gleich ($a = b$)
<=	Kleiner oder gleich ($a \leq b$)
>=	Größer oder gleich ($a \geq b$)
<>	Ungleich ($a \neq b$)

Tabelle 3.8:
Vergleichsoperatoren

Solche Vergleichsoperatoren werden häufig in Schleifen und Verzweigungen eingesetzt:

```
While a < 10
..
Wend
If a > 100 Then
...
End If
```

Sie werden in den folgenden Kapiteln noch viele Beispiele zum Einsatz solcher Operatoren kennenlernen.

Auf der Begleit-CD-ROM finden Sie im Ordner `\Beisp\Kap03` die Datei `WSHLogic.vbs`. Diese demonstriert Ihnen die Wirkung einiger logischer und Vergleichs-Operatoren durch die Ausgabe der Werte in einem Meldungsfeld. Für die logischen Operatoren habe ich dabei die Darstellung im Hexadezimalzahlenformat gewählt. In diesem Format werden jeweils 4 Bit eines Dezimalwerts zu einer Ziffer im Bereich 0 bis F zusammengefaßt. Die folgende Tabelle enthält eine Gegenüberstellung der Kodierung einer 4-Bit-

Hinweis

Zahl in verschiedenen Zahlensystemen. Beachten Sie beim Ausführen des Beispiels, dass in VBScript die Zahlen dort automatisch als 16-Bit-Integer, also mit 4 Hexadezimalziffern angegeben werden. Der Vergleich *Not 3* liefert daher das Ergebnis *FFF3* als Hexadezimalzahl zurück, da der Wert 3 als 16-Bit Zahl der Form *0003* im Hexadezimalsystem dargestellt wird.

Tabelle 3.9:
Kodierung in
verschiedenen
Zahlensystemen

Dezimal	Hexadezimal	Binär
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Wichtig

Beachten Sie beim Einsatz von logischen und Vergleichsoperatoren, dass VBScript bei ungleichen Subtypen in der *Variant*-Variablen automatisch eine Konvertierung vornimmt. Dadurch kann es vorkommen, dass ungleiche Werte durch die Konvertierung als gleich interpretiert werden.

Prioritäten von Operatoren

Operatoren lassen sich klammern, um bestimmte Prioritäten bei der Abarbeitung zu erzwingen. Ohne Klammerung gelten implizite Prioritäten (Potenzieren, Negation, Multiplikation/Division, Ganzzahldivision, Modulo, Addition/Subtraktion, Zeichenfolgenverkettung). In obiger Folge besitzt die

Potenzierung die höchste Priorität. Bei Vergleichen gilt die Reihenfolge (=, <>, <, >, <=, >=), wobei das Gleichheitszeichen = die höchste Priorität besitzt. Bei logischen Vergleichen besitzt der *Not*-Operator die höchste Priorität. Es gilt die folgende Reihenfolge *Not*, *And*, *Or*, *Xor*, *Eqv*, *Imp*, *&*.

Kontrollstrukturen

VBScript kennt verschiedene Kontrollstrukturen zur Steuerung des Programmablaufs. Der nachfolgende Abschnitt enthält eine kurze Übersicht über die für diese Zwecke in VBScript verfügbaren Anweisungen.

If ... Then

Die *If*-Anweisung lässt sich benutzen, um eine einfache Verzweigung oder eine Anweisung aufgrund eines Vergleichs durchzuführen. Die Anweisung:

```
If a > 100 Then a = 100
```

setzt die Variable *a* auf den Wert 100 zurück, falls dieser größer als 100 war. Mit der folgenden Sequenz:

```
If a > 100 Then  
    a = 100  
    b = 20  
End If
```

erreichen Sie, dass der Wert der Variablen *a* überprüft wird. Ist der Wert größer als 100, werden die Anweisungen zwischen *If .. Then* und *End If* ausgeführt. Trifft die Bedingung nicht zu, wird das Programm hinter der *End If*-Zeile fortgesetzt.

If ... Then ... Else

Diese Variante der *If*-Anweisung lässt sich benutzen, um zwei Blöcke mit zu verarbeitenden Anweisungen zu definieren. Die Sequenz:

```
If a > 100 Then  
    a = 100  
    b = 20  
Else  
    a = a + 10  
    b = a \ 10
```

End If

überprüft den Wert der Variablen *a*. Ist der Wert größer als 100, werden die Anweisungen zwischen *If .. Then* und *Else* ausgeführt. Trifft die Bedingung nicht zu, werden die Anweisungen zwischen *Else* und *End If* bearbeitet.

If ... Then ... Elself

Diese Variante der *If*-Anweisung erlaubt eine Schachtelung mehrerer *If*-Blöcke. Die folgende Sequenz wendet diese Konstruktion an:

```
If a = 1 Then
  b = 100
  c = 20
ElseIf a = 2 Then
  b = 200
  c = 40
ElseIf a = 3 Then
  b = 300
  c = 60
End If
```

Hier wird die Variable *a* auf verschiedene Werte überprüft. Liefert ein Vergleich den Wert *true*, werden die Bedingungen zwischen *Elself* und der nächsten *Elself*- oder *End If*-Anweisung ausgeführt.

Hinweis

Diese Art der Abfrage ist aber sehr undurchsichtig und fehlerträchtig. Sie sollten daher die *Select Case*-Anweisung vorziehen.

Select Case

Mit dieser Anweisung können Sie eine Variable auf mehrere Werte überprüfen und in Abhängigkeit vom Wert bestimmte Anweisungsblöcke formulieren. Die folgende Sequenz wendet diese Konstruktion an:

```
Select Case a
  Case 1
    b = 100
    c = 20
  Case 2
    b = 200
    c = 40
  Case 3
    b = 300
    c = 60
```

```
Case Else
    b = 0
    c = 0
    a = 1
End Select
```

Hier wird die Variable *a* ebenfalls ausgewertet. Die nachfolgenden *Case*-Anweisungen geben den zu überprüfenden Wert vor. Trifft die Bedingung zu, werden die Anweisungen hinter der *Case*-Anweisung bis zur nächsten *Case*-Anweisung ausgeführt. Trifft keine der Abfragen zu, wird der (optionale) *Case Else*-Zweig ausgeführt.

Die *Select Case*-Anweisung eignet sich sehr gut, um mehrfache Vergleiche auf einen bestimmten Wert durchzuführen. Der Code bleibt trotz der vielen Abfragen übersichtlich. Hinweis

Schleifen

Schleifen erlauben die wiederholte Ausführung bestimmter Anweisungen innerhalb der VBScript-Programme. Der nachfolgende Abschnitt enthält eine kurze Übersicht über die für diese Zwecke in VBScript verfügbaren Anweisungen.

Do While ... Loop

Die *Do While*-Anweisung erzeugt eine Schleife, die in Abhängigkeit von einer am Schleifenanfang ausgewerteten Bedingung die Anweisungen des Blocks ausführt oder überspringt. Die Anweisungen der Schleife werden ausgeführt, wenn die Bedingung den Wert *true* liefert. Die folgende Sequenz zeigt den Einsatz dieser Anweisung:

```
a = 1
Do While a < 10
    a = a + 1
```

Loop

In dieser Schleife wird beim Eintritt die Bedingung *a < 10* geprüft. Solange diese erfüllt ist, führt das System die Anweisungen bis zum *Loop*-Statement aus. Danach wird die Bedingung am Schleifenanfang erneut geprüft. Trifft die Bedingung nicht mehr zu, führt VBScript den auf die *Loop*-Anweisung folgenden Befehl aus.

Die Bedingung am Schleifenanfang muß die Werte *false* oder *true* liefern, damit die Schleife ausgeführt und irgendwann beendet wird.

Do Until ... Loop

Die *Do Until*-Anweisung erzeugt eine Schleife, die beim Eintritt in die Schleife überprüft wird. Ist die Bedingung *false*, wird die Schleife ausgeführt. Die Schleife wird unterbrochen, sobald die Bedingung *true* wird. Die folgende Sequenz zeigt den Einsatz dieser Anweisung:

```
a = 1
Do Until a > 10
    a = a +1
```

Loop

In dieser Schleife wird beim Eintritt die Bedingung *a > 10* geprüft. Solange dies *nicht* erfüllt ist, führt das System die Anweisungen bis zum *Loop*-Statement aus. Danach wird die Bedingung am Schleifenanfang erneut geprüft. Trifft die Bedingung zu, führt VBScript den auf die *Loop*-Anweisung folgenden Befehl aus.

Do ... Loop While

Mit *Do ... Loop While* läßt sich eine Schleife erzeugen, die erst am Schleifenende überprüft wird. Ist die Bedingung *true*, wird die Schleife weiter ausgeführt. Die Schleife wird unterbrochen, sobald die Bedingung *false* wird. Die folgende Sequenz zeigt den Einsatz dieser Anweisung:

```
a = 1
Do
    a = a +1
```

Loop While a < 10

In dieser Schleife wird am Ende die Bedingung *a < 10* geprüft. Solange dies erfüllt ist, führt das System die Anweisungen zwischen *Do* und *Loop* aus. Danach wird die Bedingung am Schleifenanfang erneut geprüft. Trifft die Bedingung nicht mehr zu, führt VBScript den auf die *Loop*-Anweisung folgenden Befehl aus.

Do ... Loop Until

Mit *Do ... Loop Until* lässt sich eine Schleife erzeugen, die erst am Schleifenende überprüft wird. Ist die Bedingung *false*, wird die Schleife weiter ausgeführt. Die Schleife wird unterbrochen, sobald die Bedingung *true* wird. Die folgende Sequenz zeigt den Einsatz dieser Anweisung:

```
a = 1
Do
    a = a + 1

Loop Until a > 10
```

In dieser Schleife wird am Ende die Bedingung $a > 10$ geprüft. Solange dies *nicht* erfüllt ist, führt das System die Anweisungen bis zum *Loop*-Statement aus. Danach wird die Bedingung am Schleifenanfang erneut geprüft. Trifft die Bedingung zu, führt VBScript den auf die *Loop*-Anweisung folgenden Befehl aus.

Exit Do

Die Anweisung *Exit Do* lässt sich in alle *Do*-Schleifen einsetzen, um die Schleife direkt abubrechen. Erreicht der Interpreter diese Anweisung, beendet er die Schleife und setzt die Programmausführung mit der Anweisung, die auf das Schleifenende folgt, fort.

For ... Next

Mit einer *For*-Schleife lässt sich eine vordefinierte Anzahl an Durchläufen erzeugen. Die Anweisungen innerhalb des *For ... Next*-Blocks werden bei jedem Durchlauf ausgeführt. Die folgende Sequenz zeigt den Einsatz dieser Anweisung:

```
For i = 1 To 10
    a = a + 1
Next
```

Diese Schleife wird 10 mal durchlaufen. Der Wert *i* gibt dabei den Schleifenindex an.

Die Schrittweite in der Schleife wird standardmäßig auf 1 gesetzt. Sie können aber die Variante *For i = start To ende Step x* wählen, wobei der Wert *x* für die Schrittweite steht. Hinweis

VB oder VBA-Programmierer werden häufig über eine VBScript-Einschränkung stolpern (zumindest geht es dem Autor so). Üblicherweise gibt Wichtig

man hinter *Next* den Namen der Schleifenvariable an, um die Lesbarkeit des Programmes zu erhöhen. Bei VBScript wird der Name der Schleifenvariable hinter der *Next*-Anweisung nicht akzeptiert. Der Interpreter löst einen Fehler aus!

For Each ... Next

Mit einer *For Each*-Schleife läßt sich eine Auflistung (Collection) oder ein Feld (Array) bearbeiten. Die Schleife wird für jedes Element der Auflistung oder des Feldes durchlaufen. Die folgende Sequenz zeigt den Einsatz dieser Anweisung für den Zugriff auf eine Auflistung:

```
For Each x In Worksheets
...
Next
```

Exit For

Die Anweisung *Exit For* läßt sich in alle *For*-Schleifen einsetzen, um die Schleife vorzeitig zu beenden. Erreicht der Interpreter diese Anweisung, beendet er die Schleife und setzt die Programmausführung mit der Anweisung, die hinter *Next* folgt, fort.

While ... Wend

Mit einer *While ... Wend*-Schleife können Sie eine Folge von Anweisungen wiederholt bearbeiten. Die Schleife wird beendet, sobald die Eingangsbedingung falsch wird. Die folgende Sequenz zeigt den Einsatz dieser Anweisung:

```
Dim zahl
Zahl = 1
While Zahl < 10
    Zahl = Zahl + 1
...
Wend
```

Die obige Schleife wird so lange durchlaufen, bis der Wert der Variablen *Zahl* den Wert 10 erreicht. Mit der *Do ... Loop*-Anweisungen können Sie aber die gleichen Effekte erreichen.

Prozeduren und Funktionen

In VBScript können Sie sowohl vordefinierte Prozeduren und Funktionen aufrufen, als auch eigene Funktionen und Prozeduren definieren. Nachfolgend erfahren Sie, was für in VBScript beim Erstellen von Prozeduren und Funktionen zu beachten ist.

Funktionen

Verwenden Sie Funktionen, falls nur ein einziger Wert als Ergebnis an das aufrufende Programm zurückzugeben ist. Eine Funktion wird mit den Anweisungen:

```
Function Name (Argumente)
...
Name = xxx
End Function
```

definiert. Der Rückgabewert muß dem Funktionsnamen explizit innerhalb der Funktion zugewiesen werden. In VBScript liefern die Funktionen immer einen Wert vom Datentyp *Variant* zurück. In den Klammern lassen sich die Parameter (d. h. Argumente) für den Funktionsaufruf übergeben. Das folgende Listing verdeutlicht den Einsatz einer Funktion in VBScript.

```
' *****
' File:   Function.vbs (WSH-Beispiel in VBScript)
' Autor:  (c) Günter Born
'
' Zweck:  Zeigt den Einsatz einer Funktion.
' *****

Dim i, j

j = 0

For i = 1 to 10    ' Versuche 10 Durchläufe
    j = addiere (i, j) ' Addiere per Funktion
Next

WScript.Echo "Ergebnis: ", j

WScript.Quit

Function addiere (wert1, wert2)
```

Listing 3.3:
*Beispiel für eine
Funktion*

```

    addiere = wert1 + wert2
End Function
'*****
'***           Ende -> WSH-VBScript           ***
'*****

```

In diesem Beispiel wird die Funktion *addiere* vereinbart, die zwei Parameter *wert1* und *wert2* erwartet. Diese Parameter werden innerhalb der Funktion addiert und an das rufende Programm zurückgegeben. Dies erfolgt durch eine einfache Zuweisung innerhalb der Funktion:

```
addiere = wert1 + wert2
```

Mit *addiere* ist der Funktionsname gemeint, an den das Ergebnis der Funktion zugewiesen wird. Die Funktion wird beendet, sobald die Anweisung *End Function* erreicht wird.

Hinweis

Sie haben in VBScript aber zusätzlich die Möglichkeit, eine *Exit Function*-Anweisung einzubauen. Dann wird die Funktion bei Erreichen dieser Anweisung vorzeitig beendet.

Der Aufruf einer Funktion erfolgt wie der Zugriff auf eine der internen VBScript-Funktionen. Sie geben den Funktionsnamen sowie die benötigten Parameter auf der rechten Seite einer Zuweisung an. In obigem Listing ist dies die Anweisung:

```
j = addiere (i, j)
```

Mit diesem Ansatz können Sie den VBScript-Funktionsumfang sehr elegant erweitern. Nehmen wir an, Sie müssen häufiger die Berechnung von Bruttopreisen einschließlich Mehrwertsteuer vornehmen. Dann können Sie sich eine Funktion *Brutto* mit folgenden Anweisungen erstellen:

Listing 3.4: Funktion zur Ermittlung des Bruttobetrags

```

Function Brutto(Netto, MwSt) ' Ermittle Bruttobetrag
    Brutto = Netto * (1.0 + MwSt/100.0)
End Function

```

Der Mehrwertsteuersatz läßt sich dann direkt als Prozentwert an die Funktion übergeben. Um die Funktion in einer Anweisung zu verwenden, geben Sie nur noch den Namen sowie die Argumente an.

```

Netto = 10.0
MwSt = 16.0
Preis = Brutto (Netto, MwSt)
Preis1 = Brutto (100.0, 16.0)

```

Die obigen Anweisungen zeigen, dass Sie als Argumente sowohl Konstante als auch Variable verwenden dürfen.

Gültigkeit von Funktionen

An dieser Stelle möchte ich Ihnen noch einige Hinweise zur Gültigkeit von Funktionen geben. Sofern Sie die Funktion gemäß der obigen Beschreibung deklarieren, ist der Funktionsname innerhalb des kompletten Skripts gültig. Sie haben aber die Möglichkeit, die Gültigkeit durch Voranstellen der Schlüsselwörter:

```
Public Function xxx ()  
Private Function xxx ()
```

explizit festzulegen. Mit *Private* erreichen Sie, dass die Funktion nur innerhalb des Skripts gültig ist. *Public* vereinbart dagegen eine Gültigkeit, die über das Skript bzw. Modul hinausgeht.

Auf den ersten Blick ist diese Gültigkeitsbetrachtung für WSH-Skripte nicht relevant. Die Schlüsselwörter benötigen Sie aber, wenn Sie VBScript in einem HTML-Dokument verwenden, welches mehrere `<SCRIPT>...</SCRIPT>`-Blöcke aufweist. Sie können beispielsweise aus WSH-Skripten auf solche *Public*-Funktionen in einem HTML-Skript zugreifen. Daher habe ich die obigen Erläuterung an dieser Stelle mit aufgenommen. Hinweis

Sie können keine Funktion innerhalb einer anderen Funktion oder Prozedur definieren. Der Code muß daher auf der Ebene des Skripts erstellt werden. Weiterhin ist zu beachten, dass lokale Variable innerhalb der Funktion nur während des Funktionsaufrufs Gültigkeit haben. Wichtig

Funktionsargumente mit ByRef/ByVal übergeben

Beim Funktionsaufruf können Sie die Argumente mittels Call by Value oder mittels Call by Reference übergeben. Dies muß durch die Schlüsselworte *ByVal* bzw. *ByRef* erfolgen:

```
Public Function Brutto(ByRef Netto, ByVal MwSt) ' Ermittelt Bruttobetrag  
    Brutto = Netto * (1.0 + MwSt/100.0)  
End Function
```

Findet der Interpreter die Angabe *ByVal*, wird lediglich der Wert des Arguments an die Funktion übergeben. Verwenden Sie dagegen *ByRef*, erhält die Funktion eine Adresse des Arguments als Parameter (➡ siehe auch die folgenden Abschnitte).

Interne Funktionen

VBScript ist mit einer ganzen Reihe interner Funktionen ausgestattet. Sie können diese Funktionen wie selbst definierte Funktion verwenden, um bestimmte Operationen auszuführen. Mit der Anweisung:

```
i = Asc ("A")
```

weisen Sie beispielsweise der Variablen *i* den ANSI-Code des Zeichens *A* zu. Dies wird durch die interne Funktion *Asc* erreicht.

Weitere Informationen zu Funktionen (und insbesondere zu den internen Funktionen) finden Sie in der VBScript-Sprachreferenz auf der Begleit-CD-ROM.

Prozeduren

Neben den Funktionen sind Prozeduren die Hauptbestandteile der VBScript-Anwendungen. Prozeduren werden in der Form:

```
Sub Name (Parameter)
...
End Sub
```

innerhalb eines Skripts definiert. Eine Prozedur liefert im Gegensatz zu einer Funktion keinen Rückgabewert zurück. Um Variablenwerte mit der Prozedur auszutauschen, haben Sie mehrere Möglichkeiten:

- ♦ Sie deklarieren die Variablen auf Skriptebene. Dadurch sind diese Variablen global, also auch innerhalb der Prozedur gültig.
- ♦ Sie übergeben die Variable als Parameter an die Prozedur. Hierbei wird noch unterschieden, ob die geänderten Werte im rufenden Programm wirksam werden.

Die folgende Codesequenz zeigt den Einsatz einer Prozedur zur Berechnung der Bruttopreise.

Listing 3.5:
*Einsatz einer
Prozedur*

```
'*****
' File:   Prozedur.vbs (WSH-Beispiel in VBScript)
' Autor:  (c) Günter Born
'
' Zweck:  Zeigt den Einsatz einer Prozedur.
'*****

Dim Preis, Netto, MwSt

MwSt = 16.0
Netto = 100.0

Brutto Netto, MwSt
WScript.Echo "Ergebnis: ", Preis

WScript.Quit
```

```

Sub Brutto (net, tax)
    Preis = net * (1.0 + (tax/100.0))
End Sub

'*****
'***           Ende -> WSH-VBScript           ***
'*****

```

Hier werden gleich mehrere Techniken demonstriert. Auf der Skriptebene werden die globalen Variable definiert. Diese lassen sich folglich innerhalb der Prozedur nutzen, was bei der Variablen *Preis* recht hilfreich ist. Diese wird in der Prozedur direkt verändert. Weiterhin erwartet die Prozedur zwei Parameter, die beim Aufruf mit übergeben werden. Beachten Sie die Syntax des Prozeduraufrufs. Es ist der Name gefolgt von den durch Kommas getrennten Parametern anzugeben. Eine Zuweisung wie bei einer Funktion ist nicht zulässig.

Die *Call*-Anweisung ist beim Prozeduraufruf nicht erforderlich. Verwenden Sie dieses Schlüsselwort, müssen Sie beim Prozeduraufruf die Parameter in Klammern setzen. Die Anweisungen *Call Brutto (100.0, 0.16)* entspricht funktional dem Aufruf *Brutto 100.0, 0.16*. Wichtig

Genau wie bei Funktionen können Sie einer Prozedur noch die Schlüsselwörter *Public* und *Private* in der Deklaration voranstellen, um die Gültigkeit über die Skriptebene hinaus zu beeinflussen. Hinweis

Anmerkungen zur Parameterübergabe (ByRef, ByVal)

Beim Aufruf von Funktionen und Prozeduren sind die Parameter in der rufenden Prozedur durch Kommas getrennt anzugeben. Die Parameter stellen für die Prozedur eine Art lokale Variable dar. Hierbei werden nicht die Werte der Variablen sondern deren Adressen an die Prozedur übergeben. Dies erlaubt es, die Werte der Parameter in der Prozedur zu verändern. Diese Veränderungen werden unter Umständen auch im rufenden Programm wirksam.

In obigem Listing wurde die Variable *Preis* auf Skriptebene vereinbart, damit der berechnete Wert aus der Prozedur an das Skript zurückgegeben werden kann. Dies ist aber nicht erforderlich, wie der nachfolgende leicht modifizierte Code zeigt.

```

'*****
' File:   Prozedur1.vbs (WSH-Beispiel in VBScript)
' Autor:  (c) Günter Born
'
' Zweck:  Zeigt den Einsatz einer Prozedur.
'*****
Dim Preis, Netto, MwSt

```

Listing 3.6:
Modifizierter
Prozeduraufruf

```

MwSt = 16.0
Netto = 100.0

Call Brutto (Preis, Netto, MwSt)
WScript.Echo "Ergebnis: ", Preis

WScript.Quit

Sub Brutto (pris, net, tax)
    pris = net * (1.0 + (tax/100.0))
End Sub

'*****
'***           Ende -> WSH-VBScript           ***
'*****

```

Beim Prozeduraufruf wird als zusätzlicher Parameter die Variable *Preis* mit angegeben. Dieser Parameter wird innerhalb der Prozedurdefinition von *Brutto* als *Pris* vereinbart. Verändert die Prozedur *Brutto* den Wert für *Pris* wirkt sich dies auf die beim Aufruf der Prozedur als Parameter angegebene Variable *Preis* aus. In der Folge kann das rufende Programm direkt das in der Prozedur berechnete Ergebnis auswerten. Die Definition globaler Variable entfällt. In diesem Beispiel wurde weiterhin die *Call*-Anweisung zur Demonstration verwendet. Diese hat aber keinen Einfluß auf die Funktion der Prozedur.

Hinweis

Sie finden das Programm *Prozedur1.vbs* im Ordner *\Beisp\Kap03* auf der Begleit-CD.

Diese Art der Parameterübergabe an Prozeduren ist in VBScript Standard und wird als *Call by Reference* bezeichnet. Der Interpreter übergibt die Adresse der jeweiligen Variablen an die Prozedur. Diese kann über die Adresse direkt den Wert der betreffenden Variablen manipulieren. Die Übergabe als Referenz läßt sich durch das Schlüsselwort *ByRef* innerhalb der Prozedurdefinition auch explizit angeben.

In verschiedenen Fällen ist es jedoch unerwünscht, wenn eine Prozedur die übergebenen Parameter ändert und sich dies auf Variable im rufenden Programm auswirkt (dies kann beim Aufruf externer DLL-Routinen sogar tödlich sein). In obigem Beispiel müssen die beiden Parameter *net* und *tax* nicht durch die Prozedur verändert werden. Um zu verhindern, daß sich Änderungen im rufenden Programm auswirken, können Sie in VBScript die Übergabe der Parameter als Wert vereinbaren. Dies wurde in folgendem Beispiel genutzt.

Listing 3.7:
Parameterübergabe
mit *ByVal*

```

'*****
' File:   Prozedur2.vbs (WSH-Beispiel in VBScript)
' Autor:  (c) Günter Born
'
' Zweck:  Zeigt den Einsatz von Call-by-Value an einer Prozedur.
'*****

Dim Preis, Netto, MwSt

MwSt = 16.0
Netto = 100.0

Call Brutto (Preis, Netto, MwSt)
WScript.Echo "Ergebnis: ", Preis, " MwSt: ", MwSt

WScript.Quit

Sub Brutto (ByRef pris, ByVal net, ByVal tax)
    pris = net * (1.0 + (tax/100.0))
    tax = 17.0
End Sub

'*****
'***           Ende -> WSH-VBScript           ***
'*****

```

Hier wurde innerhalb der Deklaration jeweils angegeben, ob der Parameter *ByVal* oder *ByRef* zu übergeben ist. *Preis* wird als Referenz übergeben, während *net* und *tax* als Werte der gerufenen Prozedur zur Verfügung gestellt werden. Änderungen an *net* oder *tax* dürften sich daher auf das rufende Programm nicht mehr auswirken. Zur Überprüfung wird innerhalb der Prozedur den Wert des Parameters *tax* verändert. Gemäß der Definition *ByVal* in der Parameterdeklaration sollte sich dies nicht auf das rufende Programm auswirken. Der Aufruf erfolgt mit der Anweisung:

```
Call Brutto(Preis, Netto, MwSt)
```

Aber das Programm verhält sich jetzt anders. Während das Ergebnis im Parameter *Preis* zurückgegeben wird, bleibt der ursprünglichen Wert von *tax* erhalten. Sie sehen dies, sobald Sie das Beispiel ausführen.

Den Beispielcode für *Prozedur2.vbs* finden Sie im Ordner *\Beisp\Kap03* auf der Begleit-CD-ROM. Hinweis

Beachten Sie, dass VBScript verschiedene aus VB und VBScript bekannte Konstruktionen wie *On Error Goto* nicht unterstützt! Es wird lediglich die *On Error Resume Next*-Anweisung unterstützt. Auf diesen Sachverhalt komme ich später noch zurück. Sie finden eine Übersicht über die fehlenden Konstrukte in der VBScript-Referenz. Wichtig

An dieser Stelle möchte ich die Einführung in VBScript beenden. Sie finden die komplette VBScript-Sprachreferenz auf der Begleit-CD-ROM im Ordner *\Infos\VBScript*. Starten Sie die EXE-Datei *vbdoc.exe* wird die Dokumentation lokal auf dem Rechner installiert. Sie können die Dokumente über das Startmenü aufrufen. Alternativ können Sie unter Windows 98 direkt auf die Datei *Vbscript.chm* zugreifen.

Tip

Falls Sie mit VB 5 CCE arbeiten, gilt der Sprachumfang aus Visual Basic. Eine ausführlichere Einführung in Visual Basic (VBA), der sich zum Arbeiten mit der CCE eignet, finden Sie in dem im Literaturverzeichnis unter /5/ aufgeführten Titel.

Grundlagen und Hinweise	107
Ausdrücke und Operatoren	117
Kontrollstrukturen	121
Built-in-Objekte und -Funktionen	128

In diesem Kapitel erhalten Sie eine kurze Einführung in die Sprache JScript. Wer von JavaScript oder Java umsteigt, wird sich mit JScript sofort zurecht finden.

- ◆ Lernen Sie, wie ein JScript-Programm aufgebaut ist und wie Sie mit Variablen oder Konstanten umgehen.
- ◆ Erfahren Sie, welche Konstrukte JScript zur Programmsteuerung bietet.
- ◆ Lesen Sie nach, wie Funktionen in JScript gehandhabt werden und was es sonst noch zu beachten gilt.

Mit diesen Informationen sollten Sie sehr schnell eigene Skripte erstellen können. Wer die Sprache JScript sehr gut beherrscht, kann dieses Kapitel problemlos übergehen.

Grundlagen und Hinweise

In diesem Abschnitt erhalten Sie einige grundsätzliche Hinweise zur Sprache sowie zu den Ursprüngen von JScript.

Was ist JScript?

JScript ist die Microsoft-Implementierung von ECMA 262 Script. ECMA steht als Abkürzung für European Computer Manufacturer. Diese Gruppe übernahm die herstellerübergreifende Standardisierung dessen, was von

Netscape seinerzeit als JavaScript im Navigator (als »Subset« von Java) implementiert wurde. JavaScript stimmt daher weitgehend mit der ECMA 262-Spezifikation überein. Grob gesagt können Sie die Kenntnisse über die Sprache JavaScript für JScript-Programme nutzen, da lediglich einige Netscape-spezifische Anweisungen sowie bestimmte Objekte und Methoden im WSH entfallen. JScript stellt dabei eine vollständige Implementierung der ECMA 262-Spezifikation dar, die lediglich durch einige Verbesserungen im Hinblick auf die Funktionen des Microsoft Internet Explorers erweitert wurde. Diese Erweiterungen brauchen Sie in WSH-Programmen aber kaum zu berücksichtigen.

Bei JScript handelt es sich wie bei VBScript um eine interpretierte Sprache. Sie brauchen daher nur den Quellcode einzutippen und können die betreffende *.js*-Datei sofort im WSH ausführen. JScript ist (ähnlich wie C++ und Java) vollständig objektorientiert. Nachfolgend finden Sie einige Hinweise zur Gestaltung von JScript-Programmen.

Die Struktur eines JScript-Programms

Sofern Sie mit JScript oder JavaScript im Zusammenhang mit HTML-Dokumenten gearbeitet haben, hier die erste wichtige Neuerung: JScript-Programme besitzen im Gegensatz zu in HTML-Dokumenten eingebundenem JavaScript-Code keinerlei HTML-Tags. Vielmehr wird das komplette Skript in der *.js*-Datei in der JScript-Syntax hinterlegt. Das folgende Listing zeigt den typischen Aufbau eines JScript-Programms:

Listing 4.1: *JScript-Programm*

```
//*****
// File: FehlerTest1.js
// Autor: (c) G. Born
// Sprache: JScript (WSH)
//
// Testbeispiel mit einem Fehler zum Test im Debugger.
//*****
// Windows Script ()

#####
debugger; // Debugger starten
#####

var vbOKCancel = 1; // Variablen vereinbaren
var vbInformation = 64;
var vbCancel = 2;

var L_Welcome_MsgBox_Message_Text = "Testbeispiel";
```

```

var L_Welcome_MsgBox_Title_Text = "Borns Windows Scripting Host-Beispiel";

Welcome();    // Eingangsdialog zeigen

WScript.Echo("Das Beispiel wurde ausgeführt");

////////////////////////////////////////
//
// Welcome
//
function Welcome() {
    var WSHShell = WScript.CreateObject("WScript.Shell");
    var intDoIt;

    intDoIt = WSHShell.Popup(L_Welcome_MsgBox_Message_Text,
                             0,
                             L_Welcome_MsgBox_Title_Text,
                             vbOKCancel + vbInformation );

    if (intDoIt == vbCancel) {
        WScript.Quit();
    }
}

//*****
//*** End, Fin, Finito, Fertig or whatever else ***
//*** WSH powered by Günter Born          ***
//*****

```

Sofern Sie das vorherige Kapitel zu VBScript gelesen haben, ergeben sich einige Unterschiede, die ich kurz zusammenstellen möchte.

Sie finden das obige Beispielprogramm *FehlerTest1.js* im Ordner *\Beisp\Kap04* auf der Begleit-CD-ROM. JScript-Programme lassen sich auch in HTML-Dokumente einbinden und im Internet Explorer 4.0 ausführen. Es gibt aber geringfügige Abweichungen im Hinblick auf die zulässigen Funktionen zwischen JScript-Programmen im WSH und den JScript-Anweisungen in einem HTML-Dokument. So ist die komplette Ereignisverwaltung, die JScript-Skripte in HTML-Dokumenten unterstützen müssen, im WSH-Skript weitgehend ohne Bedeutung. Bei einem WSH-Skript treten solche Ereignisse wie das Anklicken einer Schaltfläche nicht auf. Hinweise zum Einbinden von JScript-Anweisungen in HTML-Dokumente sowie Beispiele finden Sie in dem im Literaturverzeichnis unter /6/ aufgeführten Titel.

Hinweis

Kommentare

Möchten Sie, dass JScript eine Zeile oder einen Teil einer Zeile nicht als Anweisung interpretiert? Dann müssen Sie diese Anweisung als Kommentar markieren. Kommentare werden in JScript durch zwei Slash-Zeichen eingeleitet.

```
// Dies ist ein Kommentar  
WScript.Quit(); // Jetzt sind wir fertig...
```

Ein Kommentar kann am Zeilenanfang oder hinter einer Anweisung folgen. Der Rest der Zeile wird dann vom Übersetzer/Interpreter ignoriert.

Hinweise zu JScript-Anweisungen

JScript-Anweisungen sind im Editor gemäß den jeweils geltenden Syntaxregeln einzugeben. Gegenüber VBScript und vielen anderen Programmiersprachen ergeben sich allerdings einige gravierende Abweichungen.

Wichtig

Ganz wichtig: Da sich JScript stark an die Syntax von C anlehnt, müssen Sie im Programm auf die Schreibweise achten. Bei Funktionen und Variablen wird die Groß-/Kleinschreibung im Namen unterschieden. Es ist daher nicht egal, ob Sie beispielsweise eine Funktion mit *res = Born()* oder mit *res = born()* aufrufen. Gerade dieser Fehler wird beim Einstieg in die JScript-Programmentwicklung häufig gemacht.

Weiterhin muß jede Anweisung in der Regel mit einem Semikolon ; abgeschlossen werden. Einzige Ausnahme stellen Anweisungen dar, die vor einer schließenden Klammer } stehen, die das Blockende markiert. Es schadet aber auch nichts, wenn Sie diese Zeile durch ein Semikolon abschließen. Die folgenden Zeilen enthalten einige gültige JScript-Anweisungen:

```
Wert = 10;  
Wert = Wert + 10;  
Mwst = 0.1;
```

Tip

Dezimalzahlen sind im JScript-Code mit einem Punkt und nicht mit einem Komma anzugeben.

Fortsetzungszeilen

JScript kennt keine Zeichen für Fortsetzungszeilen. Durch den Abschluß einer Anweisung per Semikolon ist das Ende einer Anweisung eindeutig erkennbar. Daher dürfen Sie einen JScript-Befehl ggf. auf mehrere Zeilen aufteilen. Die folgenden Zeilen enthalten einen gültigen JScript-Befehl:

```
WScript.Echo ("Hallo",  
    "Du da ");
```

Beachten Sie, dass hinter einer Fortsetzungszeile kein Kommentar folgen darf. Bei Zeichenketten in Fortsetzungszeilen müssen diese mit einem Anführungszeichen abgeschlossen und die Teilzeichenketten mit dem `+`-Zeichen verknüpft werden. Hinweis

Mehrere Anweisungen pro Zeile

In VBScript ist der Doppelpunkt zur Trennung mehrerer in einer Zeile enthaltene Anweisungen zulässig (siehe [↗](#) Kapitel 3). Dies funktioniert in JScript aber nicht. Vielmehr müssen Sie mehrere Anweisungen innerhalb einer Zeile mit Kommas trennen. Die folgende Sequenz:

```
var x = 15, y = 20;  
WScript.Echo (x + y);
```

liefert den Wert 35 in einem Meldungsfeld. Aus Gründen der Übersichtlichkeit und Programmtransparenz sollten Sie aber (wie in VBScript) auf diese Konstruktion verzichten. Oder erkennen Sie beispielsweise sofort bei der Anweisung:

```
For (var i = 0; i <= 10; i++, j++)
```

den tieferen Sinn? Hier wird das Komma benutzt, um innerhalb des Schleifenkopfes zusätzlich die Variable *j* bei jedem Durchlauf zu erhöhen. Es ist aber sicherlich kein Problem, die Anweisung zum Erhöhen der Variable *j* in den Schleifenrumpf zu verlagern.

Konstanten

Konstanten sind Zahlen oder Zeichenketten, die in JScript-Programmen direkt in den betreffenden Anweisungen angegeben werden. JScript unterstützt verschiedene Möglichkeiten, um Konstanten direkt in Anweisungen anzugeben.

```
Ergebnis = 15 + 10;  
Name = "Born";  
Pi = 3.14;
```

Die erste Anweisung besitzt die beiden Konstanten 15 und 10, die hier addiert (es sind also Berechnungen in Anweisungen möglich) und der Variablen *Ergebnis* zugewiesen werden. In der zweiten Zeile wird die Textkonstante *"Born"* einer Variablen zugewiesen. Die letzte Zeile definiert ebenfalls eine Konstante mit dem Wert 3.14.

JScript kennt keine vordefinierten symbolische Konstante wie VBScript. Möchten Sie mit symbolischen Konstanten (z. B. *vbOkonly*) arbeiten, müssen Sie diese Konstante als Variable deklarieren.

Variablen

Variablen ermöglichen Ihnen, in JScript Werte unter einem Namen zu speichern und im Programm zu verwenden. Variablen müssen in JScript vor ihrer Verwendung zunächst deklariert werden. Dies geschieht entweder implizit durch Zuweisung eines Werts zu einem Namen. Die folgenden Anweisungen nutzen dies:

```
Preis = 17;  
MwSt = 16;
```

Eine gute Praxis ist es aber, die Variable über das Schlüsselwort *var* explizit zu deklarieren. Dies wird in den folgenden Zeilen genutzt:

```
var text;      // Variable ohne Initialisierungswert  
var x = 19;  
var Preis = 19;  
var y = Math.sin(x);  
var x += y;  
var text = "Wert ";
```

Die erste Zeile definiert lediglich eine Variable, der noch kein Wert zugewiesen wird. In den restlichen Zeilen werden die Variablen vereinbart und gleichzeitig erhalten sie einen Initialisierungswert zugewiesen.

Bemerkungen zur Gültigkeit von Variablen

Vielleicht fragen Sie sich, was die Unterscheidung bei der Variablendefinition im vorhergehenden Abschnitt soll? Warum soll ich denn eine Variable mit *var* definieren, wenn auch die Zuweisung eines Werts zu einem Namen ausreicht? Hier ein Argument, welches den kleinen, aber feinen Unterschied herausstellt: Die Art der Deklaration beeinflusst den Gültigkeitsbereich (Scope) einer Variablen. Der Scope bestimmt, wie Sie auf die Variable zugreifen können:

- ♦ Eine innerhalb einer Funktion mit *var* deklarierte Variable ist nur lokal innerhalb der Funktion gültig (z. B. *var summe = 0;*).
- ♦ Deklarieren Sie eine Variable innerhalb oder außerhalb einer Funktion durch eine einfache Wertzuweisung (z. B. *MwSt = 16;*), ist diese Variable global gültig, d. h. Sie können deren Wert im kompletten Skript verwenden.

Persönlich kann ich Ihnen daher folgende Tips geben: Arbeiten Sie in Funktionen ausschließlich mit der *var*-Anweisung, um den Scope der Variablen zu begrenzen. Benötigen Sie global gültige Variable, deklarieren Sie diese am Anfang des Skripts. Dies erleichtert auch die Wartung der betreffenden Programme, da Sie ggf. Konstanten leicht im Programmkopf anpassen können.

Variablennamen

JScript ist eine case-sensitive-Programmiersprache, d. h. bei der Benennung von Variablen wird die Groß-/Kleinschreibung im Variablennamen unterschieden. Eine Variable *Born* ist daher nicht mit dem Namen *born* gleichzusetzen. Deshalb müssen Sie sich bei der Vergabe eines Variablennamens und bei dessen Benutzung peinlich genau an die Schreibweise halten. Zusätzlich sind die folgenden Regeln bei der Bestimmung des Variablennamens einzuhalten:

- ♦ Das erste Zeichen im Variablennamen muß ein Buchstabe, ein Unterstrich (`_`) oder ein Dollarzeichen (`$`) sein. Der Name *Born12* ist zulässig, während *123* keine gültige Variable darstellt, da sich dieser Name nicht von einer Zahl unterscheiden läßt.
- ♦ Die Folgezeichen im Variablennamen können Buchstaben, Ziffern, der Unterstrich oder das Dollarzeichen sein. Der Name darf aber keine Leerzeichen und keine Sonderzeichen wie Umlaute enthalten (z. B. enthält die Anweisung *Straßenname und Hausnummer* gleich zwei Fehler: mehrere Leerzeichen und das Sonderzeichen *ß*). Die Anweisung *Strassenname_und_Hausnummer= "Haferpfad 19"* definierte dagegen eine gültige Variable).

In JavaScript gilt, dass ein Name nicht länger als 32 Zeichen sein sollte. In JScript darf ein Variablenname beliebig lang werden. Aus naheliegenden Gründen sollten Sie aber Namen zwischen 8 und 15 Zeichen bevorzugen. Dies spart vor allem Tipparbeit und reduziert die Wahrscheinlichkeit für Schreibfehler.

Lassen Sie sich auch von der Regel für Programmierer leiten, die besagt, dass Sie sinnvolle Namen für die Variablen wählen sollten. Können Sie sich noch nach einem halben Jahr erinnern, was die Variable *X1* für einen Sinn hatte? Ach ja, es gibt noch eine Einschränkung hinsichtlich Ihrer Kreativität bei der Namensfindung: der Name für eine Variable darf nicht mit den reservierten JScript-Schlüsselwörtern übereinstimmen. Die folgende Tabelle enthält eine Liste reservierter Schlüsselwörter. Beim Vergleich mit der JScript-Hilfe werden Sie einige zusätzliche Schlüsselwörter finden. Diese sind JavaScript entlehnt. Es kann nicht schaden, wenn Sie diese Schlüsselwörter ebenfalls meiden.

Tabelle 4.1:
Reservierte
Schlüsselwörter in
JavaScript

abstract	enum	int	super
boolean	export	interface	switch
break	extends	long	synchronize
byte	false	native	this
case	final	new	throw
catch	finally	null	throws
char	float	package	transient
class	for	private	true
const	function	protected	typeof
continue	goto	public	try
debugger	if	return	var
default	implements	short	void
delete	import	static	while
do	in		with
double	instanceof		
else			

Die Namen *_pagecount* oder *Part9* sind zulässig, während *19Jahr* unzulässig ist, da der erste Buchstabe eine Ziffer enthält. Sobald Sie eine Variable ohne weitere Wertzuweisung deklarieren, wird diese zwar vom Interpreter eingerichtet, der Wert ist auf *undefined* gesetzt. Dies kann zu Problemen führen, wenn Sie eine solche Variable weiterverwenden:

```
var Faktor; // Wert noch undefiniert
var Preis = 100 * Faktor; // Preis wird "auf NaN" gesetzt
```

Die beiden obigen Anweisungen verdeutlichen diesen Sachverhalt. Da die erste Variable *Faktor* undefiniert ist, weist der Interpreter der Variable *Preis* den Wert *NaN* zu. *NaN* steht hier für die Abkürzung »Not a Number«. Möchten Sie der Variablen bereits bei der Instantiierung einen Wert zuweisen, läßt sich dies in der Variablendeklaration vornehmen. Sie können dabei den speziellen Wert *null* oder einen anderen Wert zuweisen:

```
var fact1 = null; // spezieller Wert
var note = 3 * fact1; // Wert wird auf 0 gesetzt
```

Die Möglichkeit zum impliziten Deklarieren von Variablen bietet eine Fehlermöglichkeit. Sie können zwar eine Variable ohne *var* deklarieren und

dann einen Wert zuweisen. Tritt eine neue Variable jedoch erstmals auf der rechten Seite einer Zuweisung auf, löst dies eine Fehlermeldung aus:

```
Name = ""; // Die Variable wird implizit definiert
var aMess = Name + vorname; //
```

Die zweite Zeile führt zu einem Laufzeitfehler, da die Variable *vorname* nicht definiert wird.

Hinweise zu Werten und Datentypen

In JScript besitzen die Variablen typischerweise keinen festen Datentyp. Sie geben ja keinen Datentyp bei der Deklaration an. Vielmehr nimmt die Variable den Wert auf und speichert diesen in einem Format, der diesem Wert entspricht. Der JScript-Interpreter nimmt dabei u.U. eine Konvertierung vor. In einigen Fällen können Sie diese automatische Konvertierung der Werte in verschiedenen Formate beeinflussen. Zahlen lassen sich beispielsweise ohne Probleme in Zeichenketten einbetten (z. B. "Zeichen" + 99). Bei der Zuweisung eine Zeichenkette der Art "99" zu einer numerischen Variablen müssen Sie aber auf Konvertierfunktionen (z. B. *parseInt()* und *parseFloat()*) zurückgreifen.

Die folgende Codesequenz demonstriert diese automatische Typkonvertierung, indem einer Zeichenkette numerische Variablen zugewiesen werden:

```
var Von = 1;
var Bis = 10;
var Aktion = "Zähle von ";
Aktion += Von + " bis " + Bis + ".";
```

Wird dieser Code ausgeführt, enthält die Variable *Aktion* den Inhalt die Zeichenkette "Zähle von 1 bis 10.". Die numerischen Werte werden automatisch in Zeichenketten umgewandelt. Die folgende Anweisungsfolge weist der Variablen *x* einen Wert zu:

```
var x = 0;
x += 1 + "10";
```

Führen Sie diese Anweisungen aus, enthält *x* den Wert "0110". Die Anweisung stammt aus der JScript-Hilfe, ist äußerst trickreich und sollte nicht angewandt werden. Die Anweisung *1 + "10"* auf der rechten Seite der Anweisung bewirkt die Addition einer numerischen Zahl zu einer Zeichenkette. Der JScript-Interpreter wandelt alles in eine Zeichenkette der Form "110" um. Nun muß diese Zeichenkette der Variablen *x* zugewiesen werden. Der Zuweisungsoperator enthält jedoch ein vorangestelltes *+*-Zeichen, was eine Addition zum ursprünglichen Wert von *x* erzwingt. Die Variable *x* enthält jedoch den numerischen Wert 0. Um die Zeichenfolge der

rechten Seite zur Variablen hinzuzufügen, muß deren Wert ebenfalls in eine Zeichenkette "0" konvertiert werden. Der +=-Operator bewirkt die Zusammenführung des Werts 0 und der Zeichenkette "110" zum Endergebnis "0110".

Hinweis

Die obigen Erläuterungen zeigen, was sich mit der Sprache JScript so alles anstellen läßt. C-Programmierer werden sich daher sofort heimisch fühlen. Persönlich möchte ich Ihnen aber den Verzicht auf solche Tricks nahelegen. Verwenden Sie einen klaren Programmierstil. Die halbe Sekunde, die Sie beim Eintippen solcher »gewagten« Konstruktionen sparen, geht mit Sicherheit bei der nachfolgenden Fehlersuche drauf.

Hinweise zu Untertypen

JScript unterscheidet bei Konstanten und Variablen nur sehr wenige Unterdatentypen:

- ◆ *Numerische Zahlen:* Sie können Konstanten wie 423 oder 3.14159 direkt angeben. Als Dezimalpunkt wird kein Komma verwendet, sondern ein Punkt. JScript unterscheidet bei Variablen im Datentyp nicht zwischen Gleitkomma- (Real) und Ganzzahlwerten (Integer). Konstanten lassen sich in verschiedenen Zahlensystemen angeben. Sind die Zeichen 0x (oder 0X) vorangestellt, signalisiert diese eine Hexadezimalzahl (dann sind die Ziffern 0 bis F zulässig). Beginnt eine Zahl mit einer 0 (Null), weist dies auf eine Oktalzahl hin (die Zahl darf nur die Ziffern 0 bis 7 enthalten). Dezimalzahlen umfassen die Ziffern 0 bis 9. Gleitkommazahlen sind durch einen Dezimalpunkt auszuweisen. Zusätzlich läßt sich ein Exponent durch ein vorangestelltes »e« oder »E« ausdrücken (z. B. 12.30, 10.0E20 oder 20E-10). Zusätzlich dürfen Sie die Vorzeichen »+« oder »-« angeben.
- ◆ *Logische (Boolean) Werte:* Hier wird einer Variablen die Konstante *true* oder *false* zugewiesen. Zusätzlich liefern natürlich auch die Ergebnisse einer Auswertung (z. B. ein Vergleich) logische Werte.
- ◆ *Zeichenketten (Strings):* Zeichenketten werden direkt durch Konstanten wie "Dies ist ein Text" oder '1234' einer Variablen dieses Datentyps zugewiesen. Zeichenketten werden in JScript in Hochkommas ' oder Anführungszeichen " eingeschlossen.
- ◆ *Null:* Dies ist ein spezieller Wert, der einer Variablen zugewiesen wird, die keinen Wert besitzt.

Mit diesen wenigen Datentypen müssen und können Sie in JScript-Programmen auskommen.

Spezielle Zeichen in Zeichenketten

Verwenden Sie Zeichenketten, sollten Sie noch einige Spezialzeichen kennen, die in JScript für besondere Funktionen reserviert sind. Die Zeichen steuern die Ausgabe von Texten auf Ausgabemedien (z. B. Dialogfeldern).

\b	Backspace
\f	Form Feed (Zeilenvorschub)
\n	New Line (neue Zeile)
\r	Carriage Return
\t	Tab-Zeichen
\'	Hochkomma
\"	Anführungszeichen
\\	Backslash

Das Backspace-Zeichen werden Sie wohl seltener verwenden. Die Zeichen für einen Zeilenumbruch (\n\r) sind aber zur Formatierung von Texten in Dialogfeldern recht hilfreich.

Ein besonderer Fall liegt vor, wenn Sie in einer Zeichenkette Anführungszeichen oder Hochkommata verwenden müssen. Die Anweisung:

```
Preis = "Er sagte: "Windows 98 WSH ist Klasse!""
```

können Sie so nicht anwenden, da Sie anderenfalls eine Fehlermeldung erhalten. Der Übersetzer findet zwei Zeichenketten und eine Konstante, die er nicht interpretieren kann. In diesem Fall sind die Anführungszeichen " oder Hochkommata ' durch ein vorangestelltes Backslash-Zeichen (\ " bzw. \ ') zu kennzeichnen. Mit der Anweisung:

```
Preis = "Er sagte: \"Windows 98 WSH ist Klasse!\""
```

erkennt der Übersetzer, was Sie vorhaben und übernimmt die ausgezeichneten \"-Zeichen als Textkonstanten in der Zeichenkette.

Ausdrücke und Operatoren

JScript erlaubt innerhalb einer Anweisung die Verwendung von Ausdrücken und Operatoren. Der nachfolgende Abschnitt enthält eine kurze Übersicht über die Möglichkeiten in JScript.

Zuweisungsoperator

Den Zuweisungsoperator (=) haben Sie bereits weiter oben bei der Definition von Variablen kennengelernt. Mit der Anweisung:

```
var MwSt = 17;
```

definieren Sie eine Variable und weisen dieser den Wert *17* mit dem Zuweisungsoperator zu. Weiter unten wird gezeigt, wie Sie zusätzliche Operatoren mit dem Zuweisungsoperator kombinieren können.

Vergleichsoperatoren

Bei *If*-Anweisungen sind beispielsweise Vergleichsoperatoren erforderlich, um zwei Werte zu überprüfen. Die Vergleichsoperatoren liefern einen Wahrheitswert (*true*, *false*) zurück. JScript unterstützt die folgenden Vergleichsoperatoren:

==	gleich
!=	ungleich
>=	größer gleich
<=	kleiner gleich
<	kleiner
>	größer

Die folgende Anweisung zeigt den Einsatz eines solchen Operators:

```
if (MwSt == 17) res1 = 1;
```

Die Anweisung setzt den Wert der Variablen *res1* auf 1, wenn die Variable *MwSt* den Wert 17 enthält.

Hinweis

Bei den Operatoren == bzw. != wird ggf. beim Vergleich zweier Werte eine automatische Typkonvertierung vorgenommen. Mit den beiden Operatoren === und !== führt JScript ebenfalls diese Vergleiche (auf gleich bzw. ungleich) aus, verzichtet aber auf eine Typkonvertierung.

Berechnungsoperatoren

Bei numerischen Berechnungen kommen Berechnungsoperatoren zum Einsatz. Der einfachste Berechnungsoperator bildet die Addition, die Sie bereits kennengelernt haben. Nachfolgend finden Sie einige weiterer Beispiele solcher Berechnungsoperatoren:

```
var preis = 10 + 1;  
brutto = netto * MwSt;  
netto = brutto - MwSt;  
var res = 100 / 25;
```

In diesen Beispielen kommen die Berechnungsoperatoren für die Grundrechenarten +, -, * und / zum Einsatz. JScript verwendet die üblichen

Regeln zur Priorisierung der Operatoren (d. h. Punktrechnung geht vor Strichrechnung). Die Sprache kennt folgende Berechnungsoperatoren:

+	Addition ($a = a + b$)
-	Subtraktion ($a = a - b$)
*	Multiplikation ($a = a * b$)
/	Division ($a = a / b$)
%	Modulo-Division ($a = a \% b$)


Diese Berechnungsoperatoren können Sie (wie in der Sprache C) mit dem Zuweisungsoperator = kombinieren (z. B. +=). Beispiele finden Sie auf den folgenden Seiten.

Bei Zeichenketten können Sie für eine Verknüpfung (Konkatenation) ^{Hinweis} mehrerer Teilstrings den Plusoperator verwenden (z. B. *var name = "Günter" + "Born"*).

Inkrement- und Dekrement-Operatoren

Zum Erhöhen (Inkrementieren) oder Erniedrigen (Dekrementieren) numerischer Variablenwerte sind in JScript die Inkrementations- und Dekrementations-Operatoren zu verwenden.

++i	inkrementiert i
--i	dekrementiert i
*=	Multiplikations-Inkrementation
/=	Divisions-Inkrementation
%=	Modulodivision-Inkrementation

Diese Anweisungsform ist für viele Programmierer, die von Pascal oder Basic kommen, recht ungewohnt. Da sie Tipparbeit spart, lieben insbesondere C-Programmierer diese Schreibweise. Um transparentere Programme zu erstellen, können Sie aber die gewohnten Berechnungsoperatoren ( siehe den vorhergehenden Abschnitt) in Ihren Anweisungen verwenden. Die folgenden Anweisungen verdeutlichen diesen Ansatz:

a += b wirkt wie die Angabe *a = a + b*

a -= b wirkt wie die Angabe *a = a - b*

*a *= b* wirkt wie die Angabe *a = a * b*

a /= b wirkt wie die Angabe *a = a / b*

a %= b wirkt wie die Angabe *a = a % b* (Modulo-Division)

a = ++i erhöht i um 1 und weist den Wert a zu

a = i++ weist a den Wert i zu und erhöht i anschließend um 1

`a = --i` erniedrigt `i` um 1 und weist den Wert `a` zu

`a = i--` weist `a` den Wert `i` zu und erniedrigt `i` um 1

Lediglich in Schleifen etc. müssen Sie die obigen Inkrement-/Dekrement-Operatoren benutzen.

Hinweis

Beachten Sie die Schreibweise (z. B. des Inkrement-Operators). Die Stellung des Operators bestimmt, wie JScript die Variable handhabt. Mit `a = ++b;` steht der Operator vor der zuzuweisenden Variablen. Der Interpreter findet daher erst den `++`-Operator und wendet diesen auf die Variable `b` an. Deren Wert wird erhöht und dann der Variablen `a` zugewiesen. Bei der Anweisung `a = b++;` erfolgt erst die Zuweisung des Wertes `b` an `a`. Dann wird `b` erhöht.

Logische Operatoren

Bei komplexeren Auswertungen sind logische Operatoren gemäß den folgenden JScript-Operatoren erforderlich:

<code>&&</code>	<i>Und</i> -Verknüpfung
<code> </code>	<i>Oder</i> -Verknüpfung
<code>>></code>	Bits nach rechts verschieben
<code><<</code>	Bits nach links verschieben
<code>>>></code>	unsigned bitshift right
<code>!</code>	Logisches <i>Not</i>
<code>~</code>	Bitweises <i>Not</i>
<code>&</code>	Bitmaske mit <i>Und</i> -Bedingung
<code> </code>	Bitmaske mit <i>Oder</i> -Bedingung
<code>^</code>	Bitmaske mit exklusiver <i>Oder</i> -Bedingung (<i>Xor</i>)

Hinweis

Eine Übersicht über die Wirkung der Bitoperatoren *And*, *Or* oder *Xor* finden Sie in [Kapitel 3](#). Beim Einsatz der auf den vorhergehenden Seiten aufgeführten Operatoren gelten bestimmte Prioritäten. Diese kommen zur Anwendung, sobald eine Anweisung mehrere nicht geklammerte Operatoren enthält. Die **Tabelle 4.2** gibt die Prioritäten wieder. Die obersten Einträge der Tabelle besitzen die niedrigste Priorität.

Tabelle 4.2:
Reihenfolge der
Prioritäten bei
Operatoren

Operatoren
Komma ,

Zuweisungen: = += -= *= /= %= <<= >>= >>>= &= ^= =
Bedingung ? :
logisches Oder
logisches Und &&
bitweises Oder
bitweises Xor ^
bitweises Und &
gleich,ungleich == !=
relational < <= > >=
bitweise shift << >> >>>
Addition/Subtraktion + -
Multiplikation/Division * / %
Negation/Inkrement ! ~ - ++ --
Call, Member () [] .

Kontrollstrukturen

In den seltensten Fällen werden Ihre Skriptprogramme einen linearen Ablauf aufweisen. In der Regel müssen Sie Kontrollstrukturen zur Programmsteuerung verwenden. Nachfolgend finden Sie die wichtigsten JScript-Kontrollstrukturen beschrieben.

if-Anweisung

Eine *if*-Anweisung lässt sich in zwei verschiedenen Formen formulieren. Mit:

```
if (Bedingung)
{
    Befehle bei erfüllter Bedingung
}
```

realisieren Sie eine einfache Abfrage. Die Bedingung ist hinter *if* in runden Klammern anzugeben und setzt sich aus den weiter oben erwähnten Vergleichsoperatoren zusammen. Trifft die Bedingung zu (z. B. *MwSt* ==

17), führt das Skript die im folgenden Ja-Zweig aufgeführte(n) Anweisung(en) aus. Handelt es sich nur um eine Anweisung, kann diese direkt in der nächsten Zeile folgen. Bei mehreren Anweisungen müssen Sie diese mit den Zeichen { } zu einem Block zusammenfassen. Die folgenden Zeilen verdeutlichen die Anwendung:

```
if (MwSt <= 16.0)
{
    WScript.Echo ("Achtung! Reduzierter MwSt-Satz");
    MwSt = MwSt_red
}
```

In dieser Struktur wird der Variablen *MwSt* ein neuer Wert zugewiesen, sofern der aktuelle Wert kleiner gleich 16 ist. In diesem Beispiel wurden die Klammern { } zur Blockbildung benutzt. Die letzte Zeile im Block muß übrigens nicht durch ein Semikolon abgeschlossen werden, da das Blockende das Anweisungsende definiert.

Möchten Sie sowohl auf eine zutreffende als auch auf eine nicht zutreffende Bedingung gleichermaßen reagieren? Dann setzen Sie die *if .. else*-Variante ein. Diese besitzt die folgender Syntax:

```
if (Bedingung)
{
    Befehle bei erfüllter Bedingung
}
else
{
    Befehle bei nicht erfüllter Bedingung
}
```

Die Abfrage prüft die Bedingung und führt dann – abhängig vom Ergebnis – die Anweisungen im Ja- oder Nein-Zweig aus. Die folgende Sequenz zeigt den Einsatz dieses Befehls:

```
if (MwSt <= 16.0)
{
    WScript.Echo ("Achtung! Reduzierter MwSt-Satz");
    MwSt = MwSt_red
}
else
{
    WScript.Echo ("Achtung! Normaler MwSt-Satz");
}
```

Tip

Eine der häufigsten Fehlerquellen ist die falsche Schachtelung von *if*-Anweisungen. Folgt nur ein Befehl, können Sie auf die Klammern { } verzichten. Die letzte Anweisung in einem Block benötigt auch kein Semikolon. Persönlich habe ich mir aber angewöhnt, diese nicht benötigten

Strukturelemente trotzdem einzutippen (da diese nicht schaden). Dies macht die Struktur deutlicher und verhindert, dass falsche Anweisungen unbeabsichtigt aus dem Block »herausfallen«.

Conditional-Operator

JScript unterstützt einen Conditional-Operator, der eine Zuweisung in Abhängigkeit von einem bestimmten Wert ermöglicht. Der Operator besitzt folgende Form:

```
(Bedingung) ? Wert1 : Wert2
```

In der Klammer wird eine Bedingung angegeben. Hinter der Klammer folgt ein Fragezeichen sowie zwei durch einen Doppelpunkt getrennte Werte. Trifft die Bedingung zu, wird *Wert1* zugewiesen. Andernfalls ist *Wert2* zu verwenden. Die folgende Anweisung nutzt diesen Operator:

```
Status = (Alter >=18) ? "Erwachsener" : "Kind";
```

Ist der Wert von *Alter* größer gleich 18, wird die Zeichenkette *Erwachsener* in die Variable *Status* geschrieben. Andernfalls wird die Zeichenkette *Kind* verwendet.

for-Schleife

for-Schleifen erlauben Ihnen, bestimmte Anweisungsfolgen auf definierte Weise zu wiederholen. Die Zahl der Wiederholungen wird durch einen Zähler angegeben. Die *for*-Schleife besitzt folgende feste Syntax:

```
for(var count = 1; count <= 100; count++)  
{  
    Anweisungen  
}
```

Die Variable *count* am Schleifenanfang wird beim Eintritt auf den Startwert gesetzt. Das Schlüsselwort *var* veranlaßt, dass die Variable lokal definiert wird. Bei jedem Schleifendurchlauf wird der Zähler *count* im dritten Parameter erhöht (*count++* erhöht um 1) oder erniedrigt (*count--* erniedrigt um 1). Die Endebedingung wird im zweiten Parameter (*count <= 100*) definiert. Das folgende Listing zeigt den Einsatz einer solchen Schleife in einem WSH-Skript.

```
//*****  
// File:   WSHDemo.js (WSH-Beispiel in JScript)  
// Autor:  (c) Günter Born  
//
```

Listing 4.2:
*Ausgabe in einer
for-Schleife*

```

// Zweck: Zeige, wie sich der Programmablauf mit
// Meldungen verfolgen lässt.
//*****
// Die folgenden Zeilen schalten die Ausgabe der
// Testmeldungen ein oder aus. Sie müssen nur den
// Kommentar in die entsprechende Zeile setzen.

// var DebugFlag = false; // Testausgabe abgeschaltet
var DebugFlag = true; // Testausgabe zulassen

j = 0;
debug ("Start", 0, 0);

for (var i = 1; i <= 10; i++) // versuche 10 Durchläufe
{
    debug ("Durchlauf: ", i, j);
    j = j + i; // Addiere alle Zahlen
}

debug ("Ende", i, j);

WScript.Echo ("Ergebnis: ", j);

WScript.Quit ();

function debug (text, count, val)
{
    if (DebugFlag) // Debug-Modus aktiviert?
        WScript.Echo (text, i , "Teilergebnis: ", j)
}
//*****
//*** Ende -> WSH-JScript ***
//*****

```

Hinweis

Sie finden die Beispieldatei *WSHDemo.js* im Ordner *\Beisp\Kap04* auf der Begleit-CD-ROM.

for ... in-Schleife

Zum Zugriff auf Objekte oder Feldelemente erlaubt JScript zusätzlich eine *for*-Schleife mit dem Schlüsselwort *in* gemäß der folgenden Syntax:

```

for (variable in [object | array])
{

```

```
Anweisungen  
}
```

Hinter dem Schlüsselwort *in* läßt sich entweder ein Objekt oder ein Feld angeben. Die *for*-Schleife sorgt dann dafür, dass alle Elemente des Feldes oder der Auflistung (Collection) in die Variable *variable* übergeben werden. Sobald alle Elemente bearbeitet wurden, terminiert die Schleife.

while-Schleife

while-Schleifen ermöglichen es, JScript-Anweisungen mehrfach zu wiederholen, bis die Schleifenbedingung nicht mehr erfüllt ist. Hierbei gilt folgende Syntax:

```
while (Bedingung)  
{  
    Anweisungen  
}
```

Die Bedingung wird bei jedem Durchlauf im Schleifenkopf geprüft. Ist die Bedingung erfüllt (*true*), werden die Anweisungen im { }-Block ausgeführt. Ähnlich wie bei der *for*-Schleife können Sie aber bei einer Folgeanweisung auf diese Klammern verzichten. Die folgende Sequenz zeigt den Einsatz dieses Schleifenbefehls:

```
var i = 0;  
  
while (i <= 10)    // Versuche 10 Durchläufe  
{  
    WScript.Echo ("Durchlauf: ", i);  
    i++;           // Addiere alle Zahlen  
}
```

Das Programm durchläuft die Schleife so lange, bis der Index auf 11 gesetzt ist. Bei jedem Durchlauf wird der Index in einem Meldungsfeld angezeigt.

do ... while-Schleife

Die *do ... while*-Schleifen verhalten sich ähnlich wie *while*-Schleifen und ermöglichen es, JScript-Anweisungen mehrfach zu wiederholen, bis die Schleifenbedingung nicht mehr erfüllt ist. Hierbei gilt folgende Syntax:

```
do  
{  
    Anweisungen
```

```

}
while (Bedingung);

```

Im Gegensatz zur *while*-Schleife wird die Schleifenbedingung aber am Schleifenende abgeprüft, d. h. die Schleife wird zwangsweise einmal durchlaufen. Ist die Bedingung erfüllt (*true*), werden die Anweisungen im { }-Block erneut ausgeführt. Ähnlich wie bei der *for*-Schleife können Sie aber bei einer Folgeanweisung auf diese Klammern verzichten. Die folgende Sequenz zeigt den Einsatz dieses Schleifenbefehls. Die folgende Codesequenz stammt aus der JScript-Hilfe und benutzt eine *do ... while*-Schleife, um alle Laufwerke einer *Drives*-Auflistung zu bearbeiten:

```

function GetDriveList()
{
    var fso, s, n, e, x;
    fso = new ActiveXObject("Scripting.FileSystemObject");
    e = new Enumerator(fso.Drives);
    s = "";
    do
    {
        x = e.item();
        s = s + x.DriveLetter;
        s += " - ";
        if (x.DriveType == 3)
            n = x.ShareName;
        else if (x.IsReady)
            n = x.VolumeName;
        else
            n = "[Laufwerk nicht bereit]";
        s += n + "<br>";
        e.moveNext();
    }
    while (!e.atEnd());
    return(s);
}

```

***switch*-Anweisung**

Die *switch*-Anweisung erlaubt Ihnen, einzelne Blöcke in Abhängigkeit von einem Wert auszuführen. Die Anweisung besitzt folgende Syntax:

```

switch (Ausdruck)
{
    case label :

```

```

    Anweisungen
case label :
    Anweisungen
...
default :
Anweisungen
}

```

Am Beginn der Anweisung wird die Bedingung formuliert. Je nach Wert dieser Bedingung wird dann einer der *case*-Zweige ausgeführt. Der Bezeichner *label* steht dabei als Platzhalter für den Wert des Ausdrucks. Entspricht der Wert des Bezeichners dem Ausdruck, werden die Anweisungen des Blocks ausgeführt. Trifft keine Bedingung zu, werden die Anweisungen des *default*-Zweigs ausgeführt. Die folgende Sequenz zeigt den Einsatz der *switch*-Anweisung:

```

function Test(x)
{
    switch (x){
        case 1:
            ...
        case 2:
            ...
        case 3:
            ...
        default:
            ...
    }
}

```

In diesem Beispiel werden die Zweige in Abhängigkeit vom Wert *x* durchlaufen. Bei *x = 1* wird z. B. der Zweig *case 1:* bearbeitet.

***break-* und *continue*-Anweisungen**

In Microsoft JScript ist es mit der *break*-Anweisung möglich, eine Schleife vorzeitig zu unterbrechen. Sobald der Interpreter in einer Schleife auf eine solche Anweisung stößt, wird die Schleife beendet und das Skript hinter dem Schleifenende fortgesetzt.

Genau entgegengesetzt arbeitet der *continue*-Befehl. Sobald der Interpreter diesen Befehl erkennt, werden die restlichen Anweisungen im Block übersprungen. Gleichzeitig wird der Schleifenindex (z. B. bei *for*-Schleifen) erhöht. Anschließend erfolgt ein neuer Schleifendurchlauf. Mit der *continue*-Anweisungen erreichen Sie also, dass der Programmablauf zum Schleifenanfang verzweigt.

Built-in-Objekte und -Funktionen

In JScript sind bereits einige Objekte (String-Objekt, Math-Objekt, Date-Objekt) und Funktionen integriert. Nachfolgend finden Sie einige Hinweise zu diesen Themen.

Funktionen

Funktionen kombinieren mehrere Operationen unter einem Namen und lassen sich in einem Programm direkt unter ihrem Namen aufrufen. Beim Beenden liefert die Funktion einen Ergebniswert zurück. JScript unterstützt dabei sowohl benutzerdefinierte Funktionen als auch interne (Built-in-Funktionen). Benutzerspezifische Funktionen werden in JScript durch einen Namen gemäß folgender Form definiert:

```
function test (Argumente)
{
    Funktionsrumpf mit Anweisungen
    return;
}
```

Die Funktion wird durch das Schlüsselwort *function*, den Funktionsnamen sowie die Argumentliste eingeleitet. Die Anweisungen im Funktionsrumpf sind durch geschweifte Klammern { } einzufassen. Die *return*-Anweisung beendet die Ausführung der Funktion.

Diese Funktion läßt sich dann innerhalb des Skripts durch Angabe des Funktionsnamens und der Parameter aufrufen. Die folgende Anweisung aktiviert beispielsweise die Funktion *test*:

```
result = test ();
```

In der Funktionsdefinition lassen sich Parameter (Argumente) angeben. Dann müssen Sie beim Aufruf der Funktion diese Parameter in den Klammern übergeben. Die einzelnen Parameter werden durch Kommata getrennt. Besitzt eine Funktion keine Argumente, ist beim Aufruf eine leere Klammer an den Funktionsnamen anzuhängen.

Built-in-Funktionen

Die Sprachdefinition von JScript besitzt bereits einige Built-in-Funktionen, mit denen Sie Ausdrücke handhaben, spezielle Zeichen behandeln oder Zeichenketten und numerische Werte zueinander konvertieren können.

Die *eval*-Funktion wertet beispielsweise eine Zeichenkette als Argument aus und liefert das Ergebnis zurück (z. B. *Wert = eval("14+15");*). *parseFloat*

wertet ein Argument aus und versucht eine Gleitkommazahl zurückzugeben. Kommt ein ungültiges Zeichen im String vor (nicht +, - oder 0 bis 9, . oder e), wird der String lediglich bis zu diesem Zeichen konvertiert. Handelt es sich nicht um eine Zahl, wird *NaN* (Not a Number) zurückgeliefert. Die Funktion *parseInt* erwartet als erstes Argument einen String und im zweiten Argument eine Zahlenbasis (10=dezimal, 8 = oktal, 16 = hexadezimal etc.) Bei fehlerhaften Strings wird ebenfalls *NaN* zurückgegeben. *parseInt* liefert immer Integer-Werte zurück.

Eine detaillierte Auflistung der Built-in-Funktionen finden Sie in der JScript- [Hinweis Sprachreferenz](#).

Objekte

Neben den Funktionen unterstützt JScript verschiedene Objekte, um direkt Operationen auf Zeichenketten auszuführen, um mathematische Operationen auszuführen oder um Datums- und Zeitwerte zu manipulieren. Bei diesen JScript-Objekten handelt es sich um Auflistungen von Eigenschaften und Methoden:

- ◆ Das *String*-Objekt wird benutzt, sobald Sie einer Variablen oder einer Eigenschaft eine Zeichenkette zuweisen (z. B. *name* = "*Born*";). Das Objekt besitzt verschiedene Methoden, um die Zeichenkette zu manipulieren.
- ◆ Das *Math*-Objekt stellt Eigenschaften und Methoden zur Verfügung, um mathematische Operationen und Funktionen auf Ausdrücken auszuführen (z. B. *wert* = *Math.PI*; weist die Eigenschaft *Pi* einer Variablen zu).
- ◆ Das *Date*-Objekt erlaubt Ihnen mit Datums- und Zeitangaben zu arbeiten (z. B. *varName* = *new Date(parameters)*; erzeugt ein neues Datumsobjekt, *today* = *new Date()*; liefert das Datum).

In JScript werden Objekte und Felder in der gleichen Weise behandelt. Sie können auf Objekte einer Auflistung oder auf Elemente eines Feldes auf die gleiche Weise zugreifen. Möchten Sie auf Methoden oder Eigenschaften eines Objekts zugreifen, geben Sie den Objektnamen gefolgt von einem Punkt und dem Namen der Eigenschaft oder der Methode an. Die Anweisung:

```
WScript.Echo ("Hallo");
```

Benutzt die Methode *Echo* des Objekts *WScript*, um einen Text auszugeben. Alternativ lassen sich Indexwerte zum Zugriff auf einzelne Objekte einer Auflistung oder auf Elemente eines Feldes verwenden. Die folgenden Anweisungen sind gleichwertig:

```
Res = Object.width;
```

```
Res = Object[3]; // [3] entspricht hier dem Index "width"
```

```
Res = Object["width"];
```

Während die Klammern beim Zugriff auf den numerischen Index zulässig sind, darf der Punkt bei der Angabe eines Indexwerts nicht angegeben werden. Die folgende Anweisung führt daher zu einem Übersetzungsfehler:

```
Res = Object.3;
```

Besitzt ein Objekt ein weiteres Objekt als Eigenschaft, werden die Konventionen zur Benennung etwas erweitert:

```
var x4 = toDoToday.shoppingList[3].substring(0,1);
```

An die Eigenschaft des Objekts wird einfach ein Punkt angehängt. Daran schließt sich dann das (Unter-) Objekt an. Felder lassen sich in JScript ebenfalls sehr einfach definieren und handhaben. Die folgende Anweisung vereinbart ein Feld:

```
var name = new Array(17);
```

Hier werden die Feldelemente von 0 bis 16 erzeugt. Sie können anschließend über:

```
name[0] = "Born";
```

auf das erste Feldelement zugreifen und einen Text hinterlegen. Die Zahl der Feldelemente läßt sich mit:

```
zahl = name.length;
```

ermitteln. Ein mehrdimensionales Feld wird mit der Anweisung:

```
var namex = name [3] [7];
```

vereinbart.

Hinweis

An dieser Stelle möchte ich die Kurzeinführung in JScript beenden. Sie finden die komplette JScript-Sprachreferenz von Microsoft mit den hier nicht erwähnten Erweiterungen als HTML-Dokumentation auf der Begleit-CD-ROM im Ordner *Infos\Jscript*. Starten Sie die EXE-Datei *jsdoc.exe* wird die Dokumentation lokal als Windows 98-Hilfedatei auf dem Rechner installiert. Sie können die Dokumente über das Startmenü aufrufen. Alternativ können Sie mit dem Internet Explorer direkt auf die Version der CD-ROM zugreifen. Laden Sie die Datei *JsTutor.htm* im Ordner *Infos\jscript\jscript*. Sie können anschließend sowohl das Tutorial als auch die Sprachreferenz benutzen. Beachten Sie aber, dass sich diese Sprachreferenz auf die JScript-Implementierung im Internet Explorer bezieht.

Einstieg in die WSH-Skriptprogrammierung

5

Objekte, Eigenschaften und Methoden	131
Dialoge im WSH anzeigen	136

In diesem Kapitel erhalten Sie einen Einstieg in die Skriptprogrammierung für den Windows Scripting Host.

- ♦ Für alle, die sich noch nicht mit der objektorientierten Programmierung befaßt haben, gibt es eine kurze Einführung in diese Thematik. Sie lernen, wie Sie mit Objekten, Methoden und Eigenschaften umgehen können.
- ♦ Sie erstellen einfache Skriptprogramme in VBScript und JScript. Hierbei erfahren Sie, wie sich Benutzermeldungen ausgeben lassen, und lernen dabei auch die verschiedenen Methoden und Funktionen zur Realisierung von Benutzerausgaben kennen.

Bei der Bearbeitung der in diesem Kapitel enthaltenen Beispiele steigen Sie schnell in die Skriptprogrammierung ein.

Objekte, Eigenschaften und Methoden

Bei der Skriptprogrammierung kommen Sie sehr schnell mit Begriffen wie Objekten, Eigenschaften und Methoden in Berührung. Sofern Sie mit diesen Begriffen noch nicht allzuviel anfangen können, möchte ich in den nachfolgenden Abschnitten eine kurze Einführung in diese Thematik geben.

Etwas Theorie ...

Eine Anwendung besteht aus zwei Komponenten: dem Inhalt und der Funktionalität. Nehmen wir beispielsweise ein Textverarbeitungsprogramm wie Microsoft Word. Die Dokumente, Wörter, Zahlen, Diagramme, Formulare, Symbolleisten etc. stellen die Inhalte der Anwendung dar. Die Funktionalität legt fest, was sich mit diesen Inhalten anstellen läßt (z. B. öffnen, schließen, löschen, einfügen, formatieren etc.). Die Inhalte und die zugehörigen Funktionalitäten einer Anwendung lassen sich in diskrete Einzelteile auflösen. Ein Textdokument besteht zum Beispiel aus Absätzen, die sich wiederum aus Wörtern und Buchstaben zusammensetzen. Auf Absätze, Wörter oder Buchstaben lassen sich Funktionen ausführen (z. B. löschen, einfügen, formatieren etc.). Inhalt und Funktionalität eines solchen Einzelteils werden dann als »Objekt« betrachtet. Ein solches Objekt kann zusätzliche Eigenschaften und Objekte aufweisen.

Abbildung 5.1:
*Dialogfeld als
Objekt*



Die obigen Ausführungen sind Ihnen zu abstrakt? Na gut, dann nehmen wir ein anderes Objekt als Beispiel (auf Anwendungsobjekte kommen wir in späteren Kapiteln nochmals zu sprechen). Nehmen wir im nächsten Beispiel ein sichtbares Objekt in Form eines Dialogfelds (**Abbildung 5.1**). Dieses Dialogfeld (auch als Meldungsfeld bezeichnet) besitzt intern Daten und eine Funktion (nämlich die Anzeige der Daten). Folglich läßt sich das Dialogfeld gemäß obiger Definition als ein Objekt auffassen.

Dieses Objekt verfügt nun über bestimmte Eigenschaften. Das Fenster besitzt zum Beispiel eine Position und eine Größe. Die X/Y-Position sowie die Breite und Höhe stellen Eigenschaften dar. Weiterhin besitzt das Dialogfeld einen Titeltext, den Text der anzuzeigenden Meldung sowie eine Hintergrundfarbe. Dies sind alles Eigenschaften, die diesem Objekt zugeordnet sind.

Die Eigenschaften hängen dabei vom jeweiligen Objekt ab. Es muß sich auch nicht unbedingt um sichtbare Eigenschaften handeln. Viele dieser Eigenschaften eines Objekts werden offengelegt und lassen sich aus einem Programm heraus ansprechen (lesen und setzen). Beim Dialogfeld legt das Objekt zwar einige Eigenschaften wie die Hintergrundfarbe selbst fest. Sie können aber beispielsweise den Titeltext angeben.

Zusätzlich kann ein Objekt selbst wieder Objekte enthalten. Das in **Abbildung 5.1** gezeigte Dialogfeld enthält beispielsweise eine *OK*-Schaltfläche sowie ein Symbol, die ihrerseits wiederum Objekte darstellen können. Kennen Sie die Hierarchie dieser Objekte (wird auch als Objekthierarchie oder Objektmodell bezeichnet), können Sie per Programm auf diese Objekte zugreifen. So läßt sich beispielsweise über den Windows Scripting Host auf ein Anwendungsobjekt (z. B. Excel) zugreifen. Dieses Anwendungsobjekt besitzt intern weitere Objekte wie beispielsweise ein geladenes Dokument. Zum Zugriff auf das Dokument muß die Objekthierarchie *Anwendungsobjekt|Dokumentobjekt* benutzt werden. Doch dazu später mehr.

Allerdings beginnt das Beispiel an dieser Stelle etwas zu »hinken«. Hinweis
Dialogfelder stellen zwar Objekte dar. Windows bzw. die Skriptsprachen stellen aber eine Funktion zur Anzeige dieser Dialogfeld-Objekte bereit. Sie können daher nicht explizit auf die im Dialogfeld enthaltenen Objekte wie Schaltflächen zugreifen. In VBScript bekommen Sie gar nicht mit, dass Sie mit einem Objekt arbeiten. In JScript sieht die Sache aber etwas anders aus. Hier wird explizit eine Methode auf einem Objekt angewandt, um das Dialogfeld anzuzeigen.

Der Vorteil des Objektansatzes besteht darin, dass Sie als Entwickler keine Informationen über das Innenleben der benutzten Objekte benötigen. Der Name des Objekts und einige Informationen über die zugehörigen Eigenschaften reichen. Sie legen z. B. lediglich fest, dass Sie das Objekt »Dialogfeld« benötigen, welches zwei Schaltflächen, einen Titel und ein Symbol enthält. Die Position der integrierten Objekte sowie verschiedene Objekteigenschaften wie Größe, Position oder Hintergrundfarbe werden dabei automatisch vom Objekt bestimmt.

Bleibt noch die Frage, was Methoden sind und wozu man diese braucht. Eigentlich könnten Sie die Objekte benutzen und deren Eigenschaften manipulieren. Dies reicht aber nicht immer aus. Manchmal möchte man eine bestimmte Operation auf einem Objekt ausführen. Dies könnte beispielsweise eine Methode zum Anzeigen eines Fensters sein. Oder das Fenster soll verschoben werden. Eine Methode führt immer eine bestimmte Operation auf einem Objekt aus. (Falls Sie bereits konventionell programmiert haben: der Aufruf einer Methode ist so etwas wie ein Funktionsaufruf.) Dadurch lassen sich die Inhalte (sprich: die Eigenschaften) der Objekte verändern, ohne dass Sie wissen müssen, wie dies genau passiert. Ob ein Objekt eine bestimmte Methode unterstützt, hängt vom jeweiligen Objekt ab. Das Objekt eines Standarddialogfelds unterstützt beispielsweise keine Methode zum Verschieben.

... und nun zur Praxis

Kommen wir nun zum Windows Scripting Host zurück. Was hat der WSH mit Objekten zu tun? Und wie lassen sich diese Objekte nutzen? Der Windows Scripting Host stellt selbst verschiedene Objekte für die Skriptprogramme zur Verfügung und erlaubt auch weitere Objekte anderer Anwendungen zu verwenden. Nehmen wir ein ablaufendes Skript im Windows Scripting Host als Beispiel. Es handelt sich hier ebenfalls um ein Objekt, da es sowohl Daten (die Skriptanweisungen) als auch eine Funktion (die Ausführung des Skripts) enthält. Um auf ein Objekt zuzugreifen, muß dessen Objektname im Skript bekannt sein.

Beim WSH wird die Sache ganz einfach. Der Windows Scripting Host stellt selbst direkt das *WScript*-Objekt beim Start eines Skripts zur Verfügung. Dieses Objekt stellt quasi das ablaufende Skript selbst dar. Wenn Sie in der Programmierreferenz des WSH im Anhang nachschlagen, sehen Sie dort, dass das *WScript*-Objekt bestimmte Eigenschaften besitzt. So lassen sich beispielsweise der Name des WSH, dessen Version, der Pfad zur Datei *Cscript.exe* als Eigenschaften betrachten und abfragen.

Hinweis

Beachten Sie, dass Eigenschaften sich in Abhängigkeit vom Objekt lesen und/oder beschreiben lassen. Beim *WScript*-Objekt können Sie die Eigenschaften allerdings nur lesen (es macht ja keinen Sinn, die WSH-Version zu verändern).

Die WSH-Programmierreferenz führt auch einige Methoden auf, die sich auf dem *WScript*-Objekt ausführen lassen. Die elementarste Methode stellt die *Quit*-Methode dar. Diese sorgt dafür, dass das Objekt beendet wird (sprich: das laufende Skript terminiert). Im Programm läßt sich diese Methode folgendermaßen anwenden:

```
WScript.Quit();
```

Als erstes ist der Name des Objekts, auf das die Methode anzuwenden ist, anzugeben. In diesem Beispiel handelt es sich um das Objekt *WScript*, welches automatisch vom WSH bereitgestellt wird (im Verlauf der nächsten Kapitel lernen Sie noch, dass normalerweise bestimmte Befehle zum Erzeugen eines Objekts erforderlich sind). Anschließend kommt der Punkt, der eine Methode oder eine Eigenschaft vom Objektnamen separiert. Hinter dem Objektnamen und dem Punkt folgt nun der Name der Methode *Quit*. Je nach Methode sind zusätzliche Parameter zu übergeben. Die Klammer hinter dem Methodennamen dient dabei zur Aufnahme dieser Parameter. Der *Quit*-Methode werden in diesem Beispiel keine Parameter übergeben, die Klammer bleibt leer.

Hinweis

Sie können der *Quit*-Methode beim Aufruf eine Ganzzahl (z. B. 0 oder 1) als Parameter übergeben. Dieser Parameter bezeichnet den sogenannten *Process-Exit-Code*. Dieser Code liefert einen Hinweis darauf, ob ein Prozeß ohne Fehler beendet wurde. Fehlt der Parameter, liefert die *Quit*-Methode

automatisch den Wert 0 für einen korrekt terminierten Prozeß zurück. Werte größer als 1 werden dagegen per Definition als Fehlercodes interpretiert. Unter Windows wird dieser Code aber nicht ausgewertet. Sofern Sie aber die Skriptdateien von MS-DOS aus über den Host *Cscript.exe* starten, kann der Fehlercode über die *ERRORLEVEL*-Funktion in einer Stapelverarbeitungsdatei ausgewertet werden.

Eine weitere vom Objekt *WScript* bereitgestellte Methode ist die *Echo*-Methode. Mittels dieser Methode können Sie in einem Skript einfache Meldungen ausgeben. Auf diese Technik kommen wir auf den folgenden Seiten noch zu sprechen.

Der WSH kann aber weitere Objekte im Skript zur Verfügung stellen. Um beispielsweise mit den Shell-Objekten des Windows Scripting Host arbeiten zu können, müssen diese zunächst definiert werden. Dies geschieht mit folgender Anweisung:

```
Set WSHShell = WScript.CreateObject ("WScript.Shell")
```

Konkret stellt *WScript* das bereits erwähnte Objekt des Windows Scripting Host dar. Die *CreateObject*-Methode erlaubt jetzt, eine Referenz auf ein anderes Objekt zu erzeugen und in der angegebenen Objektvariablen (*WSHShell*) abzulegen. In obigem Beispiel wird ein Objekt auf die Windows-Shell erzeugt. Über dieses Objekt können Sie beispielsweise Verknüpfungen auf dem Desktop erstellen. Mit der Anweisung:

```
Set oIE4 = CreateObject("InternetExplorer.Application")
```

wird beispielsweise eine Referenz auf das Internet Explorer-Objekt *Application* erzeugt und in der Objektvariablen *oIE4* abgelegt. Mit der Anweisung:

```
Set oIE4 = Nothing
```

wird die Referenz zum Objekt in der Objektvariablen freigegeben.

Mit dieser Einführung möchte ich es bewenden lassen. Im Verlauf der nächsten Seiten und Kapitel erhalten Sie viele Beispiele zum Umgang mit Objekten, Methoden und Eigenschaften. Hinweis

Welche Objekte stehen zur Verfügung?

Beim Erstellen des WSH-Programme steht Ihnen eine riesige Sammlung von Objekten (aus Windows, aus Office, aus ActiveX-Modulen etc.) zur Verfügung. Einzige Voraussetzung ist, dass diese Objekte auf dem lokalen System als COM-Objekte registriert sind. Für Interessierte: Die Objekte müssen in der Registrierung im Zweig *HKEY_CLASSES_ROOT* mit ihrem Class-ID-Code eingetragen sein. In diesem Zweig wird dann auch zur

Identifikation des Moduls dessen Programmname eingetragen. Anhand dieser *ProgID* kann das Skript über die *Set*-Anweisungen die Objekte referenzieren.

Sofern Ihnen die auf dem System installierten Anwendungsobjekte nicht reichen, können Sie sich eigene Objekte erstellen. Mit der auf der Begleit-CD-ROM enthaltenen Visual Basic 5 Control Creation Edition können Sie eigene ActiveX-Objekte erzeugen, die sich aus WSH-Skripten ansprechen lassen.

Wo erhalten Sie Informationen über Objekte und Eigenschaften?

Die Schwierigkeit bei der Erstellung von WSH-Anwendungen besteht darin, die benötigten Informationen über Objekte und deren Eigenschaften sowie die anwendbaren Methoden herauszufinden. Für den Windows Scripting Host stellt Microsoft eine entsprechende Dokumentation zur Verfügung (siehe ↗ Anhang).

Hinweis

Sie finden die von mir lokalisierte Version auf der Begleit-CD-ROM im Ordner *\Infos\Wsh* als HTML-Datei. Eine Dokumentation der Office-Objekte liefert die Hilfe der betreffenden Anwendungsprogramme. Weitere Hinweise zu den Objekten finden Sie auch in den von Microsoft Press veröffentlichten Dokumentationen (z. B. Language Referenzen für Microsoft Office 97). Einige Hinweise zu diesem Thema liefert auch der im Literaturverzeichnis unter /3/ aufgeführte VBA-Titel.

Aktuelle Informationen über die auf Ihrem System registrierten COM-Objekte liefert Ihnen der von Microsoft entwickelte OLE/COM Object Viewer (siehe ↗ Kapitel 2). Weiterhin können Sie über den Objektkatalog der Microsoft-Entwicklungsumgebungen Informationen über die Schnittstellen der Objekte abrufen. Auch dies wurde in ↗ Kapitel 2 bereits erläutert.

Dialoge im WSH anzeigen

Zur Kommunikation mit dem Benutzer werden in WSH-Skripten wohl am häufigsten Dialogfelder mit Texten und Schaltflächen benutzt. Der nachfolgende Abschnitt zeigt Ihnen, welche Methoden in JScript- und VBScript-Skripten zur Verfügung stehen, um Dialoge im WSH anzuzeigen. Dabei greifen Sie auf konkrete Beispiele zurück und lernen die Unterschiede bei der Ausgabe von Dialogfeldern in VBScript und JScript kennen.

Nutzen der Echo-Methode

Die vom Windows Scripting Host zur direkten Anzeige von Benutzermeldungen bereitgestellte *Echo*-Methode wurde bereits erwähnt. Mit dieser Methode erhalten Sie ein einfaches Hilfsmittel, um Meldungen sowohl aus VBScript als auch aus JScript auszugeben. Die *Echo*-Methode besitzt folgende Syntax:

Objekt.Echo (Parameter)

Als *Objekt* kommt in unserem Fall nur der Windows Scripting Host in Frage, der sich unter dem Objektnamen *WScript* ansprechen lässt. Folglich wird die Methode in der Form:

WScript.Echo

im Skript benutzt. Die Methode kann dabei ein oder mehrere Argumente als Parameter übernehmen. Die Parameter stellen dabei die auszugebenden Informationen dar. Wie die Anweisung in den beiden WSH-Skriptsprachen genau aufgebaut ist, wird in den folgenden beiden Beispielen gezeigt.

Die Echo-Methode in VBScript nutzen

Möchten Sie in VBScript auf sehr einfache Weise eine Ausgabe an den Benutzer vornehmen? Die *Echo*-Methode des *WScript*-Objekts bietet genau das Richtige. Das **Listing 5.1** zeigt Ihnen, wie Sie diese Methode verwenden können.

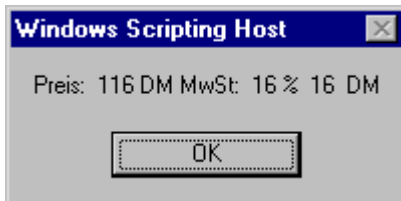


Abbildung 5.2:
Dialogfeld mit den
Ausgaben der
Echo-Methode

```
' *****  
' File:   Echo.vbs (WSH-Beispiel in VBScript)  
' Autor:  Günter Born  
'  
' Zweck:  Demonstriert die Anwendung der Echo-Methode  
' im WSH.  
' *****  
  
Option Explicit  
  
' Lege verschiedene Variable an  
DIM Preis, MwSt, Netto  
  
  
MwSt = 16.0
```

Listing 5.1:
Ausgabe mittels
der Echo-Methode
in VBScript

```

Netto = 100.0
' Berechne den Brutto-Preis

Preis = Netto * (1.0 + MwSt/100.0)

' Jetzt erfolgt die Ausgabe des Ergebnisses.
' Da das WScript-Objekt direkt exponiert wird,
' brauchen wir keine Objekte anzulegen.

WScript.Echo "Preis: ", Preis, "DM MwSt: ", MwSt, _
            "% ", Preis - Netto, " DM"

WScript.Quit

```

```

'*****
'***           Ende -> WSH-VBScript           ***
'*****

```

Um die *Echo*-Methode zu nutzen, benötigen Sie keine großen Vorbereitungen. Da der WSH automatisch das *WScript*-Objekt bereitstellt, genügt die Angabe des Objektname und der Aufruf der Methode:

```

WScript.Echo "Preis: ", Preis, "DM MwSt: ", MwSt, _
            "% ", Preis - Netto, " DM"

```

Hier kommt die typische Notation *Objektname.Methode* zum Einsatz. Beachten Sie, dass die *Echo*-Methode mehrere Parameter haben darf. In VBScript werden die gewünschten Parameter direkt hinter dem Schlüsselwort *Echo* aufgeführt. Ein Komma dient als Separator zwischen den Parametern. Sie können der *Echo*-Methode folglich mehrere auszugebende Werte direkt als Parameter übergeben. Das Ergebnis ist in **Abbildung 5.2** zu sehen. Die einzelnen Teiltexte werden von der Methode in einer Zeile des Dialogfelds ausgegeben.

Hinweis

Sie finden das Beispielprogramm *Echo.vbs* im Ordner *\Beisp\Kap05* auf der Begleit-CD-ROM.

Die Echo-Methode in JScript nutzen

Die *Echo*-Methode stellt in JScript die einzige Möglichkeit dar, mit der Sie ohne größeren Aufwand und ohne weitere Objekte zu erzeugen eine Ausgabe vornehmen können. Auch hier stellt der WSH automatisch das *WScript*-Objekt zur Verfügung. Die Ausgabe der *Echo*-Methode entspricht dem in **Abbildung 5.2** gezeigten Dialogfeld.

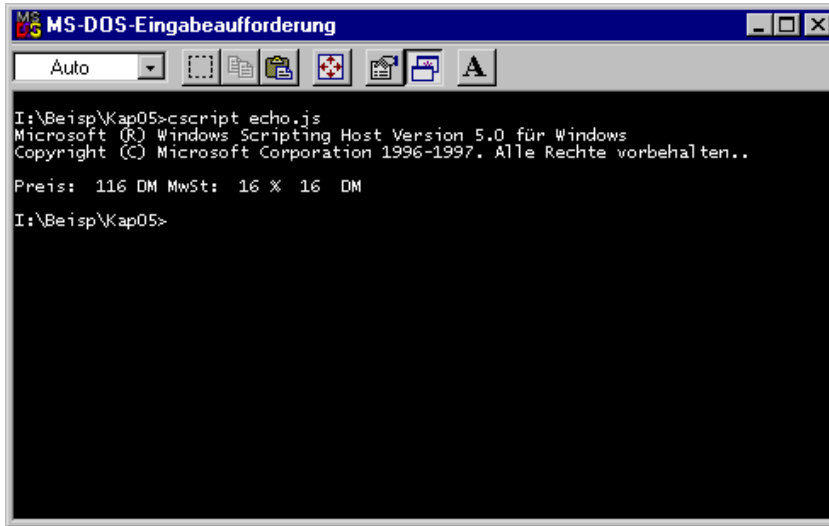


Abbildung 5.3:
Echo-Ausgaben in
MS-DOS

Sofern Sie das Skript nicht über *Wscript.exe* sondern von MS-DOS aus über *Cscript.exe* aufrufen, wird kein Dialogfeld angezeigt. Vielmehr gibt die *Echo*-Methode die Ausgaben in der MS-DOS-Befehlszeile aus (**Abbildung 5.3**). Diese Art der Ausgabe funktioniert sowohl in JScript als auch in VBScript.

Hinweis

Das folgende Listing zeigt, wie die *Echo*-Methode in JScript eingesetzt wird. Im Gegensatz zu VBScript müssen Sie die Parameter in Klammern setzen und den Befehl mit einem Semikolon abschließen.

```
//*****
// File:   Echo.js (WSH-Beispiel in JScript)
// Autor:  Günter Born
//
// Zweck: Demonstriert die Anwendung der Echo-Methode
// im WSH.
//*****
// Lege verschiedene Variable an
var Preis
var MwSt = 16.0
var Netto = 100.0
// Berechne den Brutto-Preis

Preis = Netto * (1.0 + MwSt/100.0)

// Jetzt erfolgt die Ausgabe des Ergebnisses.
// Da das WScript-Objekt direkt exponiert wird,
// brauchen wir keine Objekte anzulegen.
```

Listing 5.2:
Echo-Ausgabe in
JScript

```
WScript.Echo ("Preis: ", Preis, "DM MwSt: ", MwSt,
             "% ", Preis - Netto, " DM");

WScript.Quit();

//*****
//***      Ende -> WSH-JScript      ***
//*****
```

Auch hier kommt das Prinzip zur Anwendung einer Methode klar zum Vorschein. Die obige Anweisung besagt, dass das *WScript*-Objekt zu verwenden ist. Der Punkt *.* trennt Objekt und Methode. Die anzuwendende *Echo*-Methode besitzt als Parameter den Text, der im Dialogfeld einzublenden ist. Dabei können Sie die gewünschten Parameter – getrennt durch Kommata – in den Klammern angeben. Die Methode konvertiert numerische Werte automatisch in Zeichenketten und setzt beispielsweise bestimmte Standardeigenschaften (z. B. den Titeltext sowie die Zahl der Schaltflächen) automatisch für das Dialogfeld. Die Anweisung:

```
WScript.Quit();
```

wendet dagegen die *Quit*-Methode auf das Objekt *WScript* an. Diese Methode sorgt dafür, dass das Objekt beendet wird. Da es sich aber bei dem Objekt um das gerade im Windows Scripting Host auszuführende Skript handelt, wird dieses beendet.

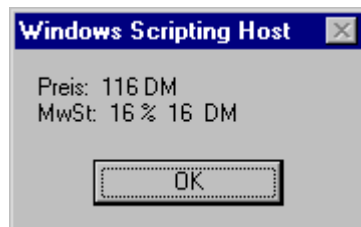
Hinweis

Sie finden die Beispieldatei *Echo.js* im Ordner *\Beisp\Kap05* auf der Begleit-CD-ROM.


So erhalten Sie einen Zeilenumbruch im Dialogfeld

Eine häufig auftretende Frage bei der Ausgabe von Dialogen gilt dem Zeilenumbruch. Sobald Sie den Text übergeben, nimmt das Laufzeitsystem die Formatierung des Textes vor. Ab einer gewissen Länge bricht der WSH den Text selbständig in mehrere Zeilen um.

Abbildung 5.4:
Mehrzeiliger Text
im Dialogfeld



Sie haben aber die Möglichkeit, den Text gezielt in mehrere Zeilen zu umbrechen (**Abbildung 5.4**). Die *Echo*-Methode sowie die in den nachfolgenden Abschnitten vorgestellten Ausgabefunktionen bieten keine Parameter, um einen Zeilenumbruch im Text des Dialogfelds vorzunehmen.

Dass es trotzdem eine Möglichkeit gibt, zeigt **Abbildung 5.4**. Sie müssen aber zu einem Trick greifen: Die auszugebende Zeichenkette muß die Steuercodes für einen Zeilenumbruch aufweisen. Die betreffenden Steuerzeichen besitzen die Codes 10 (CR-Zeichen) und 13 (LF-Zeichen). Um die Zeichen zum Zeilenumbruch in der Zeichenkette einzufügen, müssen Sie hierzu die betreffenden Anweisungen der Sprache kennen. JScript bietet eigens spezielle Steuerzeichen, die sich in die Zeichenketten einfügen lassen (siehe  Abschnitt »Spezielle Zeichen in Zeichenketten« in Kapitel 4). Ein Zeilenumbruch wird durch das Steuerzeichen "\n" ausgelöst. Möchten Sie also, dass die *Echo*-Methode bei der Ausgabe eines Textes einen Zeilenumbruch durchführt, müssen Sie in den Parametern das Steuerzeichen angeben. Die nachfolgende Anweisung nutzt genau diese Erkenntnis und platziert das Steuerzeichen in einem Teilstring.

```
WScript.Echo ("Preis: ", Preis, "DM \nMwSt: ", MwSt,
             "% ", Preis - Netto, " DM");
```

Um den gleichen Effekt in VBScript zu erreichen, ist ein etwas anderer Ansatz erforderlich. VBScript kennt kein "\n"-Steuerzeichen, welches Sie in die Zeichenkette einfügen können. Allerdings sind in der Sprache bereits einige benannte Konstanten vordefiniert. Und die Konstante *vbCrLf* enthält genau den Code für einen Zeilenvorschub mit Zeilenumbruch. Die *Echo*-Anweisung mit integriertem Zeilenumbruch sieht dann folgendermaßen aus:

```
WScript.Echo "Preis: ", Preis, "DM" + vbCrLf + "MwSt: ", MwSt, _
             "% ", Preis - Netto, " DM"
```

Die benannte Konstante *vbCrLf* wurde mit dem +-Zeichen in die Teilzeichenketten eingefügt.

Sie finden die beiden Beispieldateien *Echo1.vbs* und *Echo1.js* im Ordner *\Beisp\Kap05* auf der Begleit-CD-ROM. Hinweis

VBScript-Ausgaben per MsgBox-Funktion

Die *Echo*-Methode bietet eine sehr einfache Möglichkeit zur Ausgabe einer Meldung, erlaubt Ihnen jedoch keinerlei Gestaltung der Ausgabedialoge. Wer bereits mit Visual Basic oder VBA gearbeitet hat, kennt vermutlich die *MsgBox*-Prozedur, die sich zur Anzeige eines Dialogfeldes mit Textmeldungen eignet. Der Aufruf der *MsgBox*-Prozedur besitzt folgende Syntax:

```
MsgBox prompt, buttons, title
```

Die beim Prozeduraufruf anzugebenden Parameter besitzen dabei folgende Bedeutung:

- ◆ *prompt*: Definiert den Text, der im Dialogfeld angezeigt wird. Dieser Parameter muß angegeben werden. Mehrere Textzeilen lassen sich mit

den Zeichen für den Wagenrücklauf *Chr(13)* oder einen Zeilenvorschub *Chr(10)* bzw. einer Kombination aus beiden Zeichen trennen. Sie können auch die bereits erwähnte VBScript-Konstante *vbCrLf* benutzen.

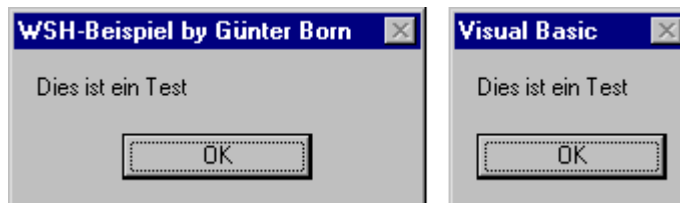
- ♦ *buttons*: Dieser optionale Parameter definiert die Anzahl und Beschriftung der Schaltflächen sowie das im Dialogfeld angezeigte Symbol. Lassen Sie diesen Parameter beim Aufruf weg, wird lediglich eine *OK*-Schaltfläche angezeigt.
- ♦ *title*: In diesem optionalen Parameter läßt sich der Titeltext für das Dialogfeld angeben. Lassen Sie diesen Parameter weg, wird der Titel der Anwendung in der Titelleiste des Dialogfelds gezeigt.

Bleibt noch die Frage, welche Werte für den Parameter *buttons* zu verwenden sind. Mit der Anweisung:

```
MsgBox "Dies ist ein Test", 0, "WSH-Beispiel by Günter Born"
```

erzeugt das Skript ein sehr einfaches Dialogfeld mit einer *OK*-Schaltfläche, dem angegebenen Text sowie dem im dritten Parameter übergebenen Titel (**Abbildung 5.5**, linkes Dialogfeld).

Abbildung 5.5:
Beispiele für
Dialoge



Lassen Sie dagegen den dritten Parameter weg, blendet das Skript das in **Abbildung 5.5** rechts gezeigte Dialogfeld ein. In diesem Dialogfeld erscheint der Text »Visual Basic« in der Titelleiste.

Hinweis

Dieses Beispiel zeigt bereits, dass die Funktionen bzw. Methoden Ihnen viele Aufgaben abnehmen. Fehlt ein optionaler Parameter, setzt der WSH einen Standardwert ein. Lassen Sie beispielsweise auch den zweiten Parameter weg, erzeugt das Skript trotzdem ein Dialogfeld mit dem im ersten Parameter übergebenen Text sowie die *OK*-Schaltfläche. Sie finden das Beispielprogramm *MsgBox.vbs* zur Anzeige der in **Abbildung 5.5** gezeigten Dialogfelder auf der Begleit-CD-ROM im Ordner *\Beisp\Kap05*.

Der obige Ansatz ist aber nur die halbe Miete. Der Text im ersten Parameter bestimmt die Ausgabe im Dialogfeld. Dieser Text läßt sich (↗ siehe den vorherigen Abschnitt) durch die *vbCrLf*-Konstante in mehrere Zeilen umbrechen. Der dritte Parameter bestimmt dagegen den Titeltext des Dialogfelds. Interessant ist aber vor allem der zweite Parameter, der das »Innenleben« des Dialogfelds festlegt. Sie können über den Wert des Parameters sowohl das Symbol als auch die Zahl der Schaltflächen samt deren Beschriftung festlegen. Und als weitere gute Nachricht gilt, dass in Windows die Werte für den Parameter als Konstante vereinbart sind. Die

Tabelle 5.1 listet die Konstanten zur Auswahl des im Dialogfeld angezeigten Symbols auf.



Konstante	Symbol	Bemerkung
0	–	Kein Symbol anzeigen (Standard)
16 vbCritical		Stopp-Symbol im Dialogfeld anzeigen.
32 vbQuestion		Fragezeichen als Symbol im Dialogfeld anzeigen.
48 vbExclamation		Ausrufezeichen als Symbol im Dialogfeld anzeigen.
64 vbInformation		Informationszeichen als Symbol im Dialogfeld anzeigen.

Tabelle 5.1:
Konstanten zur
Auswahl des
Symbols in einem
MsgBox-
Dialogfeld

Sie sehen es bereits in der ersten Spalte der Tabelle: Neben den Konstanten 0, 16 etc. sind auch benannte Konstanten festgelegt. Sie könnten direkt die numerische Konstante (z. B. 32 für das Fragezeichen) in der *MsgBox*-Anweisung einsetzen. VBScript besitzt aber die schöne Eigenschaft, dass es benannte Konstante für diesen Parameter unterstützt. Die Programme werden wesentlich besser lesbar, wenn Sie die benannten Konstanten (z. B. *vbQuestion*) benutzen. Da JScript die *MsgBox*-Prozedur nicht unterstützt, ist es auch kein Problem, dass in dieser Sprache keine benannten Konstante vordefiniert sind.

Hinweis

Zusätzlich legt der Parameter *buttons* fest, welche Schaltflächen im Dialogfeld auftreten. Die Konstanten zur Auswahl der Schaltflächen sind in **Tabelle 5.2** enthalten. Um sowohl ein Symbol als auch eine Schaltflächenkombination im Dialogfeld einzublenden, addieren Sie die betreffenden Konstanten aus **Tabelle 5.1** und **Tabelle 5.2**.

Konstante	Bemerkungen
0 vbOkOnly	Es wird die <i>OK</i> -Schaltfläche eingeblendet. Dies ist die Standardvorgabe.
1 vbOKCancel	Die Schaltflächen <i>OK</i> und <i>Abbrechen</i> sind anzuzeigen.
2 vbAbortRetryIgnore	Zeige die Schaltflächen <i>Abbruch</i> , <i>Wiederholen</i> und <i>Ignorieren</i> an.
3 vbYesNoCancel	Schaltflächen <i>Ja</i> , <i>Nein</i> und <i>Abbrechen</i> anzeigen.
4 vbYesNo	<i>Ja</i> und <i>Nein</i> als Schaltflächen anzeigen.
5 vbRetryCancel	Die Schaltflächen <i>Wiederholen</i> und <i>Abbrechen</i>

Tabelle 5.2:
Konstanten für die
anzuweisenden
Schaltflächen

anzeigen.

Hinweis

Die Beschriftung der Schaltflächen hängt von der lokalisierten Windows-Version ab. Die Tabelle zeigt die Beschriftungen für die deutsche Windows-Version. Auf der Begleit-CD-ROM finden Sie im Ordner `\Beisp\Kap05` die Datei `MsgBox1.vbs`, die nach dem Aufruf die verfügbaren Schaltflächenvarianten in mehreren Dialogfeldern anzeigt.

Die Anweisung zur Anzeige eines Dialogfelds mit einer *OK*-Schaltfläche und dem Fragezeichen sieht dann folgendermaßen aus:

```
' Zeige ein einfaches Dialogfeld mit OK-  
' Schaltfläche und Fragezeichen  
MsgBox "Frage", _  
    vbOKOnly + vbQuestion, _  
    "MsgBox-Beispiel"
```

Diese Anweisung zeigt bereits, wie hilfreich die Verwendung benannter VBA-Konstanten ist. Mit *vbOkOnly* + *vbQuestion* läßt sich leicht erkennen, was der Entwickler beabsichtigt, während die Angabe einer Konstanten der Form 33 alles andere als transparent ist.

Sofern das Dialogfeld mehr als eine Schaltfläche enthält, läßt sich über eine weitere Konstante für den Parameter *buttons* festlegen, welche dieser Schaltflächen als Standardschaltfläche benutzt wird. Diese Standardschaltfläche erhält beim Öffnen des Dialogfelds den Fokus. Sofern Sie nichts angeben, wird immer die erste Schaltfläche links im Dialogfeld als Standardschaltfläche verwendet. **Tabelle 5.3** gibt die Konstanten an, mit der Sie die Standardschaltfläche explizit festlegen können.

<i>Tabelle 5.3:</i>	Konstante	Bemerkung
Konstanten zur Auswahl der Standardschaltfläc he	0 vbDefaultButton1	Die erste Schaltfläche ist die Standardschaltfläche. Dies entspricht der Voreinstellung.
	256 vbDefaultButton2	Verwende die zweite Schaltfläche als Standardschaltfläche.
	512 vbDefaultButton3	Die dritte Schaltfläche ist die Standardschaltfläche.

Hinweis

Standardmäßig erzeugt VBA ein modales Dialogfeld, welches an die Anwendung angebunden ist. In diesem Fall muß der Benutzer das Dialogfeld schließen, bevor er mit der Anwendung weiterarbeiten kann.

Schaltflächenabfrage per MsgBox-Funktion

Dieses Wissen soll jetzt in eine einfache WSH-Anwendung umgesetzt werden. Das Programm soll sich mit den in **Abbildung 5.6** gezeigten Dialogfeldern melden.

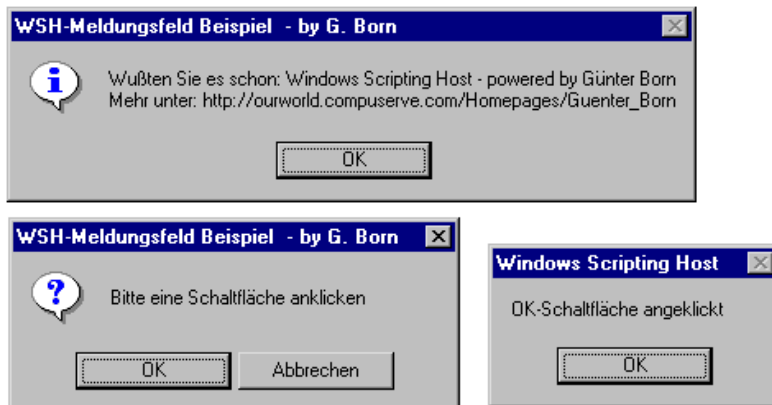


Abbildung 5.6:
Dialoge des
Beispiels

Die einzelnen Dialogfelder enthalten verschiedene Beschriftungen und Schaltflächen. Ein Dialogfeld ist dabei mit den Schaltflächen *OK* bzw. *Abbrechen* ausgestattet. Das Beispielprogramm soll die beim Schließen des Dialogfelds vom Benutzer gewählte Schaltfläche auswerten. Dies wird mit den im folgenden Listing gezeigten Programmanweisungen erreicht.

```
' *****
' File:   MsgBox2.vbs (WSH-Beispiel in VBScript)
' Autor:  Günter Born
'
' Das Script demonstriert die Ausgabe einer Meldung in VBScript.
'
' *****

Option Explicit

Dim WSHShell
Dim Text1, Text2, Text3, Text4
Dim Title, result

' Hier werden die Texte definiert

Title = "WSH-Dialogfeld Beispiel  - by G. Born"
Text1 = "Wußten Sie es schon: Windows Scripting Host - powered by Günter
Born" + vbCRLF + _
    "Mehr unter: http://ourworld.compuserve.com/Homepages/Guenter_Born"
```

Listing 5.3:
Beispiel
MsgBox2.vbs

```

Text2 = "OK-Schaltfläche angeklickt"
Text3 = "Abbrechen-Schaltfläche angeklickt"
Text4 = "Bitte eine Schaltfläche anklicken"

' Jetzt wird die MsgBox-Funktion aufgerufen
' MsgBox prompt, buttons, title
' prompt:    Der im Dialogfeld angezeigte Text
' title:     Der Titel des Dialogfelds
' buttons:   Die Schaltflächen

MsgBox Text1, + vbInformation + vbOKOnly, Title

' Jetzt versuchen wir etwas zurückzulesen
' result = MsgBox (prompt, buttons, title)

result = MsgBox (Text4, vbQuestion + vbOKCancel, Title)

' Jetzt wird die angeklickte Schaltfläche ausgewertet
If result = vbOK Then
    WScript.Echo Text2    ' Echo-Methode zur Ausgabe benutzen
Else
    WScript.Echo Text3
End If

WScript.Quit ()          ' Fertig

'*****
'***           Ende -> WSH-VBScript           ***
'*****

```

Das Programm definiert die Textkonstanten zur Ausgabe im Kopf des Skripts. Anschließend wird der in **Abbildung 5.6** links oben gezeigte Eingangsdialog eingeblendet. Sobald der Benutzer diesen Dialog über die *OK*-Schaltfläche schließt, wird das nächste Dialogfeld mit zwei Schaltflächen gezeigt. Die Schaltflächen werden durch gezielte Auswahl des *buttons*-Werts gewählt:

```
result = MsgBox (Text4, vbQuestion + vbOKCancel, Title)
```

Bei diesem Aufruf kommt jedoch eine Neuerung hinzu. *MsgBox* wird in diesem Beispiel als Funktion aufgerufen. Daher müssen Sie die Parameter in Klammern einfassen. Zusätzlich liefert die *MsgBox*-Funktion einen Wert zurück, der der vom Benutzer gewählten Schaltfläche im Dialogfeld entspricht. Die **Tabelle 5.4** gibt die Werte an, die von *MsgBox* zurückgeliefert werden. Welche Werte auftreten können, hängt von den verfügbaren Schaltflächen ab.

Wert	Konstante	Bedeutung
------	-----------	-----------

1	vbOK	Der Benutzer hat die <i>OK</i> -Schaltfläche angeklickt.
2	vbCancel	Die Schaltfläche <i>Abbrechen</i> wurde gewählt.
3	vbAbort	Die Schaltfläche <i>Abbruch</i> wurde gewählt.
4	vbRetry	Die Schaltfläche <i>Wiederholen</i> wurde gewählt.
5	vbIgnore	Der Benutzer hat auf <i>Ignorieren</i> geklickt.
6	vbYes	Anwahl der <i>Ja</i> -Schaltfläche.
7	vbNo	Die Schaltfläche <i>Nein</i> wurde ausgewählt.

Tabelle 5.4:
Rückgabecodes der
MsgBox-Funktion

Sie können dabei im VBScript-Code entweder mit symbolischen Konstanten arbeiten oder direkt die Werte 1 bis 7 abfragen. Durch Auswerten der Rückgabewerte für die gewählte Schaltfläche des Dialogfelds läßt sich eine einfache Benutzerführung realisieren. Dies wird in **Listing 5.3** mit folgenden Anweisungen genutzt:

```
' Jetzt wird die angeklickte Schaltfläche ausgewertet
If result = vbOK Then
    WScript.Echo Text2 ' Echo-Methode zur Ausgabe benutzen
Else
    WScript.Echo Text3
End If
```

Hat der Benutzer die *OK*-Schaltfläche gewählt, zeigt das Programm dies in einem weiteren Dialog an. In diesem Beispiel habe ich zur Demonstration der Ausgabe nicht die *MsgBox*-Prozedur benutzt, sondern die vom Windows Scripting Host-Objekt bereitgestellte *Echo*-Methode:

```
WScript.Echo Text2
```

In der obigen Anweisung stellt *WScript.Echo* den Inhalt der Variablen *Text2* in einem Dialogfeld dar.

Sie finden das VBScript-Beispiel *MsgBox2.vbs* auf der Begleit-CD-ROM im Ordner *\Beisp\Kap05*. Hinweis

VBScript-Dialogfelderanzeige mit Prozeduraufruf

Das nachfolgende Beispiel zeigt ebenfalls Dialogfelder an. In diesem Beispiel wird jedoch eine Prozedur mit eingebunden, um dem Benutzer die Auswahl zwischen deutschen und englischen Dialogfeldtexten zu ermöglichen. Beim Aufruf des Skripts zeigt dieses das in **Abbildung 5.7** am oberen Rand angeordnete Dialogfeld. Der Benutzer kann anschließend durch Anklicken

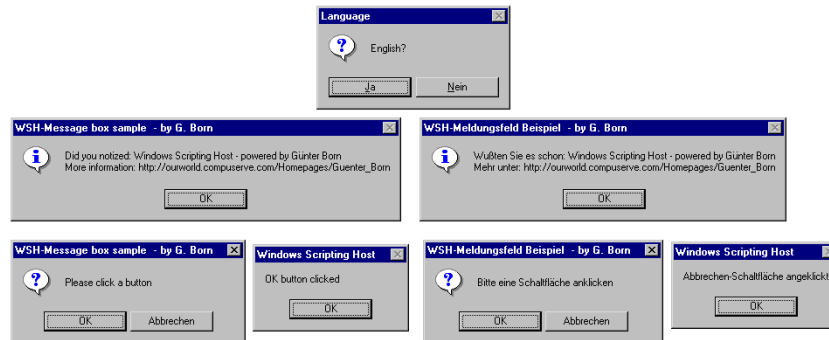
der Schaltflächen wählen, in welcher Sprache die Dialoge ausgeführt werden sollen.

Hinweis

Beachten Sie, dass die Beschriftung der Schaltflächen von der Windows Scripting Host-Version bzw. von der Betriebssystemversion abhängen. Ein deutschsprachiges Windows liefert Ihnen immer deutschsprachige Schaltflächenbeschriftungen zurück.

Die Dialogfelder in **Abbildung 5.7** geben die jeweiligen Ausgaben des Programmes in Abhängigkeit von der gewählten Sprache wieder. Die betreffenden Dialogmeldungen wurden im Kopf des Skripts als Textvariablen definiert.

Abbildung 5.7:
Mehrsprachige Dialoge



Zur Umschaltung der gewünschten Sprachversion bedient sich das Beispielprogramm der benutzerdefinierten Prozedur *Localize*. In einem einfachen *MsgBox*-Aufruf wird das erste Dialogfeld zur Auswahl der Sprache eingeblendet. Der Rückgabewert der Funktion wird direkt in einer *If*-Anweisung ausgewertet.

```
If MsgBox ("English?", vbQuestion+vbYesNo, "Language") = vbYes Then
    Localize (0)          ' *** Englisch
Else
    Localize (1)          ' *** Deutsch
End if
```

Anschließend wird die benutzerdefinierte Prozedur *Localize* mit einem Parameter aufgerufen. Dieser Parameter legt die gewünschte Sprachversion der Ausgaben fest. Die Prozedur *Localize* besitzt einen sehr einfachen Aufbau. In Abhängigkeit vom übergebenen Parameter *lang* verzweigt der Programmablauf in der *If*-Anweisung. In den Zweigen werden dann die Werte der global definierten Variablen mit Texten belegt.

```
Sub Localize (lang)
    If lang = 0 Then
        Message = "Please enter something"
        Title = "WSH-Message box sample .... "
        Text2 = "OK button clicked"
        Text3 = "Cancel button clicked"
        Text4 = "Please click a button"
    Else
        ' *** German
```

```

Message = "Eingabe"
Title = "WSH-Dialogfeld Beispiel ...."
Text2 = "OK-Schaltfläche angeklickt"
Text3 = "Abbrechen-Schaltfläche angeklickt"
Text4 = "Bitte eine Schaltfläche anklicken"
End If
End Sub

```

Die Prozedur wird durch den *End Sub*-Befehl beendet. Nach der Ausführung dieser Prozedur sind die Variablen definiert. Benutzt das Programm diese beim Aufruf der *MsgBox*-Prozedur, wird der Text automatisch in der jeweiligen Fassung ausgegeben. Der Vorteil dieses Ansatzes: Sie erhalten nicht nur eine zweisprachige Ausgabe, sondern das eigentliche Programm wird auch leichter verständlich. Beim *MsgBox*-Aufruf geben Sie lediglich die Variablen an, die »ellenlangen« Textdefinitionen entfallen in der Anweisung. Und die Pflege der Texte kann zentral in der Prozedur *Localize* erfolgen.

```

' *****
' File:   MsgBox3.vbs (WSH-Beispiel in VBScript)
' Autor:  Günter Born
'
' Beispiel mit einer Prozedur, die eine lokalisierte Ausgabe
' ermöglicht
' *****
Option Explicit

Dim WSHShell
Dim Text1, Text2, Text3, Text4
Dim Title, result, Message

' *** Hier lassen sich die Texte lokalisieren ***
'   0 = Englisch, 1 = Deutsch
If MsgBox ("English?", vbQuestion+vbYesNo, "Language") = vbYes Then
    Localize (0)           ' *** Englisch
Else
    Localize (1)           ' *** Deutsch
End if

' Jetzt wird die MsgBox-Funktion aufgerufen
' MsgBox prompt, buttons, title
' prompt:   Der im Dialogfeld angezeigte Text
' title:    Der Titel des Dialogfelds
' buttons:  Die Schaltflächen
' Fehlen Werte, benutzt MsgBox Vorgabewerte

```

Listing 5.4:
Beispiel
MsgBox3.vbs

```

MsgBox Text1, vbInformation + vbOKOnly, Title

' Jetzt versuchen wir etwas zurückzulesen
' result = MsgBox (prompt, buttons, title)

result = MsgBox (Text4, vbQuestion + vbOKCancel, Title)

If result = vbOK Then
    WScript.Echo Text2 ' Ausgabe per Echo-Methode des WScript-Objekts
Else
    WScript.Echo Text3
End If

WScript.Quit()

' Hier kommt die Prozedur, die die Lokalisierungstext setzt
Sub Localize (lang)
    If lang = 0 Then
        Message = "Please enter something"
        Title = "WSH-Message box sample - by G. Born"
        Text1 = "Did you notized: Windows Scripting Host - powered by Günter Born"
        + vbCRLF + _
            "More information:
http://ourworld.compuserve.com/Homepages/Guenter_Born"
        Text2 = "OK button clicked"
        Text3 = "Cancel button clicked"
        Text4 = "Please click a button"
    Else
        ' *** German
        Message = "Eingabe"
        Title = "WSH-Meldungsfeld Beispiel - by G. Born"
        Text1 = "Wußten Sie es schon: Windows Scripting Host - powered by Günter
Born" + vbCRLF + _
            "Mehr unter:
http://ourworld.compuserve.com/Homepages/Guenter_Born"
        Text2 = "OK-Schaltfläche angeklickt"
        Text3 = "Abbrechen-Schaltfläche angeklickt"
        Text4 = "Bitte eine Schaltfläche anklicken"
    End If
End Sub

' *****
' ***           Ende -> WSH-VBScript           ***
' *****

```

Sie finden das VBScript-Beispiel *MsgBox3.vbs* auf der Begleit-CD-ROM im Ordner *\Beisp\Kap05*.

Anwenden der Popup-Methode

Die obigen Beispiele befaßten sich mit der Anzeige der Meldungen sowie mit der Abfrage der Schaltflächen in VBScript. In JScript steht die *MsgBox*-Prozedur leider nicht zur Verfügung. Der nachfolgende Abschnitt zeigt, wie Sie mit der *Popup*-Methode die gleichen Ausgaben wie mit der *MsgBox*-Funktion erzeugen können.

Dialogfelder in JScript ausgeben

Es wurde bereits erwähnt, daß es *MsgBox* in JScript nicht gibt. Auch die im Internet Explorer unterstützten JScript-Funktionen wie *confirm()* oder der Zugriff auf die *write*-Methode in der Form *document.write* ist in einem WSH-Skript nicht möglich. Das WSH-Skript-Objekt unterstützt keine Formulare oder Dokumente. Daher müssen wir einen etwas anderen Ansatz zur Ausgabe von Benutzermeldungen sowie zur Abfrage von Schaltflächen wählen. Zu Beginn dieses Abschnitts haben Sie bereits die *Echo*-Methode des *WScript*-Objekts kennengelernt. Mit der Anweisung:

```
WScript.Echo ("OK-Schaltfläche angeklickt (Code: " + result + ")");
```

wird der Text sowie den Inhalt der Variablen *result* in einem Dialogfeld ausgegeben. Der Nachteil dieses Ansatzes: Das Dialogfeld besitzt einen Standardtitel («Windows Scripting Host») sowie eine *OK*-Schaltfläche. Möchten Sie dagegen die Schaltflächen, das im Dialogfeld angezeigte Symbol, den Titel etc. wie bei der *MsgBox*-Funktion beeinflussen sowie die angewählten Schaltfläche auswerten, bringt Sie die *Echo*-Methode nicht weiter. Hier müssen Sie auf die *Popup*-Methode des *Shell*-Objekts ausweichen. Leider stellt der Windows Scripting Host das *Shell*-Objekt nicht direkt zur Verfügung. Sie müssen dieses Objekt vielmehr zuerst als Instanz erzeugen. Bisher haben Sie diesen Schritt aber noch nicht kennengelernt.

Das Instantiieren eines Objekts erfolgt mit der *CreateObject*-Methode des *WScript*-Objekts. Sie müssen zusätzlich noch die sogenannte Prog-ID des Objekts kennen. Die hier benötigte *Popup*-Methode befindet sich im *Shell*-Objekt und dieses Objekt ist ein Unterobjekt des *WScript*-Objekts. Die folgenden Anweisungen zeigen, wie sich eine Referenz auf die benötigte Objektinstanz unter JScript realisieren läßt:

```
var WSHShell = WScript.CreateObject("WScript.Shell");
```

Über eine einfache Variablendeklaration wird eine Objektvariable *WSHShell* angelegt. In dieser Variablen wird dann eine Referenz auf das neue Objekt

hinterlegt, welches mit *WScript.CreateObject* erzeugt wird. *WScript* stellt dabei das Objekt dar, welches die *CreateObject*-Methode unterstützt. Die *CreateObject*-Methode erwartet als Parameter die Prog-ID des neuen Objekts. Für das *Shell*-Objekt ist diese Prog-ID als *WScript.Shell* festgelegt. Sobald Sie die obige Anweisung ausführen, wird folglich das neue Objekt angelegt und dann eine Referenz auf dieses Objekt in der Variablen gespeichert. Über diese Objektvariable können Sie anschließend auf die Methoden und Eigenschaften des Objekts zugreifen. Die folgenden Anweisungen zeigen die Schritte, die zur Anwendung der *Popup*-Methode erforderlich sind:

```
var WSHShell = WScript.CreateObject("WScript.Shell");
```

```
var intDoIt;
```

```
intDoIt = WSHShell.Popup(L_Welcome_MsgBox_Message_Text,  
                        0,  
                        L_Welcome_MsgBox_Title_Text,  
                        vbOKCancel + vbInformation );
```

Sobald die Objektvariable *WSHShell* angelegt wurde, läßt sich über *WSHShell.Popup* eine Ausgabe in einem Dialogfeld vornehmen. Der *WSHShell.Popup*-Aufruf besitzt anschließend folgende Syntax:

```
res = WSHShell.Popup(txt, wait, title, buttons);
```

Die *Popup*-Methode besitzt mehrere Parameter, die folgende Bedeutung besitzen:

- ◆ Der Parameter *txt* nimmt den Text auf, der im Dialogfeld auszugeben ist. Sie können diesen Text wie bei den anderen Beispielen aus Teilzeichenketten und Variablen zusammensetzen sowie Zeilenumbrüche einfügen.
- ◆ Die Variable *wait* erlaubt Ihnen die Angabe einer Wartezeit in Sekunden. Wählt der Benutzer innerhalb der Wartezeit keine Schaltfläche, schließt das Skript automatisch das Dialogfeld nach Ablauf dieser Wartezeit. Mit dem Wert 0 wird diese Funktion abgeschaltet.
- ◆ Im Parameter *title* läßt sich der Inhalt der Titelleiste angeben.
- ◆ Der Parameter *buttons* wählt das anzuzeigende Symbol sowie die auszugebenden Schaltflächen. Die Werte für die Konstante entsprechen den VBScript-Vorgaben aus den weiter oben angegebenen Tabellen (**Tabelle 5.1** bis **Tabelle 5.4**). Beachten Sie aber, das in JScript keine symbolischen Konstante vereinbart sind.

Nach dem Schließen des Dialogfelds gibt die *Popup*-Methode den Code der gewählten Schaltfläche zurück. Das folgende Listing zeigt den Aufbau des kompletten Moduls. Die Ausgabe mittels der *Popup*-Methode wurde in Anlehnung an die von Microsoft gelieferten Beispiele in die Funktion

Welcome() verlagert. Die Funktion erwartet beim Aufruf keine Parameter, da die Ausgabetexte über globale Variablen definiert wurden. Folglich lassen sich diese Variablen direkt in der Funktion beim Aufruf der *Popup*-Methode als Argumente nutzen.

In dieser Hinsicht möchte ich noch auf eine Besonderheit hinweisen. JScript kennt keine benannten Konstanten (z. B. *vbOKonly*). Daher habe ich im Modulkopf die benötigten symbolischen Konstante als Variablen vereinbart und direkt einen Wert zugewiesen. Anschließend läßt sich im Programm an beliebiger Stelle auf diese »Pseudo-Konstanten« zugreifen.

In der Funktion *Welcome()* wird als erstes das *Shell*-Objekt instantiiert (d. h. es wird eine Referenz auf eine Instanz dieses Objekts erzeugt und in einer Variablen hinterlegt). Dann erfolgt der Aufruf der *Popup*-Methode. Der Befehl:

```
return intDoIt;
```

gibt den von der *Popup*-Methode zurückgelieferten Code der angeklickten Schaltfläche an das rufende Programm zurück. Im Hauptmodul wertet die Anweisung:

```
if (result == vbOK)
```

den zurückgegebenen Code aus. In Abhängigkeit von der (vom Benutzer) gewählten Schaltfläche wird anschließend eine Nachricht ausgegeben. Im Gegensatz zu den VBScript-Beispielen wird hier im abschließenden Dialogfeld auch der Code der angeklickten Schaltfläche unter Verwendung der *Echo*-Methode eingeblendet.

```
WScript.Echo ("OK-Schaltfläche angeklickt " + "(Code: " + result + ")");
```

Dies erlaubt Ihnen, die Konstanten für die Schaltflächendefinition zu überprüfen. Einzelheiten finden Sie in folgendem Listing.

```
//*****  
// File:  Popup.js (WSH-Beispiel in JScript)  
// Autor: Günter Born  
//  
// Testbeispiel zur Ausgabe einer Meldung mit Popup.  
//*****  
  
var vbOKCancel = 1;      // Variablen vereinbaren  
var vbOK = 1;  
var vbInformation = 64;  
var vbCancel = 2;  
var result;  
  
var L_Welcome_MsgBox_Message_Text  
= "Bitte eine Schaltfläche anklicken";
```

Listing 5.5:
Beispiel Popup.js

```

var L_Welcome_MsgBox_Title_Text
= "Borns Windows Scripting Host-Beispiel";

result = Welcome();    // Eingangsdialog zeigen

// Ergebnisse anzeigen

if (result == vbOK)
{
    WScript.Echo ("OK-Schaltfläche angeklickt " +
        "(Code: " + result + ")");
}
else
{
    if (result == vbCancel)
    {
        WScript.Echo ("Abbrechen-Schaltfläche angeklickt " +
            "(Code: " + result + ")");
    }
}

WScript.Echo("Jetzt sind wir fertig");

WScript.Quit();        // Beende Skript

////////////////////////////////////
//
function Welcome()
// Funktion zur Anzeige eines Dialogfelds
{ // hier brauchen wir das Shell-Objekt !!!
    var WSHShell = WScript.CreateObject("WScript.Shell");
    var intDoIt;

    intDoIt = WSHShell.Popup(L_Welcome_MsgBox_Message_Text,
        0,
        L_Welcome_MsgBox_Title_Text,
        vbOKCancel + vbInformation );
    return intDoIt; // Code der Schaltfläche zurückliefern
}

//*****
//***      Ende -> WSH-JScript      ***

```


//*****

Sie finden das JScript-Beispiel *Popup.js* auf der Begleit-CD-ROM im Ordner *Hinweis*
\Beisp\Kap05.

Klappt Popup auch in VBScript?

Die meisten VB-Programmierer werden beim Umstieg auf VBScript wohl auf die *MsgBox*-Funktion (oder Prozedur) zurückgreifen. Diese steht direkt im Sprachumfang zur Verfügung und erlaubt eine sehr komfortable Ausgabe (↗ siehe die vorhergehenden Abschnitte). Allerdings steht es Ihnen frei, anstelle der *MsgBox*-Funktion auf die *Popup*-Methode des *Shell*-Objekts zuzugreifen. Das nachfolgende Listing demonstriert, wie der Aufruf dieser Methode in VBScript aussieht. Ich habe das Beispiel bewußt einfach gehalten, um lediglich den Einsatz der Methode zu zeigen.

```
'*****
' File:   Popup.vbs (WSH-Beispiel in VBScript)
' Autor:  Günter Born
'
' Zweck: Ausgabe eines Meldungsfelds mit der
'        Popup-Methode
'*****

Option Explicit

Dim result
Dim WSHShell

' hier brauchen wir das Shell-Objekt !!!
Set WSHShell = WScript.CreateObject("WScript.Shell")

' Jetzt wird die Popup-Methode eingesetzt

result = WSHShell.Popup("Ausgabe mit der Popup-Methode", _
                        0, _
                        "WSH - by Günter Born", _
                        vbOKCancel + vbInformation)

WScript.Echo "Rückgabewert ", result

WScript.Quit

'*****
```

Listing 5.6:
Beispiel Popup.vbs

```
'***           Ende -> WSH-VBScript           ***  
'*****
```

Hinweis

Sie finden das VBScript-Beispiel *Popup.vbs* auf der Begleit-CD-ROM im Ordner *\Beisp\Kap05*.

Nutzung der Objekte, Methoden und Eigenschaften des WSH

6

Arbeiten mit dem WScript-Objekt	158
Zugriff auf Umgebungsvariablen	169
Objekte holen und freigeben	183
Programmaufrufe per WSH	189

In den vorhergehenden Kapiteln haben Sie die grundlegenden Techniken zum Erstellen von Skripten sowie einige Methoden und Eigenschaften kennengelernt. In diesem Kapitel möchte ich Ihnen schrittweise zeigen, wie Sie den Windows Scripting Host verwenden können, um bestimmte Aufgaben zu automatisieren.

- ♦ Sie lernen, wie sich die Eigenschaften des *WScript*-Objekts abfragen und in einem Dialogfeld anzeigen lassen. Dies erlaubt Ihnen, die wichtigsten Informationen zum WSH und zum aktuellen Skript abzurufen.
- ♦ Ein weiteres Beispiel erläutert, wie Sie die einem Skript beim Aufruf übergebenen Argumente auswerten können.
- ♦ Zum Zugriff auf andere Objekte müssen diese zunächst instantiiert werden. Hier erfahren Sie, wie die Methoden *CreateObject* und *GetObject* des *WScript*-Objekts eingesetzt werden. Außerdem erfahren Sie, wie ein Objekt wieder freigegeben wird.
- ♦ Ein weiterer Abschnitt zeigt Ihnen, wie Sie andere Anwendungen aus einem Skript heraus aufrufen können.

Mit diesem Wissen sind Sie sehr schnell in der Lage, eigene einfache Anwendungen zu realisieren und die Funktionalität des Betriebssystems zu erweitern.

Arbeiten mit dem WScript-Objekt

Das *WScript*-Objekt bietet Ihnen verschiedene Methoden und Eigenschaften. Mit *Echo* und *Quit* haben Sie bereits zwei Methoden dieses Objekts kennengelernt. Der nachfolgende Abschnitt zeigt, wie Sie die Eigenschaften dieses Objekts für eigene Zwecke einsetzen können.

Anzeige der WSH- und Skript-Eigenschaften

In Kapitel 5 wurde erwähnt, dass der Windows Scripting Host das aktuell laufende Skript als Objekt auffaßt und dem Skript über das *WScript*-Objekt zur Verfügung stellt. Wenn Sie *Cscript.exe* unter MS-DOS aufrufen, gibt der Host beispielsweise die aktuelle Versionsnummer in der Befehlszeile aus. Diese Versionsnummer ist eigentlich recht interessant, da Microsoft verschiedene Versionen des WSH bereitstellt. Aber auch weitere Eigenschaften wie beispielsweise der Pfad zum Host, dessen Name etc. sind sicherlich in verschiedenen Fällen sehr informativ.

Abbildung 6.1:
Anzeige
verschiedener
WScript-
Eigenschaften



Hinweis

Der im Anhang enthaltenen WSH-Programmierreferenz können Sie entnehmen, welche Eigenschaften das *WScript*-Objekt direkt zur Verfügung stellt.

An dieser Stelle möchte ich diesen Gedanken aufgreifen und ein einfaches Skript vorstellen, welches die Eigenschaften des Hosts sowie die

Eigenschaften des Skripts ermittelt und in einem Dialogfeld anzeigt (**Abbildung 6.1**).

Der Zugriff auf eine Eigenschaft des *WScript*-Objekts ist sehr einfach. Da das *WScript*-Objekt vom WSH automatisch exponiert wird, können Sie direkt auf dessen Eigenschaften zugreifen. Die Anweisung:

```
Name = WScript.Application
```


liest beispielsweise die *Application*-Eigenschaft des *WScript*-Objekts und weist diese der Variablen *Name* zu. In dieser Eigenschaft finden Sie beim WSH den Text »Windows Scripting Host«. Die Eigenschaft läßt sich also verwenden, um zu prüfen, ob das Skript auch wirklich unter dem WSH ausgeführt wird. Der Anwender könnte das Skript ja auch im Rahmen eines HTML-Dokuments oder bei einer Active Server Page (ASP)-Datei als externes Skript nutzen. Die nachfolgende Aufstellung gibt Ihnen eine grobe Übersicht über die in diesem Beispiel benutzten Eigenschaften des *WScript*-Objekts:

- ◆ *Application*: Liefert den Namen der Anwendung als Zeichenkette. Da das Skript unter dem WSH ausgeführt wird, ist dies die Zeichenkette »Windows Scripting Host«.
- ◆ *Name*: Hier wird der Name der Anwendung (des WSH) als Zeichenkette zurückgegeben.
- ◆ *Version*: Gibt die aktuelle Versionsnummer des WSH zurück.
- ◆ *FullName*: Liefert den Namen des Hosts (Name der EXE-Datei einschließlich des Pfads).
- ◆ *Path*: Diese Eigenschaft enthält nur die Pfadangabe zur EXE-Datei des WSH.

Die *Interactive*-Eigenschaft liefert einen Hinweis, ob das Skript interaktiv ausgeführt wird, als Boolean-Wert (wahr, falsch) zurück. Sofern Sie Benutzerausgaben benötigen, muß der Interaktiv-Modus zwangsweise eingeschaltet sein. Bei der Anzeige des Status in einem Meldungsfeld werden Sie daher immer die Angabe »wahr« (bzw. *true*) finden. Hinweis

Neben diesen Eigenschaften, die sich direkt auf den WSH beziehen, können Sie über das *WScript*-Objekt auch den Namen des ausgeführten Skripts sowie dessen Pfad ermitteln.

- ◆ *ScriptFullName*: In dieser Eigenschaft steckt sowohl der Pfad als auch der Name des eigentlichen Skripts.
- ◆ *ScriptName*: Benötigen Sie nur den Namen des Skripts, läßt sich dieser über die *ScriptName*-Eigenschaft direkt abfragen.

Diese Angaben sind beispielsweise hilfreich, falls Sie den Namen des Skripts anzeigen oder den Pfad zum Skript ermitteln möchten (siehe weiter unten im  Abschnitt »MS-DOS-Befehl per Run-Methode ausführen« bzw. im Kapitel mit den Rezepten). Weitere Hinweise zu den oben aufgeführten Eigenschaften finden Sie in der WSH-Programmierreferenz.

Die Lösung in VBScript

Die Lösung in VBScript können Sie dem **Listing 6.1** entnehmen. Die Abfrage der Eigenschaften gestaltet sich sehr einfach. Sie müssen lediglich das *WScript*-Objekt – gefolgt von einem Punkt und dem Namen der Eigenschaft – in einer Zuweisung angeben. Etwas Aufwand bereitet lediglich die Aufbereitung der Informationen zur Ausgabe im Dialogfeld. In **Listing 6.1** wird die *Message*-Variable verwendet, um den Ausgabetext aufzubereiten. Über die Konstante *vbCrLf* wird die Ausgabe in mehreren Zeilen erzwungen. Die *ScriptName*-Eigenschaft wird in diesem Beispiel verwendet, um den Namen des Skripts in der Titelleiste des Dialogfelds mit einzublenden.

Listing 6.1:
Properties.vbs

```
'*****
' File:   Properties.vbs (WSH-Beispiel in VBScript)
' Autor:  (c) G. Born
'
' Zeigt die Eigenschaften des WScript-Objekts in
' einfachen Dialogfeldern an.
'*****
Option Explicit

Dim Message
Dim Title

' Zeige die Eigenschaften des WScript-Objekts
' Zuerst einige Host-Eigenschaften

Message = "WScript-Eigenschaften des Hosts" + vbCrLf + vbCrLf
Message = Message + "Application: " + WScript.Application + vbCrLf
Message = Message + "Name: " + WScript.Name + vbCrLf
Message = Message + "Version: " + WScript.Version + vbCrLf
Message = Message + "FullName: " + WScript.FullName + vbCrLf
Message = Message + "Path: " + WScript.Path + vbCrLf

' Ermittle den Interactive-Status
If (WScript.Interactive) Then
    Message = Message + "Interactive: wahr" + vbCrLf
Else
```

```

    Message = Message + "Interactive: falsch" + vbCRLF
End if

' Jetzt noch die Skript-Eigenschaften ausgeben
Message = Message + vbCRLF
Message = Message + "WScript-Eigenschaften des Skripts" + vbCRLF + vbCRLF
Message = Message + "ScriptFullName : " + WScript.ScriptFullName + vbCRLF
Message = Message + "ScriptName : " + WScript.ScriptName + vbCRLF

' Titeltext initialisieren
Title = "WSH-Beispiel " + WScript.ScriptName + " - by G. Born"

MsgBox Message, vbInformation + vbOKOnly, Title

WScript.Quit() ' Jetzt wird das Skript beendet

'*****
'*** Ende -> WSH powered by Günter Born ***
'*****

```

Sie finden die Datei *Properties.vbs* im Ordner *\Beisp\Kap06* auf der Begleit-CD-ROM. Hinweis

Die Eigenschaften in JScript anzeigen

Der Vollständigkeit halber habe ich nachfolgend die Lösung auch in JScript implementiert. Die Eigenschaften lassen sich ebenfalls durch Angabe des Objekts *WScript* – gefolgt von einem Punkt und dem Namen der Eigenschaft – abfragen. Zur Aufteilung der Informationen in mehrere Zeilen im Dialogfeld wird der Ausgabertext in der Variablen *Message* hinterlegt. Zeilenumbrüche werden mit dem Steuerzeichen "\n" markiert.

```

//*****
// File:    Properties.js (WSH-Beispiel in JScript)
// Autor:    (c) G. Born
//
// Zeigt die Eigenschaften des WScript-Objekts in
// einfachen Dialogfeldern an.
//*****
//

var Message, Title, tmp;
var vbInformation = 64;
var vbOKOnly = 0;

```

Listing 6.2:
Properties.js

```

// Zeige die Eigenschaften des WScript-Objekts
// Zuerst einige Host-Eigenschaften

Message = "WScript-Eigenschaften des Hosts \n\n";
Message = Message + "Application: " + WScript.Application + "\n";
Message = Message + "Name: " + WScript.Name + "\n";
Message = Message + "Version: " + WScript.Version + "\n";
Message = Message + "FullName: " + WScript.FullName + "\n";
Message = Message + "Path: " + WScript.Path + "\n";

// Ermittle den Interactive-Status
if (WScript.Interactive)
    Message = Message + "Interactive: wahr" + "\n"
else
    Message = Message + "Interactive: falsch" + "\n";

// Jetzt noch die Skript-Eigenschaften ausgeben
Message = Message + "\n";
Message = Message + "WScript-Eigenschaften des Skripts \n\n";
Message = Message + "ScriptFullName : " + WScript.ScriptFullName + "\n";
Message = Message + "ScriptName : " + WScript.ScriptName + "\n";

// Titeltext initialisieren
Title = "WSH-Beispiel " + WScript.ScriptName + " - by G. Born";

var objAdr = WScript.CreateObject("WScript.Shell");

tmp = objAdr.Popup (Message, vbInformation + vbOKOnly, Title);

WScript.Quit() ; // Jetzt wird das Skript beendet

//*****
//***      Ende -> WSH-JScript      ***
//*****

```

Hinweis

Sie finden die Datei *Properties.js* im Ordner *\Beisp\Kap06* auf der Begleit-CD-ROM.

Eigenschaften der Script-Engine anzeigen

Im vorherigen Abschnitt haben Sie gelernt, wie sich die Eigenschaften des WSH abfragen lassen. Diese Informationen reichen aber nicht immer aus. In einigen Fällen benötigen Sie zusätzliche Informationen über die Script-Engine. So lässt sich beispielsweise die Versionsnummer des betreffenden Interpreters abfragen (**Abbildung 6.2**). Dies ist hilfreich, da die bisherigen Implementierungen nicht immer alle Funktionen fehlerfrei ausführen.



Abbildung 6.2:
Anzeige der
ScriptEngine-
Eigenschaften

Sowohl VBScript als auch JScript stellen für diesen Zweck mehrere gleichnamige Funktionen zur Verfügung, die Sie direkt aufrufen können:

- ◆ *ScriptEngine()*: Diese Funktion liefert eine Zeichenkette, die die Sprache des Interpreters im Klartext angibt.
- ◆ *ScriptEngineMajorVersion()*: Diese Funktion gibt die Hauptversionsnummer der Script-Engine wieder. Unter Windows ist dies beispielsweise die Version 3.
- ◆ *ScriptEngineMinorVersion()*: Die Funktion gibt die Nummer der Unterversion als Zeichenkette zurück.
- ◆ *ScriptEngineBuildVersion()*: Mit dieser Funktion lässt sich die Build-Nummer der jeweiligen Script-Engine abfragen.

Das nachfolgende Listing zeigt die Anwendung dieser Funktionen. Das Skript erzeugt den in **Abbildung 6.2** gezeigten Dialog.

```
' *****  
' File:   Engine.vbs (WSH-Beispiel in VBScript)  
' Autor:  Günter Born  
'  
' Zweck: Das Skript zeigt die Version der Sprach-  
'        Engine an.  
' *****  
Option Explicit  
  
DIM txt
```

Listing 6.3:
Engine.vbs

```
' Ermittle die Version der Sprach-Engine

txt = "Script-Engine: " + CStr(ScriptEngine()) + vbCRLF
txt = txt + "Version: " + CStr(ScriptEngineMajorVersion())
txt = txt + "." + CStr(ScriptEngineMinorVersion()) + vbCRLF
txt = txt + "Build: " + CStr(ScriptEngineBuildVersion())

WScript.Echo txt



WScript.Quit()

'*****
'***           Ende -> WSH-VBScript           ***
'*****
```

Hinweis

Sie finden die Datei *Engine.vbs* im Ordner *\Beisp\Kap06* auf der Begleit-CD-ROM. Die entsprechende JScript-Version finden Sie in der Datei *Engine.js* im gleichen Ordner.

Auswerten der Skript-Parameter


In  Kapitel 1 wurde bereits gezeigt, wie Sie einem Skript bestimmte Parameter (z. B. einen Dateinamen oder einen Schalter) übergeben können. Sie benötigen lediglich ein geeignetes Verfahren, um im Skript auf diese Parameter zugreifen zu können. In  Kapitel 1 wurde ein Skript für diesen Zweck vorgestellt. Dort hatte ich im Hinblick auf zusätzliche Erläuterungen auf die Folgekapitel verwiesen. Mit dem Wissen aus den vorhergehenden Kapiteln sollten Sie die nachfolgenden Erläuterungen verstehen und später selbständig einsetzen können.

Die Parameter eines Skripts werden über die *Arguments*-Eigenschaft des *WScript*-Objekts abgefragt. Allerdings ist dies nicht ganz so trivial als beispielsweise der Zugriff auf die *Name*-Eigenschaft. Die Anweisung:

```
Parameter = WScript.Arguments
```

führt daher nicht nur zu einem Laufzeitfehler, sondern birgt auch noch eine zusätzliche Schwierigkeit. Ein Skript kann ja mehr als ein Parameter aufweisen. Die Zuweisung an *Parameter* erlaubt aber nur die Übergabe eines Wertes.

Hinweis

An dieser Stelle kommt der Objektansatz zum Tragen. Mit diesem Ansatz wird eine sehr elegante Lösung zum Zugriff auf die Parameter des Skripts möglich. In  Kapitel 5 wurde bereits erläutert, dass ein Objekt weitere Objekte aufnehmen kann (dort wurde das Beispiel eines Dialogfelds benutzt, welches eine Schaltfläche als Objekt enthält). Spinnen wir diese Idee etwas

weiter: Ein Dialogfeld könnte nicht nur ein Unterobjekt (z. B. eine Schaltfläche) sondern mehrere Objekte (d. h. mehrere Schaltflächen) enthalten. Dann stellt das Objekt so eine Art Container für die Objekte dar. Enthält ein Objekt mehrere verschiedene Unterobjekte, werden diese in der Objekthierarchie aufgeführt. Es ist aber vorstellbar, dass im Objekt einfach mehrere gleiche Objekte gesammelt werden. So kann eine Anwendung beispielsweise mehrere Dokumente geladen haben. Das Objekt *Application* (die Anwendung) besitzt dann mehrere *Documents*-Objekte. Immer wenn ein Objekt mehrere Unterobjekte enthalten kann, spricht man von einer Auflistung (englisch Collection). Und genau dieser Ansatz wird zum Zugriff auf die Argumente eines Skripts benutzt.

Die *Arguments*-Eigenschaft liefert keinen einfachen Wert zurück, sondern ein *WshArguments*-Objekt. Es handelt sich dabei aber um kein Einzelobjekt, sondern um eine Sammlung von Objekten. Sie können daher mit etwas Programmcode auf die einzelnen Objekte dieser Auflistung (Collection) zugreifen. Zum Zugriff auf die Parameter des Skripts sind mehrere Schritte erforderlich. Diese Schritte werden nachfolgend für VBScript gezeigt. Die Anweisung:

```
Set objArgs = Wscript.Arguments
```

liest die *Arguments*-Eigenschaft des *WScript*-Objekts zurück. Da diese Eigenschaft keinen einfachen Wert sondern eine Auflistung von Objekten liefert, müssen Sie das Schlüsselwort *Set* verwenden, um eine Objektvariable anzulegen. Über diese Objektvariable *objArgs* können Sie anschließend auf die Eigenschaften der Auflistung zugreifen. Das erste Objekt der Auflistung ließe sich dann mit:

```
param1 = objArgs(0)
```

lesen. In obigem Fall wird dann der Variablen *param1* der Inhalt der Standardeigenschaft des Objekts *objArgs(0)* zugewiesen. Auch dies ist eine Neuerung. Eigentlich wird zum Lesen des Parameters der Name der Objekteigenschaft benötigt. Die Zeichenkette mit dem Übergabeparameter wird über die *Item*-Eigenschaft erreicht. Dies bedeutet, Sie müssen sowohl festlegen, welches Objekt der Auflistung gemeint ist als auch die gewünschte Eigenschaft angeben. Um beispielsweise den ersten Parameter des Skripts zu lesen, wäre eigentlich die folgende Anweisung erforderlich:

```
param1 = objArgs.Item(0)
```

Den meisten Objekten ist aber eine Standardeigenschaft zugeordnet. Beim *WshArguments*-Objekt ist dies die *Item*-Eigenschaft. Greifen Sie direkt auf ein Objekt zu, wird diese Standardeigenschaft direkt zurückgegeben. Die obige verkürzte Anweisung bewirkt daher die Zuweisung des Skriptparameters (der *Item*-Eigenschaft) an die Variable *param1*.

Allerdings gibt es noch eine weitere Schwierigkeit. Woher wissen Sie denn, wieviele Objekte in der Auflistung vorhanden sind? Falls der Benutzer keine Parameter beim Skriptaufbau angibt, liefert die *Arguments*-Eigenschaft keine


Objekte in der Auflistung zurück. Der Versuch eines Zugriffs auf die Objektvariable (z. B. *objArgs(0)*) löst dann einen Laufzeitfehler aus. Glücklicherweise stellt die *Arguments*-Eigenschaft im *Collection*-Objekt die Eigenschaft *Count* zur Verfügung. Diese Eigenschaft liefert Ihnen die Anzahl der Objekte in VBScript zurück. Sie könnten daher folgende Anweisungen verwenden, um auf den ersten Skriptparameter zuzugreifen:

```
If objArgs.Count >=1 Then ' mindestens ein Argument vorhanden
    Param1 = objArgs.Item(0)
End if
```

Allerdings ist diese Art des Zugriffs auf die Dauer etwas zu mühselig. Wesentlich einfacher lassen sich die Argumente in einer Schleife bearbeiten. Die folgende Anweisungssequenz zeigt Ihnen, wie Sie alle Argumente in VBScript auslesen und in einer Textvariablen speichern.

```
For I = 0 to objArgs.Count - 1 ' über alle Argumente
    text = text + objArgs(I) + vbCRLF ' hole Argument
Next
```

Natürlich können Sie innerhalb der Schleife die Parameter auch anders auswerten. Die obige Sequenz dient lediglich zur Demonstration der Vorgehensweise.

An dieser Stelle möchte ich Ihnen noch eine weitere Alternative zum Zugriff auf die Parameter per VBScript zeigen. In  Kapitel 3 wurden die Sprachkonstrukte kurz vorgestellt. In diesem Zusammenhang wurde auch die *For Each In*-Schleife erwähnt, die sich zum Zugriff auf Felder und Auflistungen hervorragend eignet. Sobald Sie also die Auflistung als *WshArguments*-Objekt in einer Variablen vorliegen haben, können Sie VBScript eigentlich die Ermittlung aller Elemente der Auflistung überlassen. Dies wird in der nachfolgenden Sequenz genutzt:

```
Set objArgs = Wscript.Arguments ' Objekt erzeugen
For Each i in objArgs ' über alle Argumente
    text = text + i + vbCRLF ' hole Argument
Next
```

Die erste Zeile holt die *Arguments*-Eigenschaft und legt eine Referenz auf die Auflistung in einer Objektvariablen ab. Anschließend weisen Sie den Interpreter über die *For Each i in objArgs*-Anweisung an, für jedes Element der Auflistung einen Durchlauf auszuführen. Die Schleifenvariable *i* enthält dabei das betreffende Objekt. In diesem Fall können Sie den Wert der Schleifenvariable *i* direkt verwenden, um den Parameter des Skripts zu lesen. Die Zuordnung:

```
Param1 = i
```

liefert sofort das Argument. Welchen der beiden Ansätze Sie verwenden, bleibt Ihrem persönlichen Geschmack überlassen.

Die komplette Lösung in VBScript

Das **Listing 6.4** faßt die obigen Ausführungen in einem kompletten Skriptprogramm zusammen. Werden beim Aufruf Parameter übergeben, gibt das Skript diese Parameter in einem Dialogfeld aus. Dabei wird jeder Parameter in einer getrennten Zeile aufgeführt (**Abbildung 6.3**).



Abbildung 6.3:
Anzeige der
Aufrufparameter

Sie müssen beim Aufruf lediglich dafür sorgen, dass das Skript auch die Parameter erhält. Hinweise zu diesem Thema finden Sie in [Kapitel 1](#).

```
'*****
' File:   Param.vbs (WSH-Beispiel in VBScript)
' Autor:  (c) G. Born
'
' Anzeige des übergebenen Parameters in einem
' Dialogfeld.
'*****

text = "Argumente" + vbCRLF + vbCRLF

Set objArgs = Wscript.Arguments    ' Objekt erzeugen
For I = 0 to objArgs.Count - 1    ' über alle Argumente
    text = text + objArgs(I) + vbCRLF ' hole Argument
Next

Wscript.Echo text ' zeige Argument per Echo
'*****
'*** Ende -> WSH powered by Günter Born ***
'*****
```

Listing 6.4:
Param.vbs

Sie finden die Beispieldatei *Param.vbs* im Ordner *\Beisp\Kap06* auf der [Hinweis](#) Begleit-CD-ROM. Die Datei *Param.vbs.lnk* im gleichen Verzeichnis besitzt bereits eine Befehlszeile mit vordefinierten Parametern. Die Datei

Param1.vbs besitzt die gleiche Funktion, benutzt aber die *For Each i In obj*-Schleife zur Ausgabe der Argumente.

Parameteranzeige in JScript

In JScript können Sie ebenfalls über die *WScript.Arguments*-Eigenschaft auf die Argumente des Skripts zugreifen. Bedingt durch die Syntax der Sprache ergeben sich allerdings einige kleinere Abweichungen. Der Zugriff auf die *Arguments*-Eigenschaft benötigt keine *Set*-Anweisung. JScript legt automatisch einen Untertyp für den übergebenen Wert an. Daher sieht die entsprechende Anweisung folgendermaßen aus:

```
var objArgs = WScript.Arguments;      // Objekt erzeugen
```

Sobald Sie das Auflistungsobjekt über die Eigenschaft geholt haben, können Sie auf die einzelnen Elemente der Collection zugreifen. Dies läßt sich mit folgenden Anweisungen tun:

```
for (var i=0; i < objArgs.length; i++) // über alle Argumente
    text = text + objArgs(i) + '\n';    // hole Argument
```

Hier wird der Inhalt der *Item*-Eigenschaft implizit (es handelt sich ja um die Standardeigenschaft des Objekts) einer Variablen *text* zugewiesen (↗ siehe auch die Ausführungen auf den vorhergehenden Seiten). Beachten Sie aber, dass Sie zum Ermitteln der Anzahl der Elemente in der Auflistung die *length*-Eigenschaft verwenden müssen. Die *Count*-Eigenschaft löst dagegen einen Laufzeitfehler aus. Das **Listing 6.5** zeigt das komplette Skriptprogramm. Werden beim Aufruf Parameter übergeben, gibt das Skript diese Parameter in einem Dialogfeld aus. Dabei wird jeder Parameter in einer getrennten Zeile aufgeführt (**Abbildung 6.3**).

Listing 6.5: *Param.js*

```
//*****
// File:   Param.js (WSH-Beispiel in JScript)
// Autor:  (c) G. Born
//
// Anzeige des übergebenen Parameters in einem
// Dialogfeld.
//*****

var objArgs;
var text = "Argumente \n";

var objArgs = WScript.Arguments;      // Objekt erzeugen

for (var i=0; i < objArgs.length; i++) // über alle Argumente
    text = text + objArgs(i) + '\n';    // hole Argument
```

```
WScript.Echo (text); // zeige Argument per Echo
WScript.Quit();
//*****
//*** Ende -> WSH powered by Günter Born ***
//*****
```

Sie finden die Beispieldatei *Param.js* im Ordner *\Beisp\Kap06* auf der Hinweis Begleit-CD-ROM.

Zugriff auf Umgebungsvariablen

Unter Windows 95/98 sowie unter Windows NT 4.0 bzw. Windows 2000 speichert das Betriebssystem verschiedene Informationen in Umgebungsvariablen. Der folgende Abschnitt demonstriert Ihnen, wie Sie auf diese Eigenschaftensvariable aus VBScript oder JScript zugreifen können.

Allgemeine Vorbemerkungen

Die Umgebungsvariable werden unter einem Namen verwaltet und enthalten Zeichenketten. Sie können den Inhalt der Umgebungsvariable im MS-DOS-Fenster abrufen, indem Sie den Befehl *Set* eingeben.

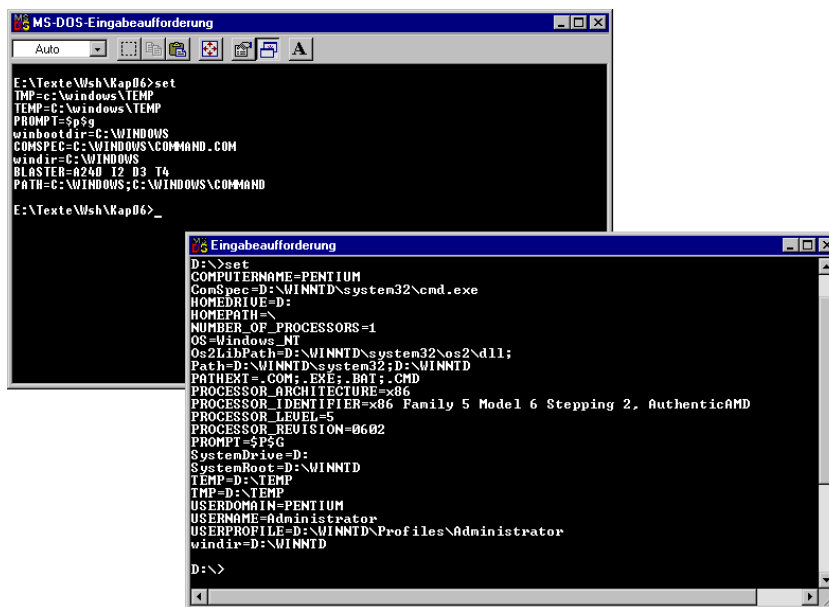


Abbildung 6.4:
Umgebungsvariablen unter Windows 95/98 (links oben) und Windows NT 4.0 (rechts unten)

In **Abbildung 6.4** sehen Sie zwei Auszüge des MS-DOS-Fensters, in denen die Umgebungsvariablen der jeweiligen Maschine angezeigt werden. Die Darstellung basiert auf einer Fotomontage zweier Bildschirmabzüge verschiedener Maschinen. Auf einer Maschine wurde Windows 98 gefahren, während die andere Maschine unter Windows NT 4.0 gestartet wurde. Die Darstellung zeigt, dass sich die Umgebungsvariable auf den jeweiligen Betriebssystemplattformen stark unterscheiden.

Die Umgebungsvariablen unter Windows 95/98 sind stark an MS-DOS orientiert. Sie finden daher nur einige wenige vom Betriebssystem vordefinierte Umgebungsvariable wie *Path*, *Prompt*, *Windir* etc.

Unter Windows NT bzw. Windows 2000 richtet das Betriebssystem verschiedene Umgebungsvariable ein, die dabei in verschiedene Kategorien unterteilt werden. So können Sie im MS-DOS-Fenster immer nur lokale Umgebungsvariable für diese MS-DOS-Maschine setzen. Das Betriebssystem ordnet dabei die Umgebungsvariable intern noch in verschiedene Kategorien »System«, »User«, »Volatile« und »Process« ein.

Ein Blick auf **Abbildung 6.4** zeigt, dass unter Windows NT wesentlich mehr Informationen als unter Windows 95/98 zur Verfügung stehen. Sie können beispielsweise den Namen des Betriebssystems, die Zahl der Prozessoren, die Plattform etc. abfragen.

Hinweis

Das Setzen der Umgebungsvariable ist im MS-DOS-Fenster nur für die aktuelle MS-DOS-Sitzung möglich. Beenden Sie die MS-DOS-Sitzung, gehen die gesetzten Umgebungsvariablen verloren. Sie haben unter Windows 95/98 aber die Möglichkeit, Umgebungsvariable beim Start über die Datei *Autoexec.bat* zu setzen. Unter Windows NT müssen Sie die globalen Umgebungsvariable über die Systemmanagement-Konsole der Systemsteuerung setzen. Dies hat natürlich auch zur Konsequenz, dass Sie per WSH-Skript keine global gültige Umgebungsvariable setzen können, die zudem noch nach dem Beenden des Skripts erhalten bleiben. Sobald ein Prozess gestartet wird, erzeugt Windows eine Kopie des UmgebungsvariablenSpeichers für diesen Prozeß. Jede Änderung bezieht sich dann nur auf diese Kopie und nicht auf den globalen Umgebungsvariablenbereich!

Zugriff auf Umgebungsvariable per Skript

Die Informationen aus den Umgebungsvariablen lassen sich beispielsweise verwenden, um in einem Skript die benutzte Rechnerplattform oder das Betriebssystem zu ermitteln. Allerdings gelten unter Windows NT beispielsweise andere Restriktionen zum Setzen bestimmter Einstellungen als unter Windows 95/98.

Zum Zugriff auf die Umgebungsvariablen der Maschine bietet das *Shell*-Objekt gemäß der Programmierreferenz die Eigenschaft *Environment*. Sie können die folgenden Anweisungen verwenden, um auf diese Eigenschaft zuzugreifen:

```
Set WshShell = CreateObject ("WScript.Shell")  
Set objEnv = WshShell.Environment("Process")
```

Die erste Zeile erzeugt eine Instanz des *WScript.Shell*-Objekts und hinterlegt eine Referenz auf dieses Objekt in der Objektvariablen *WshShell*. Anschließend können Sie auf die *Environment*-Eigenschaft dieses Objekts zugreifen.

Die WSH-Programmierreferenz ist sich nicht ganz einig, ob es sich bei *Environment* um eine Eigenschaft oder eine Methode handelt. Der betreffende Begriff wird als Eigenschaft des *Shell*-Objekts aufgeführt. Diese Eigenschaft kann verschiedene Werte (für verschiedene Umgebungsvariablen) zurückgeben. Die Syntax entspricht dabei eher dem Aufruf einer Methode. Daher spreche ich nachfolgend auch von der *Environment*-Methode. Hinweis

Bei der Verwendung der *Environment*-Methode gilt es jedoch zwei Dinge zu beachten. Einmal läßt sich ein optionaler Parameter angeben. Der Parameter legt fest, aus welchem Bereich die Umgebungsvariablen gelesen werden sollen. Fehlt diese Angabe, liefert die Methode immer Umgebungsvariablen aus dem Systembereich. Sie können im Parameter aber die Zeichenkonstanten "System", "User", "Volatile" oder "Process" angeben. Beachten Sie aber, dass diese Angabe jedoch nur unter Windows NT von Bedeutung ist. Unter Windows 95/98 unterstützt die Methode dagegen nur den Eintrag "Process".

Der zweite Punkt: Die Methode (oder die Eigenschaft) liefert ein Objekt mit einer Auflistung zurück. Sie müssen daher das Ergebnis einer Objektvariablen zuweisen, um auf die Objekte der Auflistung und letztendlich auf die Umgebungsvariablen zugreifen zu können. Dies kann über die folgende Anweisung erfolgen:

```
Text =objEnv("Path")
```

Die Anweisung setzt voraus, dass in *objEnv* die Auflistung der *Environment*-Eigenschaft hinterlegt wurde. Als Argument ist dann der Name der gewünschten Umgebungsvariable (hier *Path*) anzugeben. Die obige Zuweisung legt daher den Wert der Umgebungsvariablen *Path* in der Variablen *Text* ab.

Sie finden die Auflistung der vom System vordefinierten Umgebungsvariable sowie die Beschreibung der obigen Methoden in der WSH-Programmierreferenz im Anhang. Neben den vordefinierten Variablen können Sie aber auch auf benutzerdefinierte Umgebungsvariable (lesend) zugreifen, indem Sie deren Name in der Anweisung angeben. Die beiden nachfolgenden Beispiele zeigen, wie sich die Umgebungsvariablen in VBScript und JScript anzeigen lassen. Hinweis

Umgebungsvariablen in VBScript nutzen

In einem einfachen Beispielprogramm sollen verschiedene Umgebungsvariablen gelesen und in einem Dialogfeld ausgegeben werden. Neben vom System automatisch erstellten Umgebungsvariablen wird auch eine benutzerdefinierte Umgebungsvariable ausgelesen. Ich habe hier die Variable *Blaster* gewählt, die zur Unterstützung der Soundkarten unter MS-DOS eingerichtet wird.

Sie können das Skript sowohl unter Windows 95/98 als auch unter Windows NT 4.0 bzw. Windows 2000 aufrufen (sofern der WSH auf den Plattformen installiert ist). Je nach Betriebssystemumgebung werden aber verschiedene Umgebungsvariablen fehlen. So legt Windows 98 keine Umgebungsvariable *OS* an, um die Betriebssystemplattform anzuzeigen. Beachten Sie daher, dass die Anzeige der Umgebungsvariablen abhängig von der Betriebssystemumgebung ist.

Abbildung 6.5:
Umgebungsvariablen
(unter Windows 98)



Abbildung 6.5 zeigt die Ausgabe des Skripts unter Windows 98. Da auf dieser Plattform verschiedene Umgebungsvariable nicht definiert sind, bleiben die betreffenden Zeilen leer. Rufen Sie das Skript unter Windows NT auf, sollten beispielsweise die Einträge für *System-Drive* und *System-Root* mit Werten belegt sein.

Tip

Über eine Abfrage der Umgebungsvariable *OS* könnten Sie beispielsweise prüfen, ob das Skript unter Windows NT oder unter Windows 95/98 ausgeführt wird. Ist die Variable nicht definiert (d. h. es wird eine leere Zeichenkette zurückgegeben), wird das Skript wohl unter Windows 98 (oder Windows 95) ausgeführt. Unter Windows NT läßt sich sogar noch ermitteln, auf welcher Architektur (z. B. *x86* oder *Alpha*) das Skript läuft.

Die Programmanweisungen in **Listing 6.6** zeigen, wie auf diese Umgebungsvariable zuzugreifen ist.

```
'*****
' File:   Environment.vbs (WSH-Beispiel in VBScript)
' Autor:  (c) G. Born
'
' Zeigt den Inhalt verschiedener Umgebungsvariablen
' des WScript.Shell-Objekts in einem Dialogfeld an.
'*****

Option Explicit

Dim Message, Title
Dim WshShell, objEnv

' Hole das Shell-Objekt

Set WshShell = CreateObject ("WScript.Shell")

' Hole jetzt die Auflistung über die Environment-Eigenschaft

Set objEnv = WshShell.Environment("Process")

Message = "Umgebungsvariablen" + vbCRLF + vbCRLF
Message = Message + "Path: " + objEnv("Path") + vbCRLF
Message = Message + "Extensions: " + objEnv("PathExt") + vbCRLF
Message = Message + "Prompt: " + objEnv("Prompt") + vbCRLF

Message = Message + "System-Drive: " + objEnv("SYSTEMDRIVE") + vbCRLF
Message = Message + "System-Root: " + objEnv("SYSTEMROOT") + vbCRLF
Message = Message + "Win-Dir: " + objEnv("WinDir") + vbCRLF

Message = Message + "TEMP: " + objEnv("TEMP") + vbCRLF
Message = Message + "TMP: " + objEnv("TMP") + vbCRLF

Message = Message + "OS: " + objEnv("OS") + vbCRLF

Message = Message + "Blaster: " + objEnv("Blaster") + vbCRLF

' Titeltext initialisieren
Title = "WSH-Beispiel " + WScript.ScriptName + " - by G. Born"

MsgBox Message, vbInformation + vbOKOnly, Title

WScript.Quit() ' Jetzt wird das Skript beendet
```

Hinweis

```
*****
*** Ende -> WSH powered by Günter Born ***
*****
```

Sie finden die Datei *Environment.vbs* auf der Begleit-CD-ROM im Ordner *\Beisp\Kap06*.

So lesen Sie Umgebungsvariablen in JScript

Für die Leser, die lieber in JScript programmieren, möchte ich nachfolgend die Lösung zum Auslesen verschiedener Umgebungsvariable in dieser Sprache zeigen. Es sollen die gleichen Umgebungsvariablen wie im vorherigen Beispiel gelesen und in einem Dialogfeld ausgegeben werden (**Abbildung 6.5**). Je nach Plattform werden nicht alle Umgebungsvariablen Werte enthalten. Die Programmanweisungen in **Listing 6.7** zeigen, wie auf diese Umgebungsvariablen zuzugreifen ist. Die Struktur entspricht dem VBScript-Listing, wobei die JScript-Syntax zu berücksichtigen ist. So kennt JScript keinen *Set*-Befehl. Vielmehr ist die *var*-Anweisung zum Speichern der Objektvariablen zu verwenden.

Listing 6.7: *Environment.js*

```
//*****
// File:      Environment.js (WSH-Beispiel in JScript)
// Autor:     (c) G. Born
//
// Zeigt den Inhalt verschiedener Umgebungsvariablen
// des WScript.Shell-Objekts in einem Dialogfeld an.
//*****
//

var Message, Title, tmp;
var vbInformation = 64;
var vbOKOnly = 0;

// hole das Shell-Objekt
var WshShell = WScript.CreateObject ("WScript.Shell");

// Hole jetzt die Auflistung über die Environment-Eigenschaft

var objEnv = WshShell.Environment("Process");

// Zeige einige Umgebungsvariable

Message = "Umgebungsvariablen \n\n";
```

```

Message = Message + "Path: " + objEnv("Path") + "\n";
Message = Message + "Extensions: " + objEnv("PathExt") + "\n";
Message = Message + "Prompt: " + objEnv("Prompt") + "\n";

Message = Message + "System-Drive: " + objEnv("SYSTEMDRIVE") + "\n";
Message = Message + "System-Root: " + objEnv("SYSTEMROOT") + "\n";
Message = Message + "Win-Dir: " + objEnv("WinDir") + "\n";

Message = Message + "TEMP: " + objEnv("TEMP") + "\n";
Message = Message + "TMP: " + objEnv("TMP") + "\n";

Message = Message + "OS: " + objEnv("OS") + "\n";

// Hole eine benutzerdefinierte Umgebungsvariable
Message = Message + "Blaster: " + objEnv("Blaster") + "\n";

// Titeltext initialisieren
Title = "WSH-Beispiel " + WScript.ScriptName + " - by G. Born";

var objAdr = WScript.CreateObject("WScript.Shell");

tmp = objAdr.Popup (Message, vbInformation + vbOKOnly, Title);

WScript.Quit() ; // Jetzt wird das Skript beendet

//*****
//***      Ende -> WSH-JScript      ***
//*****

Sie finden die Datei Environment.js auf der Begleit-CD-ROM im Ordner Hinweis
\Beisp\Kap06.

```

Umgebungsvariablen setzen

Im vorhergehenden Beispiel wurde gezeigt, wie Sie gezielt auf eine Umgebungsvariable zugreifen können, um deren Wert auszulesen. Eingangs hatte ich bereits erwähnt, dass Sie per Skript keine permanente Umgebungsvariable setzen können. Die Variable bleibt nur während der Dauer der WSH-Skriptauführung für dieses Skript sichtbar. Zur Demonstration dieses Sachverhalts habe ich ein kleines Skript in VBScript erstellt. Dieses Skript zeigt beim Aufruf ein Dialogfeld, in dem nachgefragt wird, ob eine Umgebungsvariable *Born* anzulegen ist. Bestätigen Sie dieses Dialogfeld über die *Ja*-Schaltfläche, erzeugt das Skript die

Umgebungsvariable. Die entsprechenden Anweisungen sind nachfolgend aufgeführt:

```
Set WshShell = CreateObject ("WScript.Shell")
Set objEnv = WshShell.Environment("Process")
objEnv("Born") = "Hallo, WSH ist toll!"
```

Die erste Anweisung holt das *Shell*-Objekt, während die zweite Anweisung das *Environment*-Objekt des »Process«-Umgebungsbereichs erzeugt. Dies stellt sicher, dass das Skript auch unter Windows 95/98 läuft. Jetzt muß nur noch die neue Umgebungsvariable erstellt werden. Hier setzt die dritte Anweisung an, indem sie einfach die Umgebungsvariable *Born* beschreibt. Existiert die Variable noch nicht, wird sie im lokalen Umgebungsvariablenspeicher des WSH-Prozesses angelegt.

Um dem Benutzer die Wirkung zu demonstrieren, listet das Skript anschließend alle gefundenen Umgebungsvariablen auf. Hierzu wird einfach das *Environment*-Objekt geholt und über eine *For Each x In*-Schleife bearbeitet.

Hinweis

Zur Analyse der Lebensdauer können Sie das Skript starten. Dieses fragt, ob die Umgebungsvariable anzulegen ist und listet anschließend alle gefundenen Umgebungsvariablen in einem Dialogfeld auf. Lassen Sie dieses Dialogfeld geöffnet, starten das Skript ein zweites Mal und bestätigen das Dialogfeld zum Anlegen der Variablen über die *Nein*-Schaltfläche. In der zweiten Auflistung wird die Variable *Born* nicht aufgeführt. Dies zeigt, dass die vom ersten Prozeß erzeugte Umgebungsvariable lokal ist. Beenden Sie die Skripte und starten diese neu, ohne eine neue Umgebungsvariable anlegen zu lassen, fehlt die Variable *Born* ebenfalls. Dies zeigt, dass die Lebensdauer der Variablen an die Lebensdauer des Prozesses gekoppelt ist.

Die Anweisungen in **Listing 6.8** zeigen die Details der Implementierung.

Listing 6.8:
Environment1.vbs

```
'*****
' File:   Environment1.vbs (WSH-Beispiel in VBScript)
' Autor:  (c) G. Born
'
' Legt wahlweise eine Umgebungsvariable an und
' listet anschließend alle Umgebungsvariablen
' des "Process"-Bereiches in einem Dialogfeld auf.
'*****
Option Explicit

Dim Message, Title
Dim WshShell, objEnv
Dim i, tmp

' Hole das Shell-Objekt
```

```

Set WshShell = CreateObject ("WScript.Shell")

' Hole jetzt die Auflistung über die Environment-Eigenschaft

Set objEnv = WshShell.Environment("Process")

' Lege eine neue (temporäre) Umgebungsvariable an

tmp = MsgBox ("Umgebungsvariable Born anlegen ?", _
    vbYesNo + vbQuestion, _
    "WSH-Beispiel - by Günter Born")

If tmp = vbYes Then
    objEnv("Born") = "Hallo, WSH ist toll!"
End if

' Lese alle Umgebungsvariablen aus

Message = "Umgebungsvariablen" + vbCRLF + vbCRLF

For Each i In objEnv
    Message = Message + i + vbCRLF
Next

' Titeltext initialisieren
Title = "WSH-Beispiel " + WScript.ScriptName + " - by G. Born"

MsgBox Message, vbInformation + vbOKOnly, Title

WScript.Quit() ' Jetzt wird das Skript beendet

' *****
' *** Ende -> WSH powered by Günter Born ***
' *****

```

Sie finden die Datei *Environment1.vbs* auf der Begleit-CD-ROM im Ordner *Hinweis\Beisp\Kap06*.

Löschen von Umgebungsvariablen

Hinsichtlich des Löschens von Umgebungsvariablen möchte ich an dieser Stelle ebenfalls noch einige Bemerkungen anführen. Das *WScript.Shell.Environment*-Objekt unterstützt zwar die *Remove*-Methode, mit der sich eine Umgebungsvariable löschen läßt. Sie können diese Methode beispielsweise folgendermaßen einsetzen:

```
Set WSHShell = WScript.CreateObject ("Wscript.Shell")
WSHShell.Environment ("Process").Remove ("Path")
```

Diese Methode hat jedoch nur Zugriff auf die Umgebungsvariable des aktuellen Prozesses. Dies bedeutet, eine gelöschte Umgebungsvariable wird nur in der lokalen Kopie des Umgebungsbereichs entfernt. Die globalen Einstellungen für die Umgebungsvariablen werden dadurch nicht beeinflusst.

Abbildung 6.6:
Anzeige der
Umgebungsvariablen



Das nachfolgende VBScript-Listing demonstriert den Sachverhalt. Im ersten Durchlauf werden alle Umgebungsvariablen angezeigt. Dann fügt das Skript eine Variable *Born* zum Umgebungsbereich hinzu und listet das Ergebnis auf. Zum Schluß löscht das Skript die Umgebungsvariable *Born* und *Path*. Das Ergebnis wird wiederum in einem Dialogfeld angezeigt. Wenn Sie das Skript erneut starten, stellen Sie fest, dass die Umgebungsvariable *Path* wieder angezeigt wird. Das Skript besitzt also keine Möglichkeit, eine Umgebungsvariable dauerhaft zu löschen.

Listing 6.9:

```
Environment2.vbs ' *****
' File:      Environment2.vbs (WSH-Beispiel in VBScript)
' Autor:    (c) G. Born
'
' Legt wahlweise eine Umgebungsvariable an und
' listet anschließend alle Umgebungsvariablen
' des "Process"-Bereiches in einem Dialogfeld auf.
```



```

' Dann wird die Umgebungsvariable wieder gelöscht.
' *****
Option Explicit

Dim Message, Title
Dim WshShell, objEnv
Dim i, tmp

' Hole das Shell-Objekt

Set WshShell = CreateObject ("WScript.Shell")

' Hole jetzt die Auflistung über die Environment-Eigenschaft

Set objEnv = WshShell.Environment("Process")

' Titeltext initialisieren
Title = "WSH-Beispiel " + WScript.ScriptName + " - by G. Born"

' Lese alle Umgebungsvariablen aus

Message = "Umgebungsvariablen Urzustand" + vbCRLF + vbCRLF

For Each i In objEnv
    Message = Message + i + vbCRLF
Next

MsgBox Message, vbInformation + vbOKOnly, Title

' Lege eine neue (temporäre) Umgebungsvariable an

objEnv("Born") = "Hallo, WSH ist toll!"

' Alle Umgebungsvariablen auslesen

Message = "Umgebungsvariablen erweitert" + vbCRLF + vbCRLF

For Each i In objEnv
    Message = Message + i + vbCRLF
Next

```

```

MsgBox Message, vbInformation + vbOKOnly, Title

' Lösche jetzt die Umgebungsvariable

objEnv.Remove("Born")
objEnv.Remove("Path")

' Alle Umgebungsvariablen auslesen

Message = "Umgebungsvariablen nach dem Löschen" + vbCRLF + vbCRLF

For Each i In objEnv
    Message = Message + i + vbCRLF
Next

MsgBox Message, vbInformation + vbOKOnly, Title

WScript.Quit() ' Jetzt wird das Skript beendet

'*****
'*** Ende -> WSH powered by Günter Born ***
'*****

```

Hinweis

Sie finden die Datei *Environment2.vbs* auf der Begleit-CD-ROM im Ordner *\Beisp\Kap06*.

Umgebungsvariablen expandieren

Zum Abschluß möchte ich noch kurz eine weitere Technik vorstellen, die die in Umgebungsvariablen hinterlegten Informationen nutzt. Wenn Sie mit MS-DOS-Stapelverarbeitungsdateien gearbeitet haben, kennen Sie vermutlich die Möglichkeit, den Inhalt einer Umgebungsvariablen in einem Befehl zu verwenden. Die Anweisung:

```
%text%\Edit.com %document%
```

verwendet den Inhalt der Umgebungsvariablen *text* als Pfadangabe. Das zu ladende Dokument steht in der Variablen *document*. Einen ähnlichen Ansatz kennen Windows NT-Nutzer, die bereits in die Registrierung hineingeschaut haben. Häufig tauchen in Schlüsseln folgende Angaben auf:

```
%WinDir%\Notepad.exe %1
```

Diese Angabe besagt, dass anstelle von *%WinDir%* das Windows-Verzeichnis einzusetzen ist. Ist Windows im Verzeichnis *C:\Windows* installiert, ergibt sich der Pfad:

```
C:\Windows\notepad.exe %1
```

Die Zeichen *%1* stellen übrigens den Platzhalter für die aktuell in der Shell gewählte Datei dar. Hinweis

Der Vorteil bei der Verwendung der *%WinDir%*-Variablen besteht darin, dass ein solcher Aufruf auch dann funktioniert, wenn der Benutzer beispielsweise sein Betriebssystem auf einem anderen Laufwerk oder in einem anderen Ordner installiert hat.

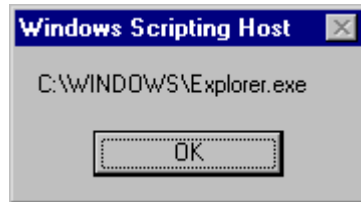
Als Skriptprogrammierer stehen Sie vermutlich häufiger vor dem Problem, auf bestimmte Pfadangaben aufsetzen zu müssen, ohne sicher sein zu können, dass absolute Laufwerksangaben oder Ordnernamen global gelten. Möchten Sie beispielsweise unabhängig vom Windows-Installationverzeichnis sein, könnten Sie auf die Umgebungsvariable *WinDir* zurückgreifen. In dieser Variablen hinterlegt Windows 95/98 beispielsweise die Pfadangabe zum Windows-Verzeichnis. Wie erhält ein Skriptprogrammierer aber die Information, welches Verzeichnis für Windows benutzt wird? Oder wie kann er den Inhalt einer anderen Umgebungsvariablen in einem Ausdruck verwenden?

Die vordergründige Lösung besteht darin, die obigen Methoden und Eigenschaften zu nutzen, um den Inhalt der Umgebungsvariablen zu lesen und anschließend in den gewünschten Befehl einzubauen. Bevor Sie sich aber mit solchen Experimenten beschäftigen, möchte ich Ihnen einen einfacheren Weg demonstrieren. Microsoft hat dem *Shell*-Objekt die *ExpandEnvironmentStrings*-Methode mitgegeben. Diese Methode erwartet eine Zeichenkette als Argument. Enthält diese Zeichenkette den in *%*-Zeichen eingefassten Namen einer Umgebungsvariable, ersetzt die Methode diesen Namen durch den Inhalt der Variablen. Anschließend wird der expandierte Wert zurückgegeben. Sofern Sie also den Namen einer Umgebungsvariablen in einer Anweisung angeben, lässt sich deren Inhalt sehr leicht über die Methode expandieren. Dies soll anhand von **Listing 6.10** demonstriert werden. Im Skript wird ein Befehl der Art:

```
Command = "%Windir%\Explorer.exe" ' zu expandierender Befehl
```

vereinbart. Das Skript soll nun dafür sorgen, dass die angegebenen Umgebungsvariable *%WinDir%* expandiert wird. Das Ergebnis wird in diesem Fall in einem Dialogfeld ausgegeben (**Abbildung 6.7**). Genauso gut können Sie den expandierten Befehl aber in einem *Run*-Aufruf einfügen und das Programm aufrufen. Allerdings ist dies bei der *Run*-Methode nicht erforderlich, da diese Umgebungsvariablen in der Form *%...%* automatisch expandiert.

Abbildung 6.7:
Anzeige der
expandierten
Zeichenkette



Weitere Details sind dem nachfolgenden Listing zu entnehmen.

Listing 6.10:
Environment3.vbs

```

'*****
' File:   Environment3.vbs (WSH-Beispiel in VBScript)
' Autor:  (c) G. Born
'
' Demonstriert die Anwendung der ExpandEnvironmentStrings-
' Methode des WScript.Shell-Objekts.
'*****
Option Explicit

Dim Command
Dim WshShell

Command = "%Windir%\Explorer.exe" ' zu expandierender Befehl

' Hole das Shell-Objekt

Set WshShell = WScript.CreateObject("WScript.Shell")

' Jetzt kann die ExpandEnvironmentStrings-Methode genutzt werden

WScript.Echo WshShell.ExpandEnvironmentStrings(Command)

WScript.Quit() ' Jetzt wird das Skript beendet

'*****
'*** Ende -> WSH powered by Günter Born ***
'*****

```

Hinweis

Sie finden die Datei *Environment3.vbs* auf der Begleit-CD-ROM im Ordner *\Beisp\Kap06*.

Objekte holen und freigeben

In den vorhergehenden Abschnitten und Kapiteln wurde bereits mehrfach auf Objekte und deren Methoden bzw. Eigenschaften zugegriffen. Sofern der WSH die Objekte nicht automatisch zur Verfügung stellt (wie beispielsweise das *WScript*-Objekt) müssen Sie diese zunächst »erzeugen«. Der nachfolgende Abschnitt befaßt sich noch etwas näher mit dieser Fragestellung.

Objekte instanziiieren

Der Zugriff auf eine Objekt setzt voraus, dass dieses Objekt im aktuellen Skript auch bekannt ist. Konkret benötigen Sie eine Referenz auf das betreffende Objekt. Diese Referenz läßt sich über die *CreateObject*-Methode erzeugen. In VBScript sieht dies folgendermaßen aus:

```
Set Objektvariable = Objekt.CreateObject("ProgID")
```

während in JScript die folgende Syntax gilt:

```
var Objektvariable = Objekt.CreateObject("ProgID");
```

Die obigen Anweisungen wurden recht allgemein gehalten. Daher möchte ich noch einige zusätzliche Erläuterungen geben. Um auf ein Objekt zuzugreifen, benötigen Sie eine Referenz auf dieses Element. Der Windows Scripting Host stellt Ihnen nur eine einzige Referenz in Form der *WScript*-Variable automatisch zur Verfügung. Alle anderen Referenzen müssen Sie über die obigen Anweisungen explizit definieren. Zur Speicherung wird dabei die als *Objektvariable* bezeichnete Variable benutzt. Sie können für diese Variable beliebige, aber gültige Namen verwenden. Die Bezeichner *objAdr*, *Born*, *WshObj* etc. stellen gültige Namen für Objektvariablen dar.

Hinweis Beginn

In den ↗ Kapiteln 3 und 4 wurde bereits erwähnt, dass VBScript bzw. JScript keine eigenen Datentypen unterstützen sondern bei Variablen *Variant*-Datentypen verwenden. Lediglich das Format zur Speicherung trägt den unterschiedlichen Werten Rechnung. Sofern Sie mit der *CreateObject*-Methode arbeiten, liefert diese einen Wert vom Unterdatentyp *Object* in die angegebene Variable zurück. Aber im Grunde müssen Sie sich um diesen Sachverhalt nicht kümmern.

Hinweis

Als Programmierer haben Sie aber die Möglichkeit, im Variablennamen einen Hinweis auf den Unterdatentyp unterzubringen. Meist arbeitet man mit einem Prefix (*c* steht beispielsweise für *character*, d. h. für Zeichenketten). Häufig benutze ich daher die Nomenklatur *Wsh...* um die Verbindung zum Windows

Scripting Host anzudeuten. Oder ich setze die Buchstabenfolge *obj* vor den Namen, um einen Hinweis auf die Objektvariable zu geben.

Hinweis Ende

Methoden lassen sich nur auf Objekte anwenden. Die *CreateMethode* macht hier keine Ausnahme. Um ein Objekt im Skript zu nutzen, wird in der Regel das *WScript*-Objekt angegeben. Um beispielsweise auf das *Shell*-Objekt zuzugreifen, ist in VBScript die Anweisung:

```
Set objShell = WScript.CreateObject("WScript.Shell")
```

erforderlich. Die JScript-Anweisung sieht entsprechend aus, Sie müssen nur *var* und das abschließende Semikolon benutzen. Werfen wir noch einen zweiten Blick auf die obige Anweisung (für alle, denen der Sachverhalt noch nicht ganz klar ist). Mit *WScript.CreateObject* wird dem Interpreter mitgeteilt, dass wir ein Objekt »haben möchten«, welches als Unterobjekt von *WScript* benutzt wird. In der Klammer wird ein Argument an die *CreateObject*-Methode übergeben. Dieses Argument gibt die sogenannte *ProgID* an, die Sie zum Zugriff auf das betreffende Objekt kennen müssen. In obigem Fall wurde die *ProgID* für das *Shell*-Objekt mit *"Wscript.Shell"* angegeben.

Hinweis


Der schwierigste Schritt für den »Einsteiger« besteht darin, die richtigen *ProgID*-Namen für die Objekte zu kennen. In den Beispielen dieses Buches wird diese *ProgID* mit angegeben. Die *ProgIDs* fremder Anwendungen finden Sie in der jeweiligen Programmdokumentation. Fehlt diese, können Sie auf Werkzeuge wie den in ↗ Kapitel 2 vorgestellten *OLE/COM Object Viewer* zugreifen. Auch der Objektkatalog der VB-Entwicklungsumgebungen liefert Hinweise auf die *ProgID*.

Lassen Sie sich auch nicht durch die Tatsache verwirren, dass in den obigen Anweisungen zweimal der Begriff »WScript« vorkommt. Bei *Wscript* in *WScript.CreateObject* handelt es sich um ein Objekt. Das im Argument der Methode übergebene *"WScript.Shell"* stellt dagegen den Namen einer Bibliothek sowie den Namen des gewünschten Objekts dar. Dass beidemal *WScript* vorkommt, ist reiner Zufall, der sich aus den Namenkonventionen bei der Benennung der Bibliotheken ergibt. Detailliertere Hintergrundinformationen finden Sie in folgendem Abschnitt.

Wichtig

Bei der Anwendung der *CreateObject*-Methode ist vor allem wichtig, dass Sie die Objekthierarchie nutzen. In ↗ Kapitel 5 wurde in der Einführung bereits erwähnt, dass Objekte weitere Objekte enthalten können. Die Abhängigkeit der Objekte untereinander wird dabei durch die Objekthierarchie oder das Objektmodell spezifiziert. Eine Anwendung wie Word kann dann ein Dokument enthalten. Um auf das *Document*-Objekt zuzugreifen, muß die *CreateObject*-Methode erst auf das *Application*-Objekt und dann auf das *Document*-Objekt angewandt werden. Sie finden auf den nachfolgenden Seiten Beispiele für den Zugriff auf solche Objekte.

Hinweise zu Objekreferenzen

Allen interessierten Leser möchte ich an dieser Stelle noch einige Hintergrundinformationen geben. Sobald Sie die *CreateObject*-Methode anwenden, nutzen Sie recht »heftig« die Möglichkeiten des von Microsoft entwickelten COM-Ansatzes (COM steht für Component Object Model). Idee hierbei ist, dass die Funktionen in Form von Objekten im System bereitgestellt werden. Konkret liegen diese Objekte aber nicht »irgendwie« im Rechner herum. Vielmehr werden die Objekte in Bibliotheken (genauer: in Type Libraries) bereitgestellt. Es handelt sich dabei um Dateien, die neben dem ausführbaren Code für das Objekt und seine Methoden noch zusätzliche Beschreibungen der Schnittstellen enthalten. Damit die *CreateObject*-Methode funktioniert, muß der Programmierer genau angeben, wo das Objekt und dessen Beschreibung zu finden ist. Theoretisch hätte es jetzt die Möglichkeit gegeben, ein Laufwerk und einen Pfad auf die Bibliotheksdatei als Argument zu übergeben (was bei der *GetObject*-Methode auch genutzt wird). Dies hat aber den Nachteil, dass jede Änderung des Pfads einen Fehler in der *CreateObject*-Methode auslöst. Die Microsoft-Entwickler haben sich daher für einen wesentlich smarteren Ansatz entschieden: Eine Objektbibliothek, die als EXE-, DLL- oder OCX-Datei vorliegen kann, darf mehrere Objekte enthalten. Um die Objekte nutzen zu können, muß der Inhalt der Bibliothek registriert werden. In  Kapitel 2 wurden bereits einige Ansätze beschrieben, mit denen diese Registrierung erfolgen kann. Bei diesem Schritt hinterlegt das Installationsprogramm alle Informationen hinsichtlich der verfügbaren Komponenten in der Windows-Registrierung. Wenn Sie diese im Zweig *HKEY_CLASSES_ROOT\CLSID* enthaltenen Einträge mit dem *OLE/COM Object Viewer* inspizieren, finden Sie dort sowohl die *ProgID* als auch den Pfad zum sogenannten Inprocess-Server. Dies ist die Datei, die die eigentlichen Funktionen bereitstellt.

Um nun ein Objekt zu »erzeugen«, rufen Sie die *CreateObject*-Methode auf und geben als Argument den Namen des gewünschten Objekts an. Dieser Name besteht aus zwei Komponenten:

`TypeLibName.ClassName`

Der erste vor dem Punkt stehende Teil gibt den Namen der Type-Bibliothek an. In den obigen Beispielen wurde beispielsweise die *WScript*-Type-Bibliothek benutzt. Andere Type-Bibliotheken werden beispielsweise von Word ("*Word*"), Excel ("*Excel*") oder von ActiveX-Komponenten (z. B. "*WSHExtend*") zur Verfügung gestellt. Innerhalb der Type-Bibliothek werden dann verschiedene Elemente für die externe Nutzung bereitgestellt. Diese »Elemente« sind dabei unter Ihrem Namen in einer sogenannten Klasse (bzw. Unterklasse) registriert. In der *WScript*-Type-Bibliothek gibt es beispielsweise die Klasse *Shell*. Als Argument der *CreateObject*-Methode müssen Sie daher den Begriff *WScript.Shell* angeben. Bei Excel wird

beispielsweise *Excel.Application* verwendet, um das Anwendungsobjekt zu holen. Beim Internet Explorer ist *InternetExplorer.Application* zu verwenden.

Sobald der Interpreter die *CreateObject*-Methode ausführt, ermittelt diese über die *ProgID* die Lage der Bibliotheksdateien aus der Registrierung. Dann wird das Objekt als Kopie in den Speicher geladen (instanziiert) und die Exportschnittstelle des Objekts an das Skript gebunden. Von diesen Schritten bekommen Sie allerdings nichts mit. Die *CreateObject*-Methode hinterlegt lediglich die Referenz auf das Objekt in der angegebenen Objektvariable. Anschließend können Sie diese Objektvariable benutzen, um auf das Objekt zuzugreifen und dessen Eigenschaften und Methoden zu nutzen.

[Einschub Ende](#)

CreateObject oder GetObject anwenden?

Um ein neues Objekt anzulegen bzw. auf ein Objekt zuzugreifen, bietet Ihnen der WSH zwei Methoden: *CreateObject* und *GetObject*. Vielleicht stellt sich Ihnen die Frage, wann Sie welche Methode anwenden sollen und wo die Unterschiede liegen.

Die *CreateObject*-Methode erzeugt das im *strProgID*-Parameter spezifizierte Objekt. Auf den vorherigen Seiten haben Sie nur den vereinfachten Aufruf mit einem Parameter kennengelernt. Die *CreateObject*-Methode erlaubt aber die Angabe eines zweiten optionalen Parameters:

```
Set objAdr = WScript.CreateObject(strProgID [,strPrefix])
```

Der erste Parameter enthält die bereits erwähnte *ProgID* des Objekts (z. B. *"WScript.Shell"*). Im optionalen zweiten Parameter *strPrefix* können Sie eine Zeichenkette hinterlegen, die als Prefix für eine Funktion im Skript dient. Dies erlaubt die Nutzung von Callback-Funktionen, d. h. das Objekt kann Funktionen des Skripts aufrufen. Auf diese Technik komme ich zu einem späteren Zeitpunkt noch zurück (wenn wir den Internet Explorer zur Formulareingabe nutzen).

Objekte lassen sich auch mit der *WScript.GetObject*-Methode instanziiieren. Diese Methode besitzt eine ähnliche Aufrufsyntax:

```
Set objAdr = WScript.GetObject(strPathname [, [strProgID] [,strPrefix]])
```

Die *GetObject*-Methode lädt entweder das Objekt aus einer Datei oder erzeugt das im *strProgID*-Parameter spezifizierte Objekt. Aus diesem Grund weist diese Methode drei Parameter auf. In *strPathname* ist der Pfad und der Name der Datei anzugeben, aus der das Objekt zu importieren ist. Der optionale Parameter *strProgID* definiert eine Zeichenkette, die den Programmbezeichner (ProgID) des Objekts angibt. Die Methode kann dann das Objekt auch über die Registrierung laden. Im dritten Parameter kann die Methode optional die Exportschnittstelle des Objekts an die Skriptdatei

anbinden. Löst das Objekt ein Ereignis aus, ruft der Windows Scripting Host eine Prozedur mit dem angegebenen Prefix auf. Diese Technik lernen Sie in einem späteren Kapitel kennen.

Verwenden Sie die *GetObject*-Methode, falls die aktuelle Objektinstanz bereits existiert, oder falls Sie das Objekt aus einer Datei erzeugen möchten, die bereits geladen ist. Gibt es keine aktuelle Objektinstanz und möchten Sie das Objekt nicht aus einer bereits geladenen Datei erzeugen, ist die *CreateObject*-Methode zu verwenden.

Hinweis Beginn

Einige Objekt können sich so registrieren, dass nur eine Instanz geladen wird. Dies ist beispielsweise beim *Word.Basic*-Objekt in Microsoft Word 7.0 der Fall. Dann wird durch *CreateObject* immer nur eine Instanz erzeugt, unabhängig von der Anzahl der Aufrufe.

Hinweis

Bei der *GetObject*-Methode läßt sich dagegen festlegen, ob eine bestehende Instanz oder eine neue Instanz anzulegen ist. Rufen Sie ein solches »Single-Instance«-Objekt mit einem Leerstring ("") im Parameter *strPathName* auf, gibt die Methode immer die gleiche Instanz des Objekts zurück:

```
Set x = GetObject ("", "Born.Test")
```

Hier würde die Instanz des Objekts *Test* aus der Bibliothek *Born* zurückgegeben. Allerdings sollten Sie wissen, dass die WSH-Skript-Engine in der Version 3.1 einen Fehler aufweist. Dieser Fehler bewirkt, dass auch *GetObject* in der obigen Form eine neue Instanz anlegt. In der Version 3.1a wurde dieser Fehler aber behoben.

Lassen Sie übrigens den *Pfad*-Parameter weg, tritt laut der WSH-Programmierreferenz ein Laufzeitfehler auf. Das gleiche gilt, falls das angegebene Objekt nicht existiert. Sie können *GetObject* nicht verwenden, um eine Referenz zu einer Visual Basic-Klasse zu erhalten, falls diese mit Visual Basic 4.0 (oder einer früheren Version) erzeugt wurde.

Hinweis Ende

Freigabe mit DisconnectObject

Haben Sie ein Objekt mit *GetObject* oder *CreateObject* in den Speicher geladen, bleibt das Objekt mit dem WSH bis zum Beenden des Skripts verbunden (und damit im Speicher). Sofern Sie ein Objekt aber nicht mehr benötigen, sollten Sie dieses freigeben. Hierzu ist die *DisconnectObject*-Methode zu verwenden. Hierzu ist die folgende Anweisung:

```
WScript.DisconnectObject Objektname
```

zu nutzen. Bei Bedarf können Sie anschließend noch die Objektvariable mit:

```
Objectname = Nothing
```

zurücksetzen. Das folgende kurze Listing zeigt den Einsatz dieser Methoden:

Listing 6.11:
Disconnect.vbs

```
'*****
' File:   Disconnect.vbs (WSH-Beispiel in VBScript)
' Autor:  Günter Born
'
' Zweck:  Zeigt den Einsatz der DisconnectObject-Methode.
'*****

Option Explicit

DIM WshShell, tmp

Set WshShell = Wscript.CreateObject ("WScript.Shell")

tmp = WshShell.Popup ("Test", 0, "WSH-Test by Günter Born")

WScript.DisconnectObject WshShell
Set WshShell = Nothing

WScript.Quit ()

'*****
'***           Ende -> WSH-VBScript           ***
'*****
```

Hinweis

Sie finden das Programm *Disconnect.vbs* im Ordner *\Beisp\Kap06* auf der Begleit-CD-ROM.

Programmaufrufe per WSH

Der nachfolgende Abschnitt zeigt Ihnen, wie Sie die *Run*-Methode einsetzen können, um aus einem Skript heraus andere Anwendungen zu starten.

Hinweise zur *Run*-Methode

Um aus dem WSH-Skript heraus eine andere Anwendung zu starten, müssen Sie die *Run*-Methode des *Shell*-Objekts einsetzen. Die *Run*-Methode erzeugt einen neuen Prozess, der den in *strCommand* enthaltenen Befehl ausführt und den Fensterstil aus *intWindowStyle* benutzt. Allgemein besitzt die Methode die folgenden Form des Aufrufs:

```
WshShell.Run (strCommand [, [intWindowStyle] [,bWaitOnReturn]])
```

Der Parameter *strCommand* muß angegeben werden, da hier der Pfad und der Name der auszuführenden Anwendung oder des auszuführenden Befehls hinterlegt wird. Als Besonderheit sei noch angemerkt, dass die Methode automatisch im Argument enthaltene Umgebungsvariable expandiert. Hinterlegen Sie beispielsweise die Befehle:

```
Set WshShell = WScript.Shell ("WScript.Shell")
Command = "%WinDir%\Calc.exe"
WshShell.Run (Command)
```

in einem Skript, expandiert die Methode die Umgebungsvariable im Befehl und startet dann den Windows-Rechner. Über die optionalen Parameter können Sie noch steuern, wie die Anwendung zu öffnen ist und ob das Skript auf das Beenden der Anwendung warten soll.

Der Parameter *intWindowStyle* ist optional und legt den Fensterstil fest, den die gestartete Anwendung unter Windows erhält. Der Wert des Variant-Arguments ist eine Ganzzahl zwischen 0 und n. Wird *intWindowStyle* nicht angegeben, erhält das Programmfenster den Fokus und wird im normalen Zustand gestartet.

Leider fehlt in der Microsoft WSH-Programmierreferenz die Beschreibung der zulässigen Werte für den Parameter. Im ersten Schritt habe ich daher die Parameter für den *Shell*-Befehl der VBA-Entwicklungsumgebung verwendet. Die *vb*-Konstanten der **Tabelle 6.1** sind dieser Umgebung entnommen. Zu einem späteren Zeitpunkt stellte ich fest, dass die Microsoft-Website ein Dokument mit der Beschreibung der *Run*-Methode enthält. Dort ist ausgeführt, dass der Parameter das *wShowWindow*-Element in der *STARTUPINFO*-Struktur des neuen Prozesses setzt. Daher entsprechen die zulässigen Parameter dem *nCmdShow*-Parameter der *ShowWindow*-Funktion. Basierend auf diesen Überlegungen sind die Werte für *intWindowStyle* in **Tabelle 6.1** aufgeführt. Die in der Spalte *Konstante* aufgeführten benannten *vb*-Konstanten sind der VBA-Hilfe entnommen. Die *wm*-Konstanten stammen aus der *STARTUPINFO*-Struktur. Beachten Sie aber, dass diese Konstanten unter VBScript nicht definiert sind. Zu Deutsch: Sie müssen die numerischen Konstanten einsetzen.

Hinweis

Konstante	Wert	Beschreibung
vbHide SW_HIDE	0	Das Fenster wird ausgeblendet, und ein anderes Fenster wird eingeblendet (d. h. es erhält den Fokus).
vbNormalFocus SW_SHOWNORMAL	1	Aktiviert das Fenster und zeigt dieses an. War der Prozeß bereits aktiv und das Fenster minimiert/maximiert, wird die

Tabelle 6.1:
Werte für
intWindowStyle

		ursprüngliche Größe und Position wiederhergestellt.
vbMinimizedFocus SW_SHOWMINIMIZED	2	Das Fenster wird aktiviert und als Symbol mit Fokus in der Taskleiste angezeigt.
vbMaximizedFocus SW_SHOWMAXIMIZED	3	Das Fenster wird aktiviert und maximiert mit Fokus angezeigt.
vbNormalNoFocus SW_SHOWNOACTIVATE	4	Die zuletzt verwendete Größe und Position des Fensters wird wiederhergestellt. Das momentan aktive Fenster bleibt aktiv.
SW_SHOW	5	Aktiviert das Fenster und zeigt es in der aktuellen Größe und Position an.
vbMinimizedNoFocus SW_MINIMIZE	6	Das Fenster wird minimiert und als Symbol angezeigt. Das in z-Richtung oberste Fenster wird aktiviert.
SW_SHOWMINNOACTIVE	7	Das Fenster wird als Symbol angezeigt. Das momentan aktive Fenster bleibt aktiv.
SW_SHOWNA	8	Zeigt das Fenster in seinem aktuellen Zustand. Das momentan aktive Fenster bleibt aktiv.
SW_RESTORE	9	Aktiviert das Fenster und zeigt es an. War das Fenster minimiert oder maximiert, restauriert Windows die Fenstergröße. Eine Anwendung sollte dieses Flag benutzen, um ein minimiertes Fenster zu restaurieren.

Hinweis

Die Werte in **Tabelle 6.1** zeigen, dass Sie mit der *Run*-Methode sowohl eine Instanz einer Anwendung aufrufen als auch eine bereits laufende Anwendung in den Vordergrund schalten können. Leider gibt es bei der *Run*-Methode ein (kleines) Problem: Diese erzeugt immer eine neue Instanz des Prozesses. Es ist daher nicht möglich, das Fenster einer laufenden Anwendung zu minimieren, sprich: nicht alle Werte der Tabelle machen im Skript Sinn. Bei Tests ist mir auch aufgefallen, dass dies nur dann funktioniert, wenn die entsprechende Anwendung diese Fensterattribute auch unterstützt. Wenn Sie beispielsweise den Windows-Editor verwenden, klappt das Setzen des Parameters für den Fensterstil. Beim Windows-Rechner bleibt der Parameter wirkungslos, da dieses Programm nicht alle Fensterattribute erlaubt (der Rechner lässt sich beispielsweise nicht maximieren).

Der optionale Parameter *bWaitOnReturn* besitzt den Unterdatentyp *Logical* und steuert, ob das Skript auf die Beendigung des aufgerufenen Prozesses wartet. Ist *bWaitOnReturn* nicht spezifiziert oder auf *False* gesetzt, kehrt die Methode sofort zur Ausführung des Skripts zurück und wartet nicht auf die Beendigung des Prozesses. Ist *bWaitOnReturn* auf *True* gesetzt, wartet die *Run*-Methode auf das Ende des gestarteten Prozesses. Anschließend liefert die *Run*-Methode den von der Anwendung zurückgegebenen Fehlercode (error code) zurück. Fehlt der Parameter *bWaitOnReturn* oder ist der Wert auf *False* gesetzt, liefert die *Run*-Methode den Fehlercode (error code) 0 (zero) zurück.

Den Fehlercode können Sie beim Beenden eines Skripts über die *Quit*-Hinweis Methode setzen.

Aufrufen einer Windows-Anwendung

Die Erkenntnisse im vorherigen Abschnitt sollen jetzt in einigen kleinen Skripten umgesetzt werden.

Den Editor aus VBScript aufrufen

Der Windows-Editor unterstützt die in der **Tabelle 6.1** aufgeführten Werte für den Fensterstil. Das folgende Skript zeigt, wie Sie den Editor *Notepad.exe* aus VBScript aufrufen. Die Lage des Windows-Ordners wird über die Umgebungsvariable *%WinDir%* spezifiziert. Daher wird dieses Skript unabhängig vom gewählten Installationsverzeichnis von Windows laufen. In einem zweiten Schritt startet das Skript den Editor erneut. Hierbei wird das Fenster als Schaltfläche in der Taskleiste angezeigt. Zusätzlich lädt der Editor bei diesem Aufruf den Quellcode des aktuellen Skripts.

```
' *****
' File:    Run.vbs (WSH-Beispiel in VBScript)
' Autor:   Günter Born
'
' Zweck:  Aktiviert den Windows-Editor über den
' Run-Befehl.
' *****

DIM WshShell

Set WshShell = WScript.CreateObject ("WScript.Shell")

WshShell.Run "%Windir%\Notepad.exe", 1

WScript.Echo "Quellcode im minimierten Editorfenster laden"
```

Listing 6.12:
Run.vbs

```

WshShell.Run "%Windir%\Notepad.exe " + WScript.ScriptFullName, 6

WScript.Quit()


'*****
'***           Ende -> WSH-VBScript           ***
'*****

```

Hinweis

Sie finden das Programm *Run.vbs* im Ordner *\Beisp\Kap06* auf der Begleit-CD-ROM.

Den Windows-Rechner aus JScript aufrufen

In diesem Beispiel habe ich JScript verwendet, um den Windows-Rechner aufzurufen. Die Lage des Windows-Ordners wird über die Umgebungsvariable *%WinDir%* spezifiziert. Daher wird dieses Skript unabhängig vom gewählten Installationsverzeichnis von Windows laufen. In diesem Beispiel habe ich die benannten Konstante für die Fensterstile im Programmkopf definiert. Beachten Sie aber, dass der Rechner nicht alle Fensterstile unterstützt. Der Aufruf eines minimierten Fensters wird daher nicht funktionieren. Bei der Umsetzung des VBScript-Beispiels auf JScript müssen Sie neben den Syntaxregeln der Sprache auch daran denken, dass die Parameter der *Run*-Methode in Klammern zu setzen sind. Als weiteres sind die *Backslash*-Zeichen in der Pfadangabe als ** einzugeben, da das **-Zeichen in JScript eine Escape-Sequenz einleitet. Dies bedeutet, der Interpreter weist dem Folgezeichen eine besondere Bedeutung zu. Um ein Backslash-Zeichen in der Pfadangabe zu erhalten, müssen Sie daher – wie in  Kapitel 4 beschrieben – die Form ** wählen. Einzelheiten finden Sie in nachfolgendem Listing.

Listing 6.13: *Run.js*

```

//*****
// File:   Run.js (WSH-Beispiel in JScript)
// Autor:  Günter Born
//
// Zweck:  Aktiviert den Windows-Rechner über
// den Run-Befehl. Achtung: Diese Anwendung
// unterstützt nicht alle Fensterstile!
//*****

var SW_SHOWNORMAL = 1;
var SW_MINIMIZE = 6;

var WshShell = WScript.CreateObject ("WScript.Shell");

```

```
// Jetzt den ersten Versuch unternehmen
WshShell.Run ("%Windir%\Calc.exe", SW_SHOWNORMAL);

WScript.Echo ("Der Rechner wird jetzt minimiert gestartet");

// Dies dürfte nicht funktionieren
WshShell.Run ("%Windir%\Calc.exe", SW_MINIMIZE);

WScript.Quit();

//*****
//***      Ende -> WSH-JScript      ***
//*****
```

Sie finden das Programm *Run.js* im Ordner *\Beisp\Kap06* auf der Begleit-CD- Hinweis ROM.

Warten auf den Prozeß und Auswerten des Rückgabecodes

Im nächsten Beispiel möchte ich noch den Aufruf einer anderen Anwendung demonstrieren, wobei das Skript auf das Beenden der Anwendung wartet. Gibt diese Anwendung einen Rückgabecode zurück, zeigt das Skript diesen Code in einem Dialogfeld an. Zur Simulation des Prozesses wurde das folgende kurze VBScript-Programm verwendet. Das Programm zeigt über die *Echo*-Methode ein Dialogfeld an. Erst wenn der Benutzer dieses Dialogfeld über die *OK*-Schaltfläche beendet, terminiert das Skript. Sie können also sehr gut verfolgen, ob das »Master«-Skript auf das Beenden des *Test*-Skripts wartet. Beim Terminieren gibt das *Test*-Skript dann den Fehlercode 2 über die *Quit*-Methode an Windows zurück. In der *Quit*-Methode sind übrigens Fehlerwerte zwischen 0 und 255 zulässig. Das **Listing 6.14** zeigt den Aufbau des *Test*-Skripts.

```
' *****
' File:   Test.vbs (WSH-Beispiel in VBScript)
' Autor:  Günter Born
'
' Zweck:  Gebe einen Fehlercode zurück.
' *****

WScript.Echo "Test-Skript", vbCRLF, _
            "Wir geben den Fehlercode 2 zurück"
WScript.Quit (2)

' *****
' ***      Ende -> WSH-VBScript      ***
' *****
```

Listing 6.14:
Test.vbs

```
'*****
```

Nun benötigen wir noch das eigentliche Skript, welches das Testprogramm aufruft, auf dessen Beendigung wartet und dann den Fehlercode in einem Dialogfeld anzeigt. Das folgende Listing zeigt den Aufbau des Skripts in VBScript. Als kleine Besonderheit möchte ich noch auf die Funktion *GetPath* verweisen. Das Skript muß ja den Pfad des Test-Skripts kennen. Hier bin ich davon ausgegangen, dass die Datei *Test.vbs* im gleichen Ordner wie das Programm *Run1.vbs* gespeichert ist. Folglich können wir über *WScript.ScriptFullName* dessen Pfad ermitteln. In *GetPath* wird anschließend der Dateiname vom Pfad entfernt, es liegt der Pfad zum Skriptordner vor. Dieser Pfad wird um den Namen *Test.vbs* erweitert und dient dann zum Aufruf des Skripts aus VBScript.

Das Beispiel demonstriert daher auch, wie Sie ein Skript aus einem zweiten Skript aufrufen können. Weiterhin zeigt die Ausgabe des Fehlercodes auch, dass ein aufgerufenes Skript ein Ergebnis (eine Zahl zwischen 0 und 255) an das rufende Skript zurückgeben kann. Dies reicht zumindest für einfache Statusinformationen.

Beim Aufruf der *Run*-Methode müssen Sie daran denken, den dritten Parameter auf *true* zu setzen. Dies bewirkt, dass das Skript auf das Beenden des aufgerufenen Prozesses wartet. Weiterhin möchten Sie den Rückgabewert auswerten. Daher ist die Methode in Form einer Funktion anzuwenden:

```
ErrCode = WshShell.Run (name, 1, true)
```

Einzelheiten sind dem **Listing 6.15** zu entnehmen. Das Skript ist so einfach gehalten, dass keine weiteren Erläuterungen notwendig werden.

Listing 6.15:
Run1.vbs

```
'*****
' File:   Run1.vbs (WSH-Beispiel in VBScript)
' Autor:  Günter Born
'
' Zweck:  Aktiviert einen zweiten Prozeß in Form eines
' Skripts, wartet auf dessen Beendigung und zeigt dann
' den Rückgabecode an.
'*****
Option Explicit

DIM WshShell, name, ErrCode

' Das Objekt wird für die Run-Methode benötigt
Set WshShell = WScript.CreateObject ("WScript.Shell")

' Hole jetzt den Pfad zum auszuführenden Programm. Hier
' muß das Skript im gleichen Ordner wie Run1.vbs liegen.
```



```

name = GetPath + "Test.vbs"

errcode = WshShell.Run (name, 1, true)    ' Wir warten!

WScript.Echo ErrCode

WScript.Quit()

Function GetPath
' Ermitteln den Pfad des Skripts und entferne den
' Skriptnamen -> geben Pfad zurück.
DIM path
path = WScript.ScriptFullName ' Name Skript
GetPath = Left(path, InstrRev(path, "\"))
End Function

'*****
'***           Ende -> WSH-VBScript           ***
'*****

Sie finden die beiden Programme Test.vbs und Run1.vbs im Ordner Hinweis
\Beisp\Kap06 auf der Begleit-CD-ROM.

```

MS-DOS-Befehle per RUN ausführen

Zum Abschluß möchte ich noch kurz zeigen, wie Sie die *Run*-Methode verwenden können, um MS-DOS-Befehle auszuführen. Sofern Sie ein MS-DOS-Programm wie beispielsweise den MS-DOS-Editor aufrufen möchten, ist dies kein Problem. Sie gehen wie bei Windows-Anwendungen vor:

```
WshShell.run ("%windir%\Command\Edit.com");
```

Die obige Anweisung öffnet das Fenster des MS-DOS-Editors. Hängen Sie hinter den Programmnamen noch den Namen der Dokumentdatei an, öffnet der Editor diese automatisch.

Möchten Sie die Eigenschaften des MS-DOS-Programmes beeinflussen? **Tip**
Dann erzeugen Sie eine PIF-Datei für die MS-DOS-Anwendung (siehe /5/ im Literaturverzeichnis), indem Sie die MS-DOS-Programmdatei mit der rechten Maustaste anklicken und im Kontextmenü den Befehl *Eigenschaften* wählen. Setzen Sie die gewünschten Eigenschaften für das MS-DOS-Fenster. Sobald Sie die Registerkarten über die *OK*-Schaltfläche schließen, erzeugt Windows im Ordner der Programmdatei eine PIF-Datei. Diese besitzt den Namen des Programmes aber die (nicht angezeigte) Dateinamenerweiterung *.pif*. Wenn Sie anschließend die Anwendung aufrufen, lädt Windows die PIF-Datei,

wertet die Umgebungseinstellungen aus und aktiviert dann die MS-DOS-Anwendung mit diesen Einstellungen.

Aber kommen wir zurück zur Eingangsfrage. Manchmal möchte man ja einen MS-DOS-Befehl wie *Copy*, *Dir* etc. aus einem Skript heraus aufrufen. Kann ich diesen Befehl auf die gleiche Art in der *Run*-Methode angeben, wie dies mit anderen Programmen möglich ist? Die Anweisung:

```
WshShell.Run "Dir C:\"
```

wird nicht funktionieren. Das Problem: Es gibt keine Programmdatei *Dir.com* oder *Dir.exe*, die *Run*-Methode setzt aber auf Windows-Funktionen auf, die eine ausführbare Programmdatei (*exe*, *com*, *bat*, *pif*) erwartet. Die internen MS-DOS-Befehle wie *Dir*, *Copy*, *Rename* etc. sind (im Gegensatz zu den externen Befehlen wie *Edit*, die als *Com*-Dateien vorliegen) im MS-DOS-Befehlsprozessor *Command.com* integriert. Diese Datei ist im Windows-Ordner zu finden. Um also einen MS-DOS-Befehl auszuführen, müssen Sie diesen Kommandoprozessor aufrufen und diesem den auszuführenden Befehl übergeben. Der Aufruf zur Anzeige eines Verzeichnisinhalts sieht folgendermaßen aus:

```
C:\Windows\Command.com /k dir *.*
```

Es wird vorausgesetzt, das Windows im Ordner *C:\Windows* installiert wurde. Wichtig ist der Schalter */k*, der die Ausführung des Befehls bewirkt, ohne das MS-DOS-Fenster zu schließen. Möchten Sie, dass das MS-DOS-Fenster automatisch nach dem Ausführen des Befehls geschlossen wird, verwenden Sie statt dessen den Schalter */c*.

Tip

Eine Auflistung der für *Command.com* verfügbaren Optionen erhalten Sie übrigens, indem Sie auf der MS-DOS-Befehlszeilenebene die Anweisung *command /?* eintippen.

Das folgende kleine in JScript verfaßte Skript zeigt den Inhalt des Windows-Ordners über den *Dir*-Befehl an. Als Besonderheit habe ich in diesem Beispiel die Befehle zum Ausführen der gewünschten Anweisung in Variablen hinterlegt. Die Variable *command* stellt beispielsweise die Zeichenkette zum Aufruf des MS-DOS-Befehlsprozessors bereit. Die Variable *dos_command* dient zur Aufnahme des gewünschten MS-DOS-Befehls. Und in der Variablen *option* können Sie zusätzliche Optionen hinterlegen, die nichts mit dem MS-DOS-Befehl zu tun haben. In diesem Beispiel habe ich in *option* das *|*-Zeichen zum Aufruf des MS-DOS-Filters sowie den *more*-Befehl hinterlegt. Dies bewirkt, dass der MS-DOS-Befehlsprozessor die Ausgaben des *Dir*-Befehls an einen Filter *More* weiterleitet. Dieser Filter gibt die Ausgaben seitenweise im MS-DOS-Fenster aus.

Tip

Auf ähnliche Weise können Sie das Umleitungszeichen *>* nutzen, um die Ausgaben des *Dir*-Befehls beispielsweise zum Drucker oder in eine Datei umzuleiten. Auf diese Weise könnten Sie sich beispielsweise einen Befehl zum Drucken eines Verzeichnisses schaffen. Die im Literaturverzeichnis

unter /1,2,5/ aufgeführten Titel erläutern, wie Sie einen solchen Befehl in der Windows-Shell registrieren. Dann können Sie Verzeichnisinhalte per Kontextmenü ausdrucken.

Das **Listing 6.16** zeigt, wie sich dieses Wissen in einem JScript-Skriptprogramm nutzen läßt.

Listing 6.16:
RunDos.js

```
//*****
// File:   RunDOS.js (WSH-Beispiel in JScript)
// Autor:  Günter Born
//
// Zweck:  Demonstriert, wie sich ein MS-DOS-Befehl
// über die Run-Methode ausführen läßt.
//*****

var command, dos_command, option
// erzeuge Shell-Objekt für die Run-Methode

var WshShell = WScript.CreateObject ("WScript.Shell");

// Jetzt setzen wir einen Befehl ab, der den Inhalt
// des Windows-Verzeichnisses anzeigen soll.

// Befehlsprozessor aufrufen
command = "%windir%\\command.com /k ";

// hier folgt der eigentliche MS-DOS-Befehl,
dos_command = "dir " + "%windir%";

// Sie können noch Optionen anhängen:
// | more      erzwingt eine seitenweise Anzeige
// > PRN:      leitet die Ausgaben an den Drucker
// > Dir.txt leitet die Ausgabe in eine Textdatei
//
option = "|more"; // seitenweise Anzeige

// Befehl ausführen
WshShell.Run (command + dos_command + option);

WScript.Quit(0);

//*****
//***      Ende -> WSH-JScript      ***
```

//*****

Hinweis Sie finden das Programm *RunDos.js* auf der Begleit-CD-ROM im Ordner *\Beisp\Kap06*.

Tip Einige Leute berichteten in Newsgroups von Problemen bei der Ausführung der *Run*-Methode in einem Skript, welches unter Windows NT über den *AT*-Befehl. Das Skript wurde niemals beendet sondern »hing«. In diesem Fall müssen Sie im Aufruf des *AT*-Kommandos das */Interactive*-Flag angeben. Im nächsten Kapitel lernen Sie alternative Methoden zum Starten und Aktivieren von Programmen kennen.

Windows Scripting Host-Objektreferenz

A

WSH-Objektmodell	524
<i>WScript</i> -Objekt	524
<i>WshArguments</i> -Objekt	534
<i>WshShell</i> -Objekt	536
<i>WshNetwork</i> -Objekt	545
<i>WshCollection</i> -Objekt	551
<i>WshEnvironment</i> -Objekt	553
<i>WshShortcut</i> -Objekt	556
<i>WshSpecialFolders</i> -Objekt	560
<i>WshUrlShortcut</i> -Objekt	562

In diesem Anhang finden Sie eine Referenz des WSH-Objektmodells mit der Beschreibung aller Methoden und Eigenschaften.

- ♦ Schlagen Sie nach, wenn Sie etwas zum WSH-Objektmodell wissen möchten.
- ♦ Lesen Sie, welche Eigenschaften das *WScript*-Objekt, das *WshShell*-Objekt oder das *WshArguments*-Objekt bieten.

Beispiele zeigen Ihnen, wie Sie die betreffenden Eigenschaften und Methoden anwenden.

WSH-Objektmodell

Das Objektmodell des Windows Scripting Host bietet zwei Hauptkategorien für ActiveX-Schnittstellen:

- ♦ **Skriptausführung und Fehlersuche:** Diese Kategorie umfaßt Eigenschaften und Methoden, die direkt mit der Skriptausführung zu tun haben. Dieser Set an Schnittstellen ermöglicht den Skripts die Manipulation des Windows Scripting Host, die Anzeige von Meldungen auf dem Bildschirm sowie Basisfunktionen wie *CreateObject* und *GetObject*.
- ♦ **Hilfsfunktionen:** Eigenschaften und Methoden, die Netzwerklaufwerke zuweisen, Verbindungen zu Drucker einrichten, Funktionen, die Umgebungsvariablen einrichten bzw. ändern, und Funktionen zur Manipulation von Registrierungseinträgen. Mittels dieser Funktionen können Administratoren den Windows Scripting Host verwenden, um einfache Anmeldeskripte auszuführen.

Um verschiedene Aufgaben unter Windows durchzuführen, kann ein Administrator zusätzlich zu den vom Windows Scripting Host angebotenen Objekten jedes andere ActiveX-Steuerelement verwenden, welches Automatisierungsschnittstellen exportiert. Beispielsweise kann ein Administrator Skripte schreiben, die die Active Directory Services von Windows NT verwenden.

WScript-Objekt

Die Script-Engine *WScript.exe* stellt dem Skript ein Objekt unter dem Objektnamen *WScript* zur Verfügung. Nachfolgend finden Sie eine Übersicht über die jeweiligen Eigenschaften und Methoden, die dieses Objekt unterstützt.

Eigenschaften des *WScript*-Objekts

Das *WScript*-Objekt wird direkt durch die *WScript.exe*-Datei dem Skript zur Verfügung gestellt und besitzt verschiedene Eigenschaften. Diese Eigenschaften beziehen sich direkt auf den WSH bzw. auf das ausgeführte Skript und lassen sich in einem Skript direkt abrufen. Die nachfolgende Tabelle listet diese Eigenschaften auf.

Eigenschaft	Beschreibung
Application	Beschreibt das <i>IDispatch</i> -Interface für <i>WScript</i> .
Arguments	Es handelt sich um ein <i>Parameters</i> -Auflistungsobjekt, über welches sich auf die dem Skript übergebenen Parameter zugreifen läßt.
FullName	Diese Eigenschaft liefert den Pfad zur Programmdatei des Hosts einschließlich des Namens der EXE-Datei (meist <i>WScript.exe</i>).
Name	Liefert den Namen von WScript im Klartext (als Friendly Name in der Form »Windows Scripting Host«). Es handelt sich um die Standardeigenschaft des Objekts.
Path	Enthält den Namen des Ordners, in dem <i>WScript.exe</i> oder <i>Cscript.exe</i> enthalten ist. Im Gegensatz zu <i>FullName</i> liefert diese Eigenschaft nicht den Namen des Hosts.
ScriptFullName	Diese Eigenschaft enthält den Pfad zum Skript, welches vom Windows Scripting Host ausgeführt wird.
ScriptName	Dateiname des Skripts, welches durch den Windows Scripting Host ausgeführt wird.
Version	Ein Text mit der Version des Windows Scripting Host.

Tabelle A.1:
WScript-
Eigenschaften

Beachten Sie, dass sich diese Eigenschaften nicht setzen lassen, d. h. Sie können die Eigenschaftswerte nicht durch eine Zuweisung ändern. Die nachfolgenden Abschnitte enthalten detailliertere Hinweise zu diesen Eigenschaften. Hinweis

WScript.Application

Die Eigenschaft *Application* liefert das *IDispatch*-Interface des *WScript*-Objekts.

Syntax:

WScript.Application = objWscript

Um beispielsweise den Inhalt der *Application*-Eigenschaft anzuzeigen, können Sie folgende VBScript-Anweisungen verwenden:

```

Txt = "Application: " + WScript.Application
MsgBox Txt

```

Die beiden Anweisungen lesen die *Application*-Eigenschaft des *WScript*-Objekts in die Variable *Txt* und geben dann die Variable über *MsgBox* in

einem Dialogfeld aus. Das Dialogfeld wird die Meldung »Windows Scripting Host« liefern.

WScript.Arguments

Die *Arguments*-Eigenschaft liefert Ihnen die beim Aufruf des Skripts übergebenen Parameter. Die Eigenschaft wird vom Objekt aber als ein *WshArguments*-Auflistungsobjekt zur Verfügung gestellt.

Syntax:

WScript.Arguments = objArguments

Da es sich um eine Auflistung handelt, können Sie die Eigenschaft nicht so einfach abfragen. Vielmehr müssen Sie alle Eigenschaften der Auflistung abfragen. Die folgende Sequenz an VBScript-Anweisungen zeigt Ihnen alle Befehlszeilenparameter an.

```
' Anzeige aller Befehlszeilenparameter
Set objArgs = WScript.Arguments ' Objekt erzeugen
For I = 0 to objArgs.Count - 1 ' Über alle Argumente
    WScript.Echo objArgs(I) ' Zeige Argument per Echo-Methode
Next
```

In der ersten Anweisung wird das Objekt zur Aufnahme der Auflistung erzeugt. Anschließend lassen sich die einzelnen Parameter in einer Schleife abfragen. Der Zugriff auf die Auflistung erfolgt dabei über den Schleifenindex, der von 0 bis *Count - 1* läuft.

WScript.FullName

Die Eigenschaft *FullName* liefert eine Zeichenkette zurück, die den Pfad sowie den Namen für den aktuellen Host angibt.


Syntax:

WScript.FullName = strFullName

Die folgenden Zeilen zeigen, wie sich das Objekt in einem Programm nutzen läßt.

```
Txt = "FullName: " + WScript.FullName
MsgBox Txt
```

Die beiden Anweisungen lesen die *FullName*-Eigenschaft des *WScript*-Objekts in die Variable *Txt* ein und geben dann die Variable über *MsgBox* in einem Dialogfeld aus. Das Dialogfeld wird eine Meldung der Art »C:\WINDOWS\WSCRIPT.EXE« enthalten.

Die  *Path*-Eigenschaft liefert Ihnen dagegen lediglich den Pfad zum WSH Hinweis zurück.

WScript.Name

Die *Name*-Eigenschaft liefert eine Zeichenkette, die den vollständigen Namen (Friendly Name) des *WScript*-Objekts enthält. Dies ist die Standardeigenschaft des Objekts, d. h. der Eigenschaftsname muß nicht angegeben werden, um die Eigenschaft zu manipulieren.

Syntax:

```
WScript.Name = strName
```

Das folgende Beispiel ermittelt die *Name*-Eigenschaft und gibt diese in einem Meldungsfeld aus:

```
Txt = "Name: " + WScript.Name  
MsgBox Txt
```

Die erste Zeile bewirkt, dass der Variablen *Txt* die *Name*-Eigenschaft zugewiesen wird. Der Inhalt der Variablen wird anschließend in einem Meldungsfeld ausgegeben. Das Meldungsfeld liefert den Text »Windows Scripting Host«.

WScript.Path

Die *WScript.Path*-Eigenschaft liefert eine Zeichenkette mit dem Namen des Ordners zurück, in dem *WScript.exe* oder *Cscript.exe* abgelegt sind.


Syntax:

```
WScript.Path = strPath
```

Die folgende Sequenz gibt diesen Pfad in einem Dialogfeld zurück:

```
Txt = WScript.Path  
MsgBox Txt
```

Die erste Anweisung liest die Eigenschaft in die Variable *Txt* ein, die dann in der Folgezeile über die *MsgBox*-Funktion angezeigt wird. Unter Windows 98 enthält das Meldungsfeld in der Regel den Text »C:\Windows«, da der WSH im Windows-Ordner installiert wird.

Über die  *FullName*-Eigenschaft des *WScript*-Objekts läßt sich sowohl der Hinweis Pfad als auch der Name des Hosts abfragen.

WScript.ScriptFullName

Die Eigenschaft *ScriptFullName* liefert den kompletten Pfad zum gerade vom Windows Scripting Host ausgeführten Skript.

Syntax:


WScript.ScriptFullName = strScriptFullName

Die folgenden Anweisungen zeigen, wie diese Eigenschaft genutzt wird:

```
Txt = "Ausgeführtes Skript: " + WScript.ScriptFullName  
MsgBox Txt
```

Befindet sich das Skript *Test.vbs* beispielsweise im Ordner *C:\Test*, wird das Meldungsfeld den Text »C:\Test\Test.vbs« aufweisen.

Hinweis

Benötigen Sie lediglich den Namen der Skriptdatei, ist die Anweisung  *WScript.ScriptName* zu verwenden.

WScript.ScriptName

Die Eigenschaft *ScriptName* liefert lediglich den Dateinamen jener Skriptdatei zurück, die gerade im Windows Scripting Host ausgeführt wird.

Syntax:


WScript.ScriptName = strScriptName

Die folgende Anweisungssequenz gibt diesen Namen in einem Meldungsfeld aus:

```
Txt = WScript.ScriptName  
MsgBox Txt
```

Sie können diese Eigenschaft nutzen, um beispielsweise dem Benutzer einen Hinweis auf das ausgeführte Skript zu liefern.

Hinweis

Benötigen Sie auch den Pfad der Datei, ist die  *ScriptFullName*-Eigenschaft des *WScript*-Objekts zu verwenden.

WScript.Version

Der WSH besitzt eine bestimmte Version, die sich über die *Version*-Eigenschaft des *WScript*-Objekts abfragen läßt.

Syntax:

WScript.Version = strVersion

Die folgende Anweisung zeigt die Anwendung der Eigenschaft:

```
Txt = WScript.Version  
MsgBox Txt
```

Die Eigenschaft *Version* liefert eine Zeichenkette mit der Versionsnummer des Windows Scripting Host. Bei der in Windows 98 ausgelieferten WSH-Version wird der Text »5.0« ausgegeben.

Methoden des *WScript*-Objekts

Neben den auf den vorherigen Seiten beschriebenen Eigenschaften lassen sich noch verschiedene Methoden auf das *WScript*-Objekt anwenden. Die folgende Tabelle beschreibt die Methoden des *WScript*-Objekts.

Methode	Beschreibung
CreateObject	Erzeugt ein Objekt und richtet die Ereignisbearbeitung ein.
DisconnectObject	Hebt die Verbindung eines vorher mit dem Windows Scripting Host verbundenen Objekts auf.
Echo	Zeigt die Parameter in einem Fenster oder in der Befehlszeile des Befehlszeilenfensters an.
GetObject	Holt ein Automatisierungsobjekt aus einer Datei.
Quit	Beendet die Skriptausführung mit einem spezifizierten Fehlercode.

Tabelle A.2:
*Methoden des
WScript-Objekts*

Die nachfolgenden Abschnitte liefern Ihnen weitere Hinweise zu den einzelnen Methoden.

WScript.CreateObject

Die *CreateObject*-Methode erzeugt ein Objekt, welches im *strProgID*-Parameter spezifiziert wird. Für die *CreateObject*-Methode gilt folgende Syntax:

Syntax:

```
WScript.CreateObject(strProgID [,strPrefix]) = objObject
```

Die Methode erwartet zwei Parameter. Der erste Parameter muß angegeben werden und enthält den Namen des betreffenden Objekts. Der zweite Parameter ist optional. Ist der Parameter *strPrefix* vorhanden, verbindet der Windows Scripting Host nach dem Erzeugen des Objekts dessen Exportschnittstelle mit der Skriptdatei (d. h. das Objekt läßt sich anschließend in der Skriptdatei nutzen). Löst ein Objekt ein Ereignis aus, ruft der Windows

Scripting Host eine Ereignisprozedur auf, dessen Name aus *strPrefix* und dem Ereignisnamen zusammengesetzt ist.

Ist *strPrefix* beispielsweise "MYOBJ_" und das Objekt löst das Ereignis mit dem Namen *OnBegin* aus, ruft der Windows Scripting Host das Unterprogramm *MYOBJ_OnBegin* innerhalb des Skripts auf. Die folgende Anweisung zeigt den Einsatz dieser Methode:

```
Set WSHShell = WScript.CreateObject("WScript.Shell")
```

In diesem Beispiel wird *CreateObject* auf das *Shell*-Objekt angewandt. Die Methode liefert eine Referenz auf die neue Objektvariable in die Variable *WSHShell* zurück.

Hinweis

Objekte lassen sich auch mit der ↗ *WScript.GetObject*-Methode und mit der ↗ *WScript.DisconnectObject*-Methode manipulieren.

WScript.DisconnectObject

Die *DisconnectObject*-Methode hebt die Verbindung eines Objekts zum Windows Scripting Host auf (d. h. ein an den Windows Scripting Host angebundenes Objekt wird wieder entfernt). Ist das Objekt noch nicht mit den Windows Scripting Host verbunden, besitzt die Methode keine Auswirkungen.

Syntax:

WScript.DisconnectObject obj

Das nachfolgende Beispiel zeigt den Einsatz dieser Methode:

```
' Dieses Codefragment instantiiert ein fiktives Objekt und
' verbindet es mit der Skriptdatei. Im Skript wird dann die
' "SomeMethod"-Methode auf das Objekt angewandt.
' Tritt ein Fehler mit dem Namen "Event" im Objekt auf, wird
' das MyEvent_Event-Unterprogramm aufgerufen.
Set MyObject = WScript.CreateObject("SomeObject", "MyEvent")
MyObject.SomeMethod
Sub MyEvent_Event(strName)
WScript.Echo strName
End Sub
' Sobald wir fertig sind: Verbindung des Objekts zum Skript aufheben
' und auf Nothing setzen
WScript.DisconnectObject MyObject
Set MyObject = Nothing
```

Hinweis

Die Methoden ↗ *WScript.CreateObject* und ↗ *WScript.GetObject* erlauben ebenfalls die Manipulation des neuen Objekts.

WScript.Echo

Die *Echo*-Methode zeigt die Parameter in einem Fenster (in *WScript.exe*) oder bei der Benutzung der Befehlszeilenversion im Befehlszeilenfenster (in *CScript.exe*). Hierbei gilt folgende Syntax:

Syntax:

WScript.Echo [anyArg...]

Hinter dem Methodennamen lassen sich mehrere optionale Parameter angeben. Die Parameter werden durch jeweils ein Leerzeichen getrennt. In *Cscript.exe* gibt diese Methode eine Carriage-Return/Line Feed (CR/LF)-Zeichenfolge aus, nachdem der letzte Parameter angezeigt wurde. Die nachfolgenden Anweisungen verdeutlichen den Einsatz dieser Methode:

```
WScript.Echo
WScript.Echo 1, 2, 3
WScript.Echo "Der Windows Scripting Host ist toll."
```

WScript.GetObject

Die *GetObject*-Methode erzeugt entweder ein Objekt aus einer Datei oder das im *strProgID*-Parameter spezifizierte Objekt. Hierbei gilt die folgende Syntax:

Syntax:

WScript.GetObject(strPathname [,strProgID , [strPrefix]]) = objObject

Die Parameter der *GetObject*-Methode sind folgendermaßen definiert:

<i>strPathname</i>	Gibt den Pfad und den Namen der Datei an, aus der das Objekt zu importieren ist. Der Parameter <i>strPathname</i> sollte angegeben werden.
<i>strProgID</i>	Eine Zeichenkette, die den Programmbezeichner (ProgID) des Objekts angibt.
<i>strPrefix</i>	<p>Wird der Parameter <i>strPrefix</i> angegeben, bindet der Windows Scripting Host die Exportschnittstelle des Objekts nach dem Anlegen des Objekts an die Skriptdatei an. Löst das Objekt ein Ereignis aus, ruft der Windows Scripting Host ein Unterprogramm auf. Der Name des Unterprogramms setzt sich aus dem in <i>strPrefix</i> enthaltenen Namen und dem Ereignisnamen zusammen.</p> <p>Ist <i>strPrefix</i> beispielsweise "MYOBJ_" und das Objekt löst das Ereignis "OnBegin" aus, ruft der Windows Scripting Host das im Skript enthaltene Unterprogramm "MYOBJ_OnBegin" auf.</p>

	auf.
--	------

Die Methode liefert eine Referenz auf das betreffende Objekt zurück. Verwenden Sie die *GetObject*-Methode, falls die aktuelle Objektinstanz bereits existiert oder falls Sie das Objekt aus einer Datei erzeugen möchten, die bereits geladen ist. Gibt es keine aktuelle Objektinstanz und möchten Sie das Objekt nicht aus einer bereits geladenen Datei erzeugen, ist die *CreateObject*-Methode zu verwenden.

Hinweis

Hat sich ein Objekt selbst als Single-Instance-Objekt registriert (zum Beispiel das *Word.Basic*-Objekt in Microsoft Word 7.0), wird nur eine Instanz des Objekts erzeugt (unabhängig von der Zahl der *CreateObject*-Aufrufe). Bei einem Single-Instance-Objekt gibt *GetObject* beim Aufruf mit einem Leerstring ("") immer die gleiche Instanz des Objekts zurück. Weiterhin tritt ein Fehler auf, falls der Pfad-Parameter weggelassen wird. Sie können *GetObject* nicht verwenden, um eine Referenz zu einer Visual Basic-Klasse zu erhalten, falls diese mit Visual Basic 4.0 (oder einer früheren Version) erzeugt wurde.

GetObject arbeitet mit allen COM-Klassen, unabhängig von der Sprache, die zum Erzeugen des Objekts benutzt wurde. Die folgende Anweisung zeigt den Einsatz der Methode:

```
Dim MyObject
Set MyObject = GetObject("C:\CAD\SCHEMA.CAD", "MyCAD.Application")
```

Bei der Ausführung dieses Codes wird die mit *strPathname* verknüpfte Anwendung gestartet und das Objekt in der spezifizierten Datei aktiviert. Enthält der Parameter *strPathname* eine leere Zeichenkette (""), gibt *GetObject* eine neue Objektinstanz des angegebenen Typs zurück. Wird der *strPathname*-Parameter weggelassen, gibt *GetObject* ein aktuell aktives Objekt des angegebenen Typs zurück. Existiert kein Objekt des angegebenen Typs, tritt ein Laufzeitfehler auf.

Einige Anwendungen erlauben es, Teile einer Datei zu aktivieren. Für diesen Zweck müssen Sie an den Dateinamen ein Ausrufezeichen (!) gefolgt von einer Zeichenkette, die den Teil der zu aktivierenden Datei angibt, anhängen (dies ist z. B. beim Zugriff auf Excel-Tabellen der Fall). Informationen zum Aufbau der Zeichenkette finden Sie in der Dokumentation der Anwendung, die dieses Objekt erzeugt. In einer Zeichenanwendung gibt es beispielsweise mehrere Zeichenebenen innerhalb einer Datei. Sie könnten beispielsweise folgenden Code benutzen, um eine Zeichenebene in der Datei *Schema.cad* zu aktivieren:

```
Set LayerObject = GetObject("C:\CAD\SCHEMA.CAD!Layer3")
```

Geben Sie die Objektklasse nicht an, bestimmt COM die zu startende Anwendung sowie das zu aktivierende Objekt in Abhängigkeit von dem von Ihnen angegebenen Dateinamen. Einige Dateien unterstützen jedoch mehr als eine Klasse eines Objekts. Beispielsweise kann eine Zeichnung (d. h., die zugehörige Anwendung) drei unterschiedliche Objekttypen unterstützen: Ein

Application-Objekt, ein *Drawing*-Objekt und ein *Toolbar*-Objekt. Alle Objekte gehören zur gleichen Datei. Um festzulegen, welches Objekt innerhalb der Datei zu aktivieren ist, verwenden Sie den optionalen *Class*-Parameter:

```
Dim MyObject As Object
Set MyObject = GetObject("C:\DRAWINGS\SAMPLE.DRW", "FIGMENT.DRAWING")
```

In obigem Beispiel ist FIGMENT der Name der Anwendung zum Zeichnen, und DRAWING ist einer der von der Anwendung unterstützten Objekttypen.

Wurde das Objekt einmal zur Laufzeit aktiviert, können Sie im Code über die Objektvariable auf dieses Objekt zugreifen. Im obigen Beispiel läßt sich die Objektvariable *MyObject* verwenden, um auf die Objekteigenschaften zuzugreifen:

```
MyObject.Line 9, 90
MyObject.InsertText 9, 100, "Hello, world."
MyObject.SaveAs "C:\DRAWINGS\SAMPLE.DRW"
```

Neben der *GetObject*-Methode kennt das *WScript*-Objekt auch die Methoden *CreateObject* und *DisconnectObject*. Hinweis

WScript.Quit

Die *Quit*-Methode beendet die Ausführung des Skripts mit einem spezifizierten Fehlercode. Hierbei gilt folgende Syntax:

Syntax:

```
WScript.Quit [(intErrorCode)]
```

Als optionalen Parameter *intErrorCode* läßt sich ein Fehlercode angeben. Ist der Parameter vorhanden, verwendet WScript den Inhalt des Parameters als Process Exit-Code. Fehlt *intErrorCode*, liefert WScript den Wert (0) als Process Exit-Code zurück. Die Anweisung:

```
WScript.Quit(1)
```

bewirkt beispielsweise, dass der Exit-Code 1 an den Host zurückgeliefert wird. Dieser Exit-Code läßt sich beispielsweise aus MS-DOS heraus über die ERRORLEVEL-Variable auswerten.

WshArguments-Objekt

Das Objekt ist nicht direkt nutzbar. Ein Zugriff auf das Objekt ist nur über die *WScript.Arguments*-Eigenschaft möglich. Die folgende Tabelle beschreibt die Eigenschaften, die über das *WshArguments*-Objekt verfügbar sind.

Tabelle A.3:
Eigenschaften des
WshArguments-
Objekts

Eigenschaft	Beschreibung
Item	Der n-te Parameter in der Befehlszeile (Standardeigenschaft).
Count	Zahl der Parameter in der Befehlszeile.
length	Die Zahl der Befehlszeilenparameter (JScript).

Eigenschaften des *WshArguments*-Objekts

Die nachfolgenden Abschnitte geben Ihnen weitere Hinweise auf die jeweiligen Eigenschaften des *WshArguments*-Objekts.

WshArguments.Item

Die *Item*-Eigenschaft enthält den *natIndexth*-Befehlszeilenparameter als Zeichenkette. Es handelt sich hierbei um die Standardeigenschaft des Objekts. Daher können Sie sowohl über die Anweisung:

Syntax:

`Arguments(natIndex)`

als auch über die Anweisung:

`Arguments.Item(natIndex)`

auf die Eigenschaft zugreifen. Die Eigenschaft liefert dann die Zeichenkette mit dem jeweiligen Parameter. Die folgenden Anweisungen zeigen den Einsatz der Eigenschaft in VBScript:

```
Set oArgs = Wscript.Arguments ' Hole die Arguments-Eigenschaft
Wscript.Echo oArgs(0)         ' Erster Parameter
Wscript.Echo oArgs.Item(0)    ' Zweiter Parameter
```

Hinweis

Informationen zur ➤ *WScript.Arguments*-Eigenschaft finden Sie weiter oben. Die Eigenschaften ➤ *Count* und ➤ *length* liefern Ihnen Hinweise, ob und wieviele Parameter beim Skriptaufruf angegeben wurden.

WshArguments.Count

Die Eigenschaft *Count* gibt die Zahl der Befehlszeilenparameter an. Ist der Wert der Eigenschaft 0, liegen keine Befehlszeilenparameter vor.

Syntax:

Arguments.Count = natNumberOfArguments

Die Zahl der Parameter lässt sich mit folgender Anweisung ermitteln:

Zahl = Arguments.Count

Die *Arguments.Count*-Eigenschaft lässt sich über das Objekt der *WScript.Arguments*-Eigenschaft lesen. Beachten Sie, dass in JScript die *WshArguments.length*-Eigenschaft zu verwenden ist.

WshArguments.length

Die *length*-Eigenschaft gibt die Zahl der Befehlszeilenparameter an. Diese Eigenschaft liefert den gleichen Wert wie die *Count*-Eigenschaft und wird aus Kompatibilitätsgründen zu Microsoft JScript benutzt. Es gilt folgende Syntax:

Syntax:

Arguments.length = natNumberOfArguments

Weitere Hinweise finden Sie unter der *WScript.Arguments*-Eigenschaft [Hinweis](#) und unter der *WshArguments.Count*-Eigenschaft [Hinweis](#)

WshShell-Objekt

Vom Windows Scripting Host wird das *WshShell*-Objekt zur Verfügung gestellt. Dieses Objekt besitzt die ProgId *WScript.Shell* und wird über die Datei *WSHOM.ocx* realisiert. Dieses Objekt stellt sowohl verschiedene Eigenschaften als auch mehrere Methoden zur Verfügung. Nachfolgend werden diese Eigenschaften und Methoden beschrieben.

Eigenschaften des WshShell-Objekts

Die folgende Tabelle beschreibt die Eigenschaften, die mit dem *WshShell*-Objekt assoziiert sind.

Eigenschaft	Beschreibung
Environment	Gibt das WshEnvironment-Collection-Objekt zurück.
SpecialFolders	Erlaubt den Zugriff auf Ordner der Windows Shell wie <i>Desktop</i> -Ordner, <i>Startmenü</i> -Ordner den Ordner <i>Eigene Dateien</i> über das WshShell-Objekt.

Tabelle A.4:
*Eigenschaften des
WshShell-Objekts*

Nachfolgend erhalten Sie weitere Hinweise zu diesen Eigenschaften.

WshShell.Environment

Die Eigenschaft *Environment* liefert das *WshEnvironment*-Objekt zurück.

Syntax:

WshShell.Environment ([strType]) = objWshEnvironment

In *strType* wird angegeben, wo die gewünschte Umgebungsvariable liegt. Zulässige Werte sind "System", "User", "Volatile" und "Process". Wird *strType* nicht angegeben, erhält diese Methode unter Windows NT die System-Umgebungsvariable und in Windows 95/98 die Prozess-Umgebungsvariable. In Windows 95/98 wird nur "Process" im *strType*-Parameter akzeptiert. Die folgenden Variablen werden durch das Windows-Betriebssystem unterstützt. Ein Skript kann aber auch Umgebungsvariable lesen, die durch andere Anwendungen gesetzt wurden.

Tabelle A.5:
Abfragbare
Umgebungsvariablen


Name	Beschreibung
NUMBER_OF_PROCESSORS	Anzahl der auf dem lokalen Rechner benutzten Prozessoren.
PROCESSOR_ARCHITECTURE	Prozessortyp der Arbeitsstation des Benutzers.
PROCESSOR_IDENTIFIER	Prozessor-ID der Arbeitsstation des Benutzers.
PROCESSOR_LEVEL	Prozessor-Level der Arbeitsstation des Benutzers.
PROCESSOR_REVISION	Prozessor-Version der Arbeitsstation des Benutzers.
OS	Betriebssystem-Version auf der Arbeitsstation des Benutzers.
COMSPEC	Ausführbarer Befehl für den Aufruf der Befehlszeile (typischerweise <i>cmd.exe</i> bzw. <i>command.com</i>).
HOMEDRIVE	Primäre lokale Festplatte (typischerweise Laufwerk C:).
HOMEPATH	Standardverzeichnis für Benutzer (in Windows NT ist dies typischerweise <i>\users\default</i>).
PATH	PATH-Umgebungsvariable
PATHEXT	Erweiterungen für ausführbare Dateien

	(typischerweise <i>.com</i> , <i>.exe</i> , <i>.bat</i> oder <i>.cmd</i>).
PROMPT	Command-Prompt (typischerweise <i>\$p\$g</i>).
SYSTEMDRIVE	Lokales Laufwerk, auf dem das System-Verzeichnis gespeichert ist (z. B. C:\).
SYSTEMROOT	System-Verzeichnis (z. B. <i>c:\winnt</i>). Entspricht der Variablen WINDIR.
WINDIR	System-Verzeichnis (z. B. <i>c:\winnt</i>). Entspricht der Variablen SYSTEMROOT.
TEMP	Verzeichnis zur Aufnahme temporärer Dateien (z. B. <i>c:\temp</i>). Es handelt sich um eine benutzerdefinierte Umgebungsvariable, deren Inhalt variierbar ist (User, Volatile).
TMP	Verzeichnis zur Aufnahme temporärer Dateien (z. B. <i>c:\temp</i>). Benutzerdefiniert und variierbar (User, Volatile).

Das folgende Beispiel zeigt den Zugriff auf eine Umgebungsvariable in einem VBScript-Programm:

```
' System-Umgebungsvariable NUMBER_OF_PROCESSORS einlesen
Set WshShell = WScript.CreateObject("Wscript.Shell")
Set WshSysEnv = WshShell.Environment("SYSTEM")
Wscript.Echo WshSysEnv("NUMBER_OF_PROCESSORS")
```

Hier wird eine Objektreferenz auf *WScript.Shell* erzeugt und in der Objektvariablen *WshShell* hinterlegt. Anschließend wird über dieses Objekt auf die *Environment*-Eigenschaft zugegriffen. Diese Eigenschaft liefert wiederum ein Objekt in die Objektvariable *WshSysEnv* zurück. Anschließend läßt sich direkt auf die betreffende Umgebungsvariable zugreifen.

Weitere Informationen finden Sie im Abschnitt mit der Beschreibung des  Hinweis *WshEnvironment*-Objekts.

WshShell.SpecialFolders

Die *SpecialFolders*-Eigenschaft stellt das *WshSpecialFolders*-Objekt zum Zugriff auf die Ordner der Windows-Shell wie *Desktop*, *Startmenü* und *Eigene Dateien*. Zur Verfügung. Hierbei gilt folgende Syntax:


Syntax:

WshShell.SpecialFolders = objWshSpecialFolders

Die folgende Anweisung zeigt, wie dies konkret in einem VBScript-Programm aussieht:

```
' Das Codefragment zeigt den Zugriff auf den Ordner Desktop
Set WshShell = Wscript.CreateObject("Wscript.Shell")
MsgBox "Ihr Desktop ist " & WshShell.SpecialFolders("Desktop")
```

Hinweis

Weitere Hinweise finden Sie unter dem  *WshSpecialFolders*-Objekt.

Methoden des WshShell-Objekts

Die folgende Tabelle beschreibt die mit dem *WshShell*-Objekt assoziierten Methoden.

Tabelle A.6:
*Methoden des
WshShell-Objekts*

Methode	Beschreibung
CreateShortcut	Erzeugt ein <i>WshShortcut</i> -Objekt und gibt dieses zurück.
ExpandEnvironmentStrings	Expandiert eine PROCESS-Umgebungsvariable und liefert den resultierenden String zurück.
Popup	Öffnet ein Meldungsfeld mit der angegebenen Nachricht.
RegDelete	Löscht den angegebenen Schlüssel oder den Wert aus der Registrierung.
RegRead	Liest den angegebenen Schlüssel oder Wert aus der Registrierung.
RegWrite	Setzt den angegebenen Wert oder Schlüssel in der Registrierung.
Run	Erzeugt einen neuen Prozeß, der das angegebene Kommando im vorgegebenen Windows-Stil ausführt.

Die folgenden Abschnitte geben detaillierte Hinweise auf die einzelnen Methoden des *WshShell*-Objekts.

WshShell.CreateShortcut

Die *CreateShortcut*-Methode erzeugt ein *WshShortcut*-Objekt und gibt dieses zurück. Endet der Text der Verknüpfung (der Titel der Verknüpfungsdatei) mit *.url*, wird ein *WshUrlShortcut*-Objekt angelegt. Für die Methode gilt folgende Syntax:

Syntax:

`WshShell.CreateShortcut(strPathname) = objShortcut`

Die folgende Sequenz zeigt, wie die Methode benutzt wird:

```
' Dieses Codefragment erzeugt eine Verknüpfung (shortcut)
' auf das aktuell ausgeführte Skript
Set WshShell = WScript.CreateObject("Wscript.Shell")
Set oShellLink = WshShell.CreateShortcut("Current Script.lnk")
oShellLink.TargetPath = WScript.ScriptFullName
oShellLink.Save
Set oUrlLink = WshShell.CreateShortcut("Microsoft Website.URL")
oUrlLink.TargetPath = "http://www.microsoft.com"
oUrlLink.Save
```

Weitere Hinweise finden Sie unter dem ↗ *WshShortcut*-Objekt und dem ↗ *WshUrlShortcut*-Objekt.

WshShell.ExpandEnvironmentStrings

Die *ExpandEnvironmentStrings*-Methode erweitert die in *strString* angegebene PROCESS-Umgebungsvariable und gibt das Ergebnis als String zurück. Variablen sind in das Zeichen "%" einzuschließen. Die Groß-/Kleinschreibung ist bei der Umgebungsvariable ohne Bedeutung. Die Methode benutzt folgende Syntax:

Syntax:

`WshShell.ExpandEnvironmentStrings(strString) = strExpandedString`

Das folgende Beispiel zeigt die Anwendung dieser Methode:

```
MsgBox "Prompt ist " & WshShell.ExpandEnvironmentStrings("%PROMPT%")
```

WshShell.Popup

Die *Popup*-Methode öffnet ein Meldungsfeld, welches die in *strText* enthaltene Meldung anzeigt. Der Titel des Meldungsfelds wird über den Parameter *strTitle* gesetzt. Fehlt der Parameter *strTitle*, wird als Titel »Windows Scripting Host« benutzt. Die Methode besitzt die folgende Syntax:

Syntax:

WshShell.Popup(strText, [natSecondsToWait], [strTitle], [natType]) = intBut

Wird der Parameter *natSecondsToWait* angegeben und ist der Wert größer 0, schließt sich das Meldungsfeld nach der in *natSecondsToWait* in Sekunden angegebenen Wartezeit.

Der Parameter *natType* entspricht in seiner Bedeutung dem gleichen Parameter der Windows-*MessageBox*-Funktion. Die folgende Tabelle enthält die Werte und deren Bedeutung für den Parameter *natType*. Sie können dabei die Werte kombinieren.

Tabelle A.7:
Schaltflächentypen

Wert	Beschreibung
0	Zeige OK-Schaltfläche
1	Zeige OK- und Abbrechen-Schaltflächen
2	Zeige Schaltflächen Abbrechen, Wiederholen und Ignorieren
3	Zeige Schaltflächen Ja, Nein und Abbrechen
4	Zeige Schaltflächen Ja und Nein
5	Zeige Schaltflächen Wiederholen und Abbrechen

Hinweis

Beachten Sie, dass die Beschriftung der Schaltflächen von der lokalisierten Windows-Version abhängt (z. B. *Abbrechen*, *Wiederholen*, *Ignorieren*, *Ja*, *Nein* oder *Abort*, *Retry*, *Ignore*, *Yes*, *No*).

Die folgende Tabelle enthält die Werte zur Auswahl des im Meldungsfeld angezeigten Symbols.

Tabelle A.8:
Meldungsfeld-
Symbole

Wert	Beschreibung
16	Stop-Zeichen
32	Fragezeichen
48	Ausrufezeichen
64	Informationssymbol

Die beiden obigen Tabellen enthalten nicht alle Werte für *natType*. Eine komplette Liste finden Sie in der Win32-Dokumentation. Weiterhin enthält der im Literaturverzeichnis unter /3/ aufgeführte Microsoft Press-Titel eine Beschreibung der betreffenden Werte. Dort werden auch die VBA-Konstanten für die betreffenden Werte vorgestellt.

Der Rückgabewert in *intBut* bezeichnet die Schaltfläche, die vom Benutzer beim Schließen des Dialogfeldes angeklickt wurde. Hat der Benutzer keine Schaltfläche angeklickt und das Dialogfeld wurde wegen Ablaufs der in *natSecondsToWait* angegebenen Wartezeit geschlossen, ist der Rückgabewert von *intBut* gleich -1. Die folgende Tabelle beschreibt die Rückgabewerte.

Wert	Beschreibung
1	OK-Schaltfläche angeklickt
2	Cancel/Abbrechen-Schaltfläche angeklickt
3	Abort/Beenden-Schaltfläche angeklickt
4	Retry/Wiederholen-Schaltfläche angeklickt
5	Ignore/Ignorieren-Schaltfläche angeklickt
6	Yes/Ja-Schaltfläche angeklickt
7	No/Nein-Schaltfläche angeklickt

Tabelle A.9:
Popup-
Rückgabewerte

Die folgende Anweisungsfolge zeigt den Einsatz dieser Methode:

```
Set WshShell = WScript.CreateObject("WScript.Shell")
WshShell.Popup "Hallo"
```

Weitere Informationen finden Sie unter der ➤ *WScript.Echo*-Methode.

Hinweis

WshShell.RegDelete

Die *RegDelete*-Methode löscht einen Wert oder einen Schlüssel aus der Registrierung. Der Wert bzw. Schlüssel ist im Parameter *strName* anzugeben. Hierbei gilt folgende Syntax:

Syntax:

WshShell.RegDelete strName

Als Parameter wird *strName* übergeben, um den Schlüssel oder den Wert festzulegen. Endet *strName* mit einem Backslash-Zeichen (\), entfernt die Methode den Schlüssel anstelle des Werts. Der *strName*-Parameter muß mit einem der nachfolgend aufgeführten Hauptschlüssel (root key) beginnen:

Tabelle A.10:
Kürzel für die
Hauptschlüssel

Kürzel	Langer Name
HKCU	HKEY_CURRENT_USER
HKLM	HKEY_LOCAL_MACHINE
HKCR	HKEY_CLASSES_ROOT
	HKEY_USERS
	HKEY_CURRENT_CONFIG

Das folgende Beispiel zeigt den Einsatz der betreffenden Methode:

```
Set WshShell = WScript.CreateObject("WScript.Shell")
WshShell.RegDelete "HKCU\ScriptEngine\Value" ' Lösche Wert "Value"
WshShell.RegDelete "HKCU\ScriptEngine\Key\" ' Lösche Schlüssel "Key"
```

Weitere Hinweise finden Sie unter der ➤ *WshShell.RegRead*-Methode und unter der ➤ *WshShell.RegWrite*-Methode.

WshShell.RegRead

Die *RegRead*-Methode liest den im Parameter *strName* aufgeführten Wert oder Schlüssel aus und liefert den Wert zurück.

Syntax:

WshShell.RegRead(strName) = strValue

Der Parameter *strName* gibt den Namen des Schlüssels oder Werts an, der zu lesen ist. Endet *strName* mit einem Backslash-Zeichen (\), gibt die Methode den Schlüssel anstelle des Werts zurück. *strName* muß mit einem der in **Tabelle A.10** angegebenen Werte beginnen.

Hinweis

Die *RegRead*-Methode unterstützt lediglich die folgenden Datentypen: REG_SZ, REG_EXPAND_SZ, REG_DWORD, REG_BINARY und REG_MULTI_SZ. Enthält die Registrierung andere Datentypen, gibt *RegRead* den Wert DISP_E_TYPEMISMATCH zurück.

Das folgende Beispiel zeigt den Einsatz dieser Methode:

```
Set WshShell = WScript.CreateObject("WScript.Shell")
WshShell.RegRead("HKCU\ScriptEngine\Val") ' Lese aus Wert "Val"
WshShell.RegRead("HKCU\ScriptEngine\Key\") ' Lese aus Schlüssel "Key"
```

Hinweis

Weitere Informationen finden Sie unter ➤ *WshShell.RegDelete*-Methode und ➤ *WshShell.RegWrite*-Methode.

WshShell.RegWrite

Die *RegWrite*-Methode setzt den in *strName* angegebenen Wert oder Schlüssel der Registrierung.

Syntax:

WshShell.RegWrite strName, anyValue, [strType]

Der Parameter *strName* gibt den Namen des Werts oder Schlüssels an. Endet *strName* mit einem Backslash-Zeichen (\), schreibt die Methode in den Schlüssel und nicht in einen Wert. *strName* muß mit einem der in **Tabelle A.10** angegebenen Werte beginnen.

RegWrite unterstützt in *strType* die Datentypen REG_SZ, REG_EXPAND_SZ, REG_DWORD und REG_BINARY. Wird in *strType* ein anderer Datentyp übergeben, gibt *RegWrite* den Wert E_INVALIDARG zurück. Hinweis

RegWrite wandelt den Wert im Typ *anyValue* automatisch in eine Zeichenkette (String) um, falls *strType* vom Typ REG_SZ oder REG_EXPAND_SZ ist. Falls *strType* vom Typ REG_DWORD ist, wird der Wert in *anyValue* in eine Integer-Größe konvertiert. Ist *strType* vom Typ REG_BINARY, muß *anyValue* einen Integer-Wert enthalten. Das folgende Beispiel zeigt die Verwendung dieser Methode.

```
Set WshShell = WScript.CreateObject("WScript.Shell")
WshShell.RegWrite "HKCU\ScriptEngine\Value", "Ein String Wert"
WshShell.RegWrite "HKCU\ScriptEngine\Key\", 1 "REG_DWORD"
```

Siehe auch unter der ➤ *WshShell.RegDelete*-Methode und unter der ➤ Hinweis *WshShell.RegWrite*-Methode.

WshShell.Run

Die *Run*-Methode erzeugt einen neuen Prozeß, der den in *strCommand* enthaltenen Befehl ausführt und den Fensterstil aus *intWindowStyle* benutzt.

Syntax:

WshShell.Run (strCommand, [intWindowStyle], [bWaitOnReturn])

Als Parameter muß der Methode das auszuführende Kommando in *strCommand* übergeben werden. Umgebungsvariablen im *strCommand*-Parameter werden automatisch expandiert.

Mit dem optionalen Parameter *bWaitOnReturn* läßt sich festlegen, ob das Skript auf das Ende des gestarteten Prozesses warten soll. Ist *bWaitOnReturn* nicht spezifiziert oder auf *FALSE* gesetzt, kehrt die Methode sofort zur Ausführung des Skripts zurück und wartet nicht auf die Beendigung des

Prozesses. Ist *bWaitOnReturn* auf *TRUE* gesetzt, liefert die *Run*-Methode den von der Anwendung zurückgegebenen Fehlercode (error code) zurück.

Fehlt der Parameter *bWaitOnReturn* oder ist der Wert auf *FALSE* gesetzt, liefert die *Run*-Methode den Fehlercode (error code) 0 (zero) zurück. Das folgende Beispiel zeigt die Anwendung dieser Methode:

```
' Dieses Codefragment startet Notepad mit dem aktuell ausgeführten Skript
Set WshShell = WScript.CreateObject("WScript.Shell")
WshShell.Run ("notepad " & WScript.ScriptFullName)
WshShell.Run ("%windir%\notepad" & WScript.ScriptFullName)

' Dieses Codefragment gibt den Fehlercode der ausgeführten Anwendung zurück
Return = WshShell.Run("notepad " & WScript.ScriptFullName, 1, TRUE)
```

Hinweis

Beachten Sie aber, dass die Methode beim Aufruf des Windows-Explorers nicht wartet.

WshNetwork-Objekt

Das *WshNetwork*-Objekt besitzt die ProgID *WScript.Network* und wird über die Datei *WSHom.ocx* bereitgestellt. Die folgende Tabelle beschreibt die mit dem *WshNetwork*-Objekt assoziierten Eigenschaften.

Tabelle A.11:
Eigenschaften des
WScript.Network-
Objekts

Eigenschaft	Beschreibung
ComputerName	Eine Zeichenkette, die den Computernamen angibt.
UserDomain	Zeichenkette mit dem Namen der User Domain.
UserName	Zeichenkette mit dem Benutzernamen.

Die folgende Tabelle beschreibt die mit dem *WshNetwork*-Objekt assoziierten Methoden.

Tabelle A.12:
Methoden des
WshNetwork-
Objekts

Methode	Beschreibung
AddPrinterConnection	Verbindet einen Netzwerkdrucker mit einem lokalen Ressourcennamen.
EnumNetworkDrives	Gibt die Verbindungen für die aktuellen Netzlaufwerke (Network Drive Mappings) zurück.
EnumPrinterConnections	Gibt die Verbindungen für die aktuellen Netzdrucker (Printer Connection Mappings) zurück.

MapNetworkDrive	Verbindet einen freigegebenes Netzlaufwerk mit einem lokalen Laufwerksbuchstaben (local resource name).
RemoveNetworkDrive	Löst die Zuordnung eines Laufwerksbuchstabens zu einem Netzlaufwerk.
RemovePrinterConnection	Entfernt die Zuordnung eines Netzwerkdrucker.
SetDefaultPrinter	Setzt den Standarddrucker.

Die Eigenschaften des *WshNetwork*-Objekts

Nachfolgend finden Sie eine kurze Beschreibung der von diesem Objekt bereitgestellten Eigenschaften.

WshNetwork.ComputerName

Die *ComputerName*-Eigenschaft liefert eine Zeichenkette mit dem Namen des Computers.

Syntax:

WshNetwork.ComputerName = strComputerName

Die folgenden Anweisungen zeigen die Anwendung dieser Eigenschaft.

```
Set WshNetwork = WScript.CreateObject("WScript.Network")
WScript.Echo WshNetwork.ComputerName
```

WshNetwork.UserDomain

Die *UserDomain*-Eigenschaft liefert eine Zeichenkette mit dem Domain-Namen des Benutzers.

Syntax:

WshNetwork.UserDomain = strDomain

Die folgenden Anweisungen zeigen die Anwendung dieser Eigenschaft.

```
Set WshNetwork = WScript.CreateObject("WScript.Network")
WScript.Echo WshNetwork.UserDomain
```

WshNetwork.UserName

Die *UserName*-Eigenschaft liefert eine Zeichenkette mit dem Benutzernamen.

Syntax:

```
WshNetwork.UserName = strName
```

Die folgenden Anweisungen zeigen die Anwendung dieser Eigenschaft.

```
Set WshNetwork = WScript.CreateObject("WScript.Network")
WScript.Echo WshNetwork.ComputerName
```

Die Methoden des *WshNetwork*-Objekts

Nachfolgend finden Sie eine kurze Beschreibung der von diesem Objekt bereitgestellten Methoden.

WshNetwork.AddPrinterConnection

Die *AddPrinterConnection*-Methode verbindet einen in *strRemoteName* angegebenen Netzwerkdrucker mit der in *strLocalName* angegebenen lokalen Ressource.

Syntax:

```
WshNetwork.AddPrinterConnection strLocalName, strRemoteName,
[bUpdateProfile], [strUser], [strPassword]
```

Die Methode erwartet verschiedene Parameter, die die Druckerverbindung spezifizieren. Der Parameter *strLocalName* definiert den Namen der lokalen Ressource, die zu verbinden ist. In *strRemoteName* übergeben Sie den Netzwerkdrucker, der zu verbinden ist. Der optionale Parameter *bUpdateProfile* legt fest, ob diese Verbindung dauerhaft im Benutzerprofil hinterlegt wird. Ist *bUpdateProfile* vorhanden und der Wert *TRUE*, wird die Verbindung zwischen Netzwerkeinheit und lokaler Ressource im Benutzerprofil abgespeichert. Die optionalen Parameter *strUser* und *strPassword* erlauben Ihnen, eine Verbindung für einen Netzwerkdrucker für einen anderen Benutzer (nicht den aktuellen Benutzer) zu definieren. Den Namen des Benutzers und dessen Kennwort können Sie in den Parametern *strUser* und *strPassword* angeben. Der folgende Code zeigt die Anwendung der Methode.

```
Set WshNetwork = WScript.CreateObject("WScript.Network")
WshNetwork.AddPrinterConnection "LPT1", "\\Server\Print1"
```

WshNetwork.EnumNetworkDrives

Die *EnumNetworkDrives*-Methode gibt die aktuelle Netzlaufwerkszuordnung als *WshCollection*-Objekt zurück. Die Einträge dieser Auflistung (collection) sind lokale Namen (local names) und Netzwerknamen (remote names).



Syntax:

WshNetwork.EnumNetworkDrives = objWshCollection

Die folgende VBScript-Sequenz zeigt die Anwendung dieser Methode, um eine Auflistung zu lesen und die Elemente auszugeben.

```
Set WshNetwork = WScript.CreateObject("WScript.Network")
Set oDrives = WshNetwork.EnumNetworkDrives
WScript.Echo oDrives.Item(0) = "Z:"
WScript.Echo oDrives.Item(1) = "\\Server\Share"
```

Das erste Element der Auflistung liefert den Laufwerksbuchstaben, während der UNC-Pfad zur Netzwerkressource im zweiten Element steht.

Zusätzliche Informationen finden Sie auch unter der *WshNetwork.MapNetworkDrive*-Methode, der *WshNetwork.RemoveNetworkDrive*-Methode und beim *WshCollection*-Objekt.  Hinweis 

WshNetwork.EnumPrinterConnections




Die *EnumPrinterConnections*-Methode gibt die aktuellen Druckerzuordnungen als *WshCollection*-Objekt zurück.

Syntax:

WshNetwork.EnumPrinterConnections = objWshCollection

Die Einträge dieser Auflistung (collection) enthalten die lokalen Namen und die Netzwerknamen. Das folgende Beispiel zeigt den Einsatz dieser Methode.

```
Set WshNetwork = WScript.CreateObject("WScript.Network")
Set oPrinters = WshNetwork.EnumPrinterConnections
WScript.Echo oPrinters.Item(0) = "LPT1:"
WScript.Echo oPrinters.Item(1) = "\\Server\Printer1"
```

Weitere Informationen finden Sie unter der *WshNetwork.AddPrinterConnection*-Methode, unter der *WshNetwork.RemovePrinterConnection*-Methode und unter dem *WshCollection*-Objekt.  Hinweis  

WshNetwork.MapNetworkDrive

Die *MapNetworkDrive*-Methode verbindet ein in *strRemoteName* angegebenes Netzlaufwerk mit dem in *strLocalName* aufgeführten Laufwerksbuchstaben.

Syntax:

```
WshNetwork.MapNetworkDrive strLocalName, strRemoteName,  
[bUpdateProfile], [strUser], [strPassword]
```

Die Methode erwartet verschiedene Parameter. In *strLocalName* wird der Name des Laufwerks (lokale Ressource) angegeben, die mit der Netzwerkeinheit zu verbinden ist. In *strRemoteName* übergeben Sie den Namen der zu verbindenden Netzwerkeinheit. Der optionale Parameter *bUpdateProfile* legt fest, ob die Verbindung temporär oder permanent einzurichten ist. Wird *bUpdateProfile* angegeben und ist der Wert *TRUE*, speichert die Methode die Verbindung im Benutzerprofil. Die beiden optionalen Parameter *strUser* und *strPassword* erlauben Ihnen, die Netzlaufwerkzuordnung für einen anderen als den aktuellen Benutzer vorzunehmen. Das folgende Beispiel zeigt die Anwendung der Methode:

```
Set WshNetwork = WScript.CreateObject("WScript.Network")  
WshNetwork.MapNetworkDrive "Z:", "\\Server\Share"
```

WshNetwork.RemoveNetworkDrive

Die *RemoveNetworkDrive*-Methode entfernt die durch *strName* definierte aktuelle Verbindung zur Ressource.

Syntax:

```
WshNetwork.RemoveNetworkDrive strName, [bForce], [bUpdateProfile]
```

Der *strName*-Parameter kann, abhängig von der Art der Netzlaufwerkzuordnung, entweder einen lokalen Namen oder einen Netzwerknamen aufnehmen. Wird die Verbindung zwischen einem lokalen Namen (Laufwerksbuchstaben) und einem Netzwerknamen vorgenommen, muß *strName* den lokalen Namen enthalten. Enthält der Netzwerkpfad keinen lokalen Namen (Laufwerksbuchstaben), wird in *strName* der Netzwerkname (remote name) abgelegt.

Ist der optionale Parameter *bForce* vorhanden und dessen Wert auf *TRUE* gesetzt, entfernt die Methode die Netzwerkverbindung, unabhängig davon, ob die Ressource benutzt wird oder nicht. Ist der Parameter *bUpdateProfile* angegeben und dessen Wert auf *TRUE* gesetzt, wird die Verbindung im Benutzerprofil gespeichert. Das folgende Beispiel zeigt die Anweisung in VBScript:

```
Set WshNetwork = WScript.CreateObject("WScript.Network")
```

```
' Lokalen Namen auf freigegebene Netzwerkressource abbilden
WshNetwork.MapNetworkDrive "Z:", "\\Server\Share"
WshNetwork.RemoveNetworkDrive "Z:"
' Keine Verbindung mit einem lokalen Namen wie:
' NET USE \\Server\Share WshNetwork.RemoveNetworkDrive "\\Server\Share"
WshNetwork.MapNetworkDrive "\\Server\Share"
WshNetwork.RemoveNetworkDrive "\\Server\Share"
```

WshNetwork.RemovePrinterConnection

Die *RemovePrinterConnection*-Methode hebt die aktuelle in *strName* angegebene Netzwerkverbindung.

Syntax:

```
WshNetwork.RemovePrinterConnection strName, [bForce], [bUpdateProfile]
```

Der *strName*-Parameter kann entweder einen lokalen oder einen Netzwerknamen, abhängig von der Art, wie der Drucker verbunden wurde, enthalten. Weist der Drucker eine Verbindung zwischen einem lokalen Namen (z. B. LPT1) und einer Netzwerkressource auf, muß in *strName* der lokale Name hinterlegt werden. Enthält der Netzwerkpfad keinen lokalen Namen in der Verknüpfung, muß in *strName* der Name der Netzwerkressource angegeben werden.

Wird der optionale Parameter *bForce* angegeben, und ist der Wert auf *TRUE* gesetzt, hebt die Methode die Verbindung auf, unabhängig davon, ob die Ressource benutzt wird oder nicht. Wird der Parameter *bUpdateProfile* angegeben, und ist der Wert *TRUE*, wird die Verbindung im Benutzerprofil abgelegt. Das folgende Beispiel zeigt die Anwendung der Methode.

```
Set WshNetwork = WScript.CreateObject("WScript.Network")
' Lokalen Namen auf freigegebene Netzwerkressource abbilden
WshNetwork.RemovePrinterConnection "LPT1:"
' Kein lokaler Name, z. B. NET USE "\\Server\Printer1"
WshNetwork.RemovePrinterConnection "\\Server\Printer1"
```

WshNetwork.SetDefaultPrinter

Die *SetDefaultPrinter*-Methode setzt den Standarddrucker auf den im Parameter *strPrinterName* angegebenen Netzwerkdrucker.

Syntax:

```
WshNetwork.SetDefaultPrinter strPrinterName
```

Die Methode benutzt nur den Parameter *strPrinterName*. Der Parameter enthält den Namen des Netzwerkdruckers, der als Standarddrucker zu

verwenden ist (z. B. "\\Server\Printer1"). Beachten Sie aber, dass *strPrinterName* keinen lokalen Namen wie "LPT1:" aufnehmen kann.

WshCollection-Objekt

Das *WshCollection*-Objekt steht nicht direkt zur Verfügung. Zugriffe auf diese Objekt erfolgen über *WshNetwork.EnumNetworkDrives* oder *WshNetwork.EnumPrinterConnections*. Das Objekt wird über die Datei *WSHom.ocx* bereitgestellt. Die folgende Tabelle beschreibt die mit dem *WshCollection*-Objekt assoziierten Eigenschaften.

Tabelle A.13:
*Eigenschaften des
WshCollection-
Objekts*

Eigenschaft	Beschreibung
Item	Liefert den n-ten Eintrag der Aufzählung (Enumeration) als Zeichenkette (String).
Count	Zahl der Einträge in der Aufzählung.
length	Zahl der Aufzählungsobjekte, wird für JScript benötigt.

Die Eigenschaften des *WshCollection*-Objekts

Der folgende Abschnitt enthält die Beschreibung der vom Objekt bereitgestellten Eigenschaften.

WshCollection.Item

Die *Item*-Eigenschaft liefert das in *natIndex* angegebene Element als Zeichenkette zurück. Es handelt sich um die Standardeigenschaft.

Syntax:

WshCollection(natIndex) = strEnumeratedItem

oder

WshCollection.Item(natIndex) = strEnumeratedItem

Das folgende Beispiel zeigt die Handhabung dieser Auflistung.


```
Set WshNetwork = WScript.CreateObject("WScript.Network")
Set oDrives = WshNetwork.EnumNetworkDrives
WScript.Echo oDrives.Item(0) = "Z:"
WScript.Echo oDrives.Item(1) = "\\Server\Share"
```


WshCollection.Count

Die *Count*-Eigenschaft liefert die Zahl der Einträge in der Auflistung.

Syntax:

WshCollection.Count = natNumberOfItems



Zusätzliche Informationen finden Sie bei der Beschreibung der  Hinweis *WshCollection.length*-Eigenschaft.

WshCollection.length

Die *length*-Eigenschaft liefert die Zahl der Einträge in der Auflistung. Diese Eigenschaft besitzt die gleiche Funktionalität wie die *Count*-Eigenschaft und wird aus Kompatibilitätsgründen in JScript benötigt.

Syntax:

WshCollection.length = natNumberOfItems

Siehe auch die Beschreibung der  *WshCollection.Count*-Eigenschaft.  Hinweis

WshEnvironment-Objekt

Das *WshEnvironment*-Objekt ist nicht direkt verfügbar. Der Zugriff erfolgt über die *WshShell.Environment*-Eigenschaft. Das Objekt wird über die Datei *WSHom.ocx* bereitgestellt. Die folgende Tabelle beschreibt die mit dem *WshEnvironment*-Objekt assoziierten Eigenschaften.

Eigenschaft	Beschreibung
Item	Liest oder setzt den Wert der angegebenen Umgebungsvariable.
Count	Die Zahl der Einträge.
length	Die Zahl der Einträge (enumerated items, wird für JScript benötigt).

Tabelle A.14:
*Eigenschaften des
WshEnvironment-
Objekts*

Die folgende Tabelle beschreibt die mit dem *WshEnvironment*-Objekt assoziierten Methode.

Methode	Beschreibung
Remove	Löscht die angegebene Umgebungsvariable.

Tabelle A.15:
*Methode des
WshEnvironment-
Objekts*

Die Eigenschaften des *WshEnvironment*-Objekts

Der folgende Abschnitt enthält die Beschreibung der vom *WshEnvironment*-Objekt bereitgestellten Eigenschaften.

WshEnvironment.Item

Die *Item*-Eigenschaft setzt oder liest den Wert der in *strName* angegebenen Umgebungsvariable. Es handelt sich um die Standardeigenschaft.

Syntax:

`WshEnvironment.Item("strName") = strValue`

Oder

`WshEnvironment("strName") = strValue`

Das folgende Beispiel zeigt den Einsatz der Eigenschaft:

```
' Hole den Wert der NUMBER_OF_PROCESSORS-Umgebungsvariablen
Set WshShell = WScript.CreateObject("WScript.Shell")
Set WshSysEnv = WshShell.Environment("SYSTEM")
WScript.Echo WshSysEnv("NUMBER_OF_PROCESSORS")

' Setzt den Wert der temporären (VOLATILE) Umgebungsvariable
' EXAMPLE auf A_VALUE
Set WshShell = WScript.CreateObject("WScript.Shell")
Set WshEnv = WshShell.Environment("VOLATILE")
WshEnv("EXAMPLE") = "A_VALUE"

' Liste alle SYSTEM-Umgebungsvariable
Set WshShell = WScript.CreateObject("WScript.Shell")
For Each strVarName In WshShell.Environment("SYSTEM")
MsgBox strVarName
Next
```

Hinweis

Siehe auch die Beschreibung der ➤ *WshShell.Environment*-Eigenschaft.

WshEnvironment.Count

Die Eigenschaft *Count* liefert die Zahl der Aufzählungselemente.

Syntax:

WshEnvironment.Count = natNumberOfItems

Siehe auch die Beschreibung der ➤ *WshEnvironment.length*-Eigenschaft.

Hinweis

WshEnvironment.length

Die *length*-Eigenschaft liefert die Zahl der Elemente in der Auflistung. Die Eigenschaft besitzt die gleiche Funktionalität wie die *Count*-Eigenschaft und wird für die JScript-Kompatibilität benötigt.

Syntax:

WshEnvironment.length = natNumberOfItems

Siehe auch die Beschreibung der ➤ *WshEnvironment.Count*-Eigenschaft.

Hinweis

Die Methoden des *WshEnvironment*-Objekts

Der folgende Abschnitt enthält die Beschreibung der vom Objekt bereitgestellten Methode.

WshEnvironment.Remove

Die *Remove*-Methode löscht die in *strName* angegebene Umgebungsvariable.

Syntax:

WshEnvironment.Remove(strName)

Das folgende Beispiel zeigt den Einsatz dieser Methode:

```
' Lösche die temporäre EXAMPLE (volatile) Umgebungsvariable
Set WshShell = WScript.CreateObject("WScript.Shell")
WshShell.Environment("VOLATILE").Remove("EXAMPLE")

' Lösche mehrere Variablen
Set WshUsrEnv = WScript.Environment("User")
WshUsrEnv.Remove("EXAMPLE_1")
WshUsrEnv.Remove("EXAMPLE_2")
WshUsrEnv.Remove("EXAMPLE_3")
WshUsrEnv.Remove("EXAMPLE_4")
```

Siehe auch die Beschreibung der ➤ *WshShell.Environment*-Eigenschaft.

Hinweis

WshShortcut-Objekt

Dieses Objekt ist nicht direkt verfügbar, sondern wird durch die Datei *WshObj.dll* bereitgestellt. Zum Zugriff auf das *WshShortcut*-Objekt ist die *WshShell.CreateShortcut*-Methode zu verwenden. Die folgende Tabelle beschreibt die Eigenschaften des *WshShortcut*-Objekts.

Tabelle A.16:
Eigenschaften des
WshShortcut-
Objekts

Eigenschaft	Beschreibung
Arguments	Parameter des Verknüpfungsobjekts (shortcut object).
Description	Beschreibung des Verknüpfungsobjekts.
Hotkey	Abkürzungstaste zur Aktivierung der Verknüpfung.
IconLocation	Position des Symbols des Verknüpfungsobjekts.
TargetPath	Zielpfad der Verknüpfung.
WindowStyle	Fensterstil der Verknüpfung.
WorkingDirectory	Arbeitsverzeichnis der Verknüpfung.

Die folgende Tabelle beschreibt die mit dem *WshShortcut*-Objekt assoziierte Methode.

Tabelle A.17:
Methode des
WshShortcut-
Objekts

Methode	Beschreibung
Save	Speichert die Verknüpfung im spezifizierten Dateisystem.

Die Eigenschaften des WshShortcut-Objekts

Der folgende Abschnitt enthält die Beschreibung der vom Objekt bereitgestellten Eigenschaften.

WshShortcut.Arguments

Die *Arguments*-Eigenschaft liefert die Parameter zu einem Verknüpfungsobjekt.

Syntax:

WshShortcut.Arguments = strArguments

WshShortcut.Description

Die *Description*-Eigenschaft liefert die Beschreibung des Verknüpfungsobjekts.

Syntax:

WshShortcut.Description = strDescription

WshShortcut.FullName

Die *FullName*-Eigenschaft liefert den vollständigen Pfad des Verknüpfungsobjekts.

Syntax:

WshShortcut.FullName = strFullName

WshShortcut.HotKey

Die Eigenschaft *HotKey* liefert die Aktivierungstaste zum Aufruf der Verknüpfung. Über diese Aktivierungstaste läßt sich das zur Verknüpfung zugehörige Programm starten oder in den Vordergrund schalten.

Syntax:

WshShortcut.HotKey = strHotKey

Die BNF-Syntax in *strHotKey* ist folgendermaßen definiert:

Aktivierungstaste ::= Bezeichner* Taste
Bezeichner ::= "ALT+" | "STRG+" | "SHIFT+" | "EXT+"
Taste ::= "A" .. "Z" |
 "0" .. "9" |
 "Back" | "Tab" | "Clear" | "Return" |
 "Escape" | "Space" | "Prior" | ...

Eine vollständig Liste der Tastennamen finden Sie in der Datei *Winuser.h*. Der Wert in *strHotKey* ist nicht case-sensitive, d. h. es wird nicht zwischen Groß-/Kleinschreibung unterschieden.

Die Eigenschaft *HotKey* kann nur Verknüpfungen aktivieren, die auf dem Windows-Desktop oder im Startmenü eingetragen sind. Der Windows-Explorer akzeptiert die Tasten Esc, Eingabe, Tab, Leer, Druck oder Rück nicht, auch wenn *WshShortcut.HotKey* diese in Übereinstimmung mit der Win32-API unterstützt. Daher wird empfohlen, diese Tasten nicht in Verknüpfungen zu verwenden.

Das folgende Beispiel zeigt die Anwendung der Eigenschaften und Methoden:

```
Set WshShell = WScript.CreateObject("WScript.Shell")
strDesktop = WshShell.SpecialFolders("Desktop")
Set oMyShortcut = WshShell.CreateShortcut(strDesktop & "\a_key.lnk")
oMyShortcut.TargetPath = "%windir%\notepad.exe"
oMyShortcut.HotKey = "ALT+CTRL+F"
oMyShortcut.Save
Wscript.Echo oMyShortcut.HotKey = "Alt+Ctrl+F"
```

Hinweis

Siehe auch die Beschreibung im Abschnitt zum [↗ WshSpecialFolders](#)-Objekt.

WshShortcut.IconLocation

Die *IconLocation*-Eigenschaft liefert die Position des Verknüpfungsobjekts. Das Format der Symbolposition ist als »Path,index« anzugeben.

Syntax:

WshShortcut.IconLocation = strIconLocation

WshShortcut.TargetPath

Die *TargetPath*-Eigenschaft liefert den Zielpfad zum Verknüpfungsobjekt.

Syntax:

WshShortcut.TargetPath = strTargetPath

WshShortcut.WindowStyle

Die *WindowStyle*-Eigenschaft liefert den Fensterstil des Verknüpfungsobjekts.

Syntax:

WshShortcut.WindowStyle = natWindowStyle

WshShortcut.WorkingDirectory

Die *WorkingDirectory*-Eigenschaft liefert das Arbeitsverzeichnis des Verknüpfungsobjekts.

Syntax:

WshShortcut.WorkingDirectory = strWorkingDirectory

Die Methoden des *WshShortcut*-Objekts

Der folgende Abschnitt enthält die Beschreibung der vom Objekt bereitgestellten Methoden.

WshShortcut.Save

Die *Save*-Methode speichert das Verknüpfungsobjekt an die durch die *FullName*-Eigenschaft definierte Stelle.

Syntax:

WshShortcut.Save

WshSpecialFolders-Objekt

Dieses über die Datei *WSHom.ocx* bereitgestellte Objekt ist nicht direkt verfügbar. Zum Zugriff auf das *WshSpecialFolders*-Objekt ist die *WshShell.SpecialFolders*-Eigenschaft zu verwenden. Die folgende Tabelle beschreibt die Eigenschaften des *WshSpecialFolders*-Objekts.

Eigenschaft	Beschreibung
Item	Vollständiger Pfad des angegebenen speziellen Ordners (Standard)
Count	Zahl der Einträge in der Auflistung.
length	Zahl der Einträge in der Auflistung (für JScript).

Tabelle A.18:
*Eigenschaften des
WshSpecialFolders
-Objekts*

Die Eigenschaften des *WshSpecialFolders*-Objekts

Der folgende Abschnitt enthält die Beschreibung der vom *WshSpecialFolders*-Objekt bereitgestellten Eigenschaften.

WshSpecialFolders.Item

Die *Item*-Eigenschaft liefert den vollständigen Pfad für den in *strFolderName* angegebenen speziellen Ordner zurück. Dies ist die Standardeigenschaft.

Syntax:

```
WshShell.SpecialFolders.Item("strFolderName") = strFolderPath
```

oder

```
WshShell.SpecialFolders("strFolderName") = strFolderPath
```

Die Anweisung:

```
WshShell.SpecialFolders("strFolderName")
```

liefert *NULL* zurück, falls der angeforderte Ordner (*strFolderName*) nicht vorhanden ist. Beispielsweise verfügt Windows 95 nicht über den Ordner *AllUsersDesktop*, und die Eigenschaft liefert bei *strFolderName* = *AllUsersDesktop* *NULL* zurück.

Die Namen der folgenden Spezialordner sind in Windows 95 (teilweise), in Windows 98 und Windows NT 4.0/2000 vorhanden:

- AllUsersDesktop
- AllUsersStartMenu
- AllUsersPrograms
- AllUsersStartup
- Desktop
- Favorites
- Fonts
- MyDocuments
- NetHood
- PrintHood
- Programs
- Recent
- SendTo
- StartMenu
- Startup
- Templates

Das folgende Beispiel zeigt den Zugriff auf das *SpecialFolders*-Objekt:

```
' Liefert den vollständigen Pfad des Windows-Desktop-Ordners
Set WshShell = WScript.CreateObject("WScript.Shell")
StrMyDesktop = WshShell.SpecialFolders("Desktop")

' Liste alle Spezialordner
For Each strFolder In WshShell.SpecialFolders
```


Siehe auch die Beschreibung der ➤ *WshShell.SpecialFolders*-Eigenschaft.

Hinweis

WshSpecialFolders.Count

Die *Count*-Eigenschaft liefert die Zahl der Einträge der Auflistung.

Syntax:

WshSpecialFolders.Count = natNumberOfItems

Siehe auch die Beschreibung der ➤ *WshSpecialFolders.length*-Eigenschaft.

Hinweis

WshSpecialFolders.length

Die *length*-Eigenschaft liefert ebenfalls die Zahl der Einträge in der Auflistung. Die Eigenschaft besitzt die gleiche Funktionalität wie *Count* und wird aus Kompatibilitätsgründen in Microsoft JScript unterstützt.

Syntax:

WshSpecialFolders.length = natNumberOfItems

Siehe auch die Beschreibung der ➤ *WshSpecialFolders.Count*-Eigenschaft.

Hinweis

WshUrlShortcut-Objekt

Dieses durch die Datei *WSHom.ocx* bereitgestellte Objekt ist nicht direkt verfügbar. Der Zugriff auf das *WshUrlShortcut*-Objekt erfolgt über die *WshShell.CreateShortcut*-Methode. Die folgende Tabelle beschreibt die Eigenschaften des *WshUrlShortcut*-Objekts.

Eigenschaft	Beschreibung
FullName	Vollständiger Pfad einer Verknüpfung zu einer URL.
TargetPath	Zielpfad zu einem URL-Verknüpfungsobjekt.

Tabelle A.19:
Eigenschaften des
WshUrlShortcut-
Objekts

Die folgende Tabelle beschreibt die Methode des *WshUrlShortcut*-Objekts.

Methode	Beschreibung
Save	Speichert die Verknüpfung im angegebenen Dateisystem.

Tabelle A.20:
Methode des
WshUrlShortcut-
Objekts

Die Eigenschaften des *WshUrlShortcut*-Objekts

Der folgende Abschnitt enthält die Beschreibung der vom *WshUrlShortcut*-Objekt bereitgestellten Eigenschaften.

WshUrlShortcut.FullName

Die *FullName*-Eigenschaft liefert den vollständigen Pfad einer Verknüpfung.

Syntax:

`WshUrlShortcut.FullName = strFullName`

WshUrlShortcut.TargetPath

Die Eigenschaft *TargetPath* liefert den Zielpfad des Verknüpfungsobjekts.

Syntax:

`WshUrlShortcut.TargetPath = strTargetPath`

Die Methode des *WshUrlShortcut*-Objekts

Der folgende Abschnitt enthält die Beschreibung der vom Objekt bereitgestellten Methode.

WshUrlShortcut.Save

Die *Save*-Methode speichert die Verknüpfung an dem in der *FullName*-Eigenschaft angegebenen Speicherort.

Syntax:

`WshUrlShortcut.Save`

Hinweis

Beispiele zur Anwendung dieser Methode finden Sie in verschiedenen Kapiteln dieses Buches.