

[← Zurück zur Übersichtsseite](#)

Microsoft® Windows® 2000 – Scripting-Handbuch (Teil 1) Scripting-Konzepte und -Technologien zur Systemadministration

Kapitel 4 – Die Script-Laufzeitbibliothek

(Engl. Originaltitel: [Script Runtime Primer](#))

Die Verwaltung des Dateisystems ist ein wichtiger Teil der Systemadministration – und leider bieten in diesem Bereich weder der Windows Script Host (WSH) noch Microsoft® Visual Basic® Scripting Edition (VBScript) sehr viele Möglichkeiten. Glücklicherweise können Sie zur Verwaltung von Laufwerken, Ordnern und Dateien die Script-Laufzeitbibliothek (Script Runtime Library) verwenden. Sie stellt Methoden zum Lesen und Schreiben in und aus Textdateien, zum Erstellen von "Verzeichnissen" und zum Codieren von Scripten zur Verfügung.

Inhaltsverzeichnis

Script Laufzeitbibliothek-Übersicht.....	4
Das Objekt FileSystemObject.....	4
Verwalten von Laufwerken.....	5
Eine Collection mit Laufwerken abrufen.....	5
Binden an ein bestimmtes Laufwerk.....	6
Auflisten der Laufwerkeigenschaften.....	6
Prüfen, ob ein Laufwerk bereit ist.....	8
Verwalten von Ordnern.....	9
Eine Referenz auf einen Ordner erstellen.....	9
Prüfen, ob ein Ordner vorhanden ist.....	10
Einen Ordner erstellen.....	10
Löschen eines Ordners.....	11
Ordner und deren Inhalte kopieren.....	12
Verschieben von Ordnern und deren Inhalten.....	13
Ordner umbenennen.....	13
Verwenden von Ordneigenschaften.....	14
Auflisten von Ordneigenschaften.....	14
Verwalten von Ordnerattributen.....	16
Ändern von Ordnerattributen.....	18
Auflisten der Dateien in einem Ordner.....	19
Auflisten von Unterordnern.....	20
Verwalten von Dateien.....	22
Eine Referenz auf eine Datei erstellen.....	22
Prüfen, ob eine Datei vorhanden ist.....	23
Löschen einer Datei.....	24
Kopieren einer Datei.....	25
Verschieben einer Datei.....	25
Eine Datei umbenennen.....	26

Abfragen von Dateieigenschaften	26
Auflisten der Dateiattribute	28
Konfigurieren von Dateiattributen	29
Den Pfad einer Datei verarbeiten	30
Abfragen der Dateiversion	31
Textdateien lesen und schreiben	32
Textdateien erstellen	32
Lesen von Textdateien	37
In Textdateien schreiben	42
Das Dictionary-Objekt	45
Ein Dictionary-Objekt erstellen	45
Einträge zu einem Dictionary-Objekt hinzufügen	46
Bearbeiten von Schlüsseln und Werten in einem Dictionary-Objekt.....	47
Die Zahl der Einträge in einem Dictionary-Objekt abfragen	48
Die Elemente eines Dictionary-Objekts aufzählen	48
Die Existenz eines Schlüssels prüfen.....	49
Ein Element in einem Dictionary-Objekt ändern.....	50
Elemente aus einem Dictionary-Objekt entfernen.....	51

Script Laufzeitbibliothek-Übersicht

Es gibt zwei primäre Microsoft-Scriptsprachen: *Microsoft® Visual Basic® Scripting Edition (VBScript)* und *Microsoft® JScript®*. Diese wurden ursprünglich als clientseitige Scriptsprachen für den Microsoft® Internet Explorer entworfen. Daher gibt es einige Einschränkungen bei den beiden Sprachen. Weder VBScript noch JScript verfügen zum Beispiel über Methoden, um Dateioperationen wie das Kopieren, Verschieben oder Löschen von Dateien durchzuführen. Diese Einschränkungen wurden ursprünglich zum Schutz der Kunden durchgeführt: Die meisten Besucher einer Website fänden es sicher nicht sehr nett, wenn ein Script auf einer Webseite Dateien von ihrer Festplatte löschen würde.

Die Scripting-Technologie hat sich jedoch von einer clientseitigen Technologie, die hauptsächlich für Webseitenelemente verwendet wurde, deutlich weiterentwickelt. Mit der Einführung von ASP (Active Server Pages) benötigten Webentwickler, die Möglichkeit Dateioperationen durchzuführen – und mit der Einführung des WSH (Windows Script Host) wurden Dateioperationen außerhalb des Webbrowsers erforderlich.

Als Reaktion auf diese Anforderungen veröffentlichte Microsoft die Script Runtime Library. Hierbei handelt es sich um eine einzelne DLL (dynamic-link library) mit dem Namen *scrrun.dll*, die Scriptautoren in die Lage versetzt, mit dem Dateisystem zu arbeiten. Sie implementiert die folgenden Möglichkeiten:

- Abrufen von Informationen über Elemente des Dateisystems (inklusive Laufwerke, Dateien und Ordner)
- Kopieren, Verschieben und Löschen von Dateien und Ordnern
- Erstellen, Lesen und Schreiben von Textdateien

Außerdem ermöglicht die Script Runtime Library das Erstellen von Dictionaries (Datenstrukturen, die wie Collections funktionieren) und das Kodieren von Scripten (kodierte Scripte können nicht ohne weiteres gelesen werden und schützen so Ihr geistiges Eigentum).

Anmerkung: In diesem Kapitel werden die Objekte *FileSystemObject* und *Dictionary* besprochen. Das *Script Encoder*-Objekt wird jedoch nicht behandelt.

Die Script Runtime Library ist Teil von Windows® 2000. Sie wird auch zusammen mit bestimmten Microsoft-Anwendungen installiert. Hierzu zählen unter anderem:

- Windows Script Host
- VBScript
- Internet Explorer
- Microsoft Office

Das Objekt FileSystemObject

Wie der Name (FileSystemObject – Dateisystemobjekt) schon andeutet, hilft Ihnen das Objekt *FileSystemObject* (FSO) bei der Arbeit mit dem Dateisystem. Es ermöglicht Ihnen, grundlegende Informationen über Dateisystemelemente wie Laufwerke, Ordner und Dateien abzurufen, und stellt außerdem Methoden zur Durchführung von administrativen Aufgaben zur Verfügung – zum Beispiel dem Kopieren, Löschen und Verschieben von Dateien und Ordnern.

Der Name *FileSystemObject* ist nicht ganz passend – denn das Objekt setzt sich eigentlich aus mehreren Objekten zusammen. Die einzelnen Objekte von *FileSystemObject* sehen Sie in Tabelle 4.1.

Tabelle 4.1: Die einzelnen Objekte des Objekts FileSystemObject

Objekt	Beschreibung
Drive	Stellt den Zugriff auf ein Laufwerk oder eine Collection von Laufwerken zur Verfügung.
File	Stellt den Zugriff auf eine Datei oder eine Collection von Dateien zur Verfügung.
Folder	Stellt den Zugriff auf einen Ordner Laufwerk oder eine Collection von Ordnern zur Verfügung.
TextStream	Stellt eine Quelle zur Verfügung, aus der Text gelesen werden kann, in die Text geschrieben werden kann, oder an die Text angehängt werden kann.

Die einzelnen Objekte werden in diesem Kapitel genauer besprochen.

Verwalten von Laufwerken

Als Systemadministrator müssen Sie wissen, welche Laufwerke auf einem Computer installiert sind, und Sie müssen deren Eigenschaften abfragen können – zum Beispiel Laufwerkstyp (Diskettenlaufwerk, Festplatte, CD-ROM), Laufwerksgröße und den freien Speicherplatz des Laufwerks.

Als Scriptautor haben Sie zwei primäre Optionen bei der Verwaltung von Laufwerken: das Objekt *FileSystemObject* und WMI (Windows Management Instrumentation). Im Allgemeinen ist es besser, die Laufwerksverwaltung über WMI durchzuführen – und zwar aus den folgenden Gründen:

- WMI kann einige Eigenschaften zur Verfügung stellen, die Ihnen über das *FileSystemObject* nicht zur Verfügung stehen – zum Beispiel die physikalischen Eigenschaften eines Laufwerks (Köpfe, Sektoren und Zylinder).
- WMI kann wahlweise nur bestimmte Laufwerkstypen zurückgeben (zum Beispiel nur Festplatten). Mit dem Objekt *FileSystemObject* ist dies nicht möglich.
- Mit WMI können Sie Laufwerksinformationen von Remotecomputern abfragen. Mit *FileSystemObject* ist dies nur zusammen mit dem Objekt *WshController* möglich.

Auch wenn WMI also die bessere Technologie für solche Zwecke darstellt, gibt es doch zwei Gründe dafür, dass Sie sich mit dem Objekt *FileSystemObject* auskennen sollten. Erstens kann es auch auf älteren Computern ohne WMI verwendet werden (zum Beispiel Microsoft® Windows® 98) – auch wenn es für einige ältere Betriebssysteme möglich ist, WMI nachträglich zu installieren. Und zweitens wird das Objekt *FileSystemObject* seit langer Zeit von vielen Scriptautoren verwendet. Es ist daher sehr wahrscheinlich, dass Sie auf das Objekt stoßen werden, wenn Sie Scripte anderer Autoren lesen und verwenden.

Eine Collection mit Laufwerken abrufen

Bevor Sie die Laufwerke eines Computers verwalten können, müssen Sie wissen, welche Laufwerke überhaupt zur Verfügung stehen. Das Objekt *FileSystemObject* stellt Ihnen hierzu eine Collection mit allen verfügbaren Laufwerken zur Verfügung – uns zwar inklusive aller Laufwerke mit Wechseldatenträgern und zugeordneten Netzlaufwerke (mit anderen Worten: jedes Laufwerk mit einem Laufwerksbuchstaben).

Um diese Collection zu erhalten, erstellen Sie erst eine Instanz von *FileSystemObject* und eine Referenz auf die Eigenschaft *Drives*. Dann können Sie diese Referenz (*Drives* ist die Collection) für eine Auflistung aller Elemente der Collection nutzen.

Script 4.1 ruft eine Collection mit allen auf dem Computer installierten Laufwerken ab und gibt für jedes Element der Collection den Laufwerksbuchstaben zurück.

Script 4.1: Auflisten aller auf einem Computer installierten Laufwerke

```
1 Set objFSO = CreateObject("Scripting.FileSystemObject")
```

```

2 Set colDrives = objFSO.Drives
3 For Each objDrive in colDrives
4     Wscript.Echo "Laufwerksbuchstabe: " & objDrive.DriveLetter
5 Next

```

Eine komplette Liste aller Eigenschaften des Objekts *FileSystemObject* finden Sie in Tabelle 4.2 weiter unten in diesem Kapitel.

Binden an ein bestimmtes Laufwerk

Wenn Sie bereits wissen, mit welchem Laufwerk Sie arbeiten möchten (zum Beispiel mit Laufwerk *C* oder dem Netzlaufwerk `\\accounting\receivables`), dann können Sie die Methode *GetDrive* verwenden, um eine Referenz auf dieses Laufwerk zu erhalten. Sie müssen so nicht die gesamte Collection durchgehen.

Die Methode *GetDrive* erwartet einen Parameter – den Laufwerksbuchstaben oder den UNC-Pfad des Netzlaufwerks. Den Laufwerksbuchstaben können Sie in unterschiedlichen Varianten angeben:

- C
- C:
- C:\

Script 4.2 erstellt eine Instanz von *FileSystemObject*, verwendet die Methode *GetDrive*, um eine direkte Referenz auf Laufwerk *C* zu erhalten und gibt dann den freien Speicherplatz dieses Laufwerks aus.

Script 4.2: Eine Referenz auf ein einzelnes Laufwerk

```

1 Set objFSO = CreateObject("Scripting.FileSystemObject")
2 Set objDrive = objFSO.GetDrive("C:")
3 Wscript.Echo "Verfügbarer Speicherplatz: " & objDrive.AvailableSpace

```

Wie Sie sehen, ist keine For-Each-Schleife erforderlich. Das liegt daran, dass *GetDrive* ein einzelnes Objekt vom Typ *Drive* statt ein Objekt (Collection) vom Typ *Drives* zurückgibt.

Auflisten der Laufwerkseigenschaften

Die Collection *Drives* wird meist zur Inventarisierung oder Überwachung verwendet – als Systemadministrator müssen Sie wissen, welche Laufwerke auf bestimmten Computern zur Verfügung stehen und außerdem über detaillierte Informationen zu diesen Laufwerken verfügen (zum Beispiel die Seriennummern oder den freien Speicherplatz). Mit einer *Drives*-Collection oder einem einzelnen *Drive*-Objekt können Sie die in Tabelle 4.2 aufgelisteten Eigenschaften abfragen.

Tabelle 4.2: Eigenschaften des Objekts Drive

Eigenschaft	Beschreibung
AvailableSpace	Der freie Speicherplatz auf dem Laufwerk in Byte. Um den freien Speicherplatz in KB zu erhalten, teilen Sie diesen Wert durch 1.024 (es handelt sich um den Speicherplatz, der dem aktuellen Benutzer zur Verfügung steht – wenn Kontingente verwendet werden, kann der Wert kleiner sein als der gesamte zur Verfügung stehende freie Speicherplatz).

DriveLetter	Der dem Laufwerk zugewiesene Laufwerksbuchstabe ohne den angehängten Doppelpunkt. Für das Diskettenlaufwerk wird also zum Beispiel der Buchstabe A verwendet.
DriveType	Ein Integer-Wert, der den Laufwerkstyp angibt: 1 — Wechsellaufwerk 2 — Festplatte 3 — Netzlaufwerk 4 — CD-ROM 5 — RAM-Laufwerk
FreeSpace	Im Gegensatz zu <i>AvailableSpace</i> gibt diese Eigenschaft den gesamten zur Verfügung stehenden freien Festplattenplatz in Byte zurück.
FileSystem	Der Typ des verwendeten Dateisystems (FAT, FAT 32, NTFS).
IsReady	Gibt an, ob auf das Laufwerk zugegriffen werden kann. Wenn sich zum Beispiel keine CD im CD-Laufwerk oder keine Diskette im Diskettenlaufwerk befindet, hat diese Eigenschaft den Wert <i>False</i> .
Path	Pfad zum Laufwerk. Bei lokalen Laufwerken finden Sie unter dieser Eigenschaft den Laufwerksbuchstaben (zum Beispiel A). Bei Netzlaufwerken gibt die Eigenschaft den UNC-Pfad des Laufwerkes zurück (zum Beispiel <i>\\Server1\SharedFolder</i>).
RootFolder	Der Pfad zum Stammordner des Laufwerks.
SerialNumber	Die Seriennummer des Laufwerks. Bei Diskettenlaufwerken oder Netzlaufwerken hat diese Eigenschaft normalerweise den Wert 0.
ShareName	Freigabename eines Netzwerklaufwerks.
TotalSize	Gibt die Gesamtgröße des Laufwerks in Byte zurück (um die Gesamtgröße in KB zu erhalten, teilen Sie den Wert durch 1.024. Für die Gesamtgröße in MB teilen Sie den Wert durch 1.048.576 (1.024 x 1.024).
VolumeName	Der Volumenname des Laufwerks.

Um die Laufwerke eines Computers aufzulisten erstellen Sie eine Instanz von *FileSystemObject* und eine Referenz auf die Eigenschaft *Drives* und verwenden dann eine For-Each-Schleife, um alle Elemente der Collection zu durchlaufen. In der Schleife können Sie für die einzelnen Laufwerke jeweils alle Eigenschaften ausgeben.

Script 4.3: Auflisten der Laufwerkseigenschaften

```

1 Set objFSO = CreateObject("Scripting.FileSystemObject")
2 Set colDrives = objFSO.Drives
3 For Each objDrive in colDrives
4     Wscript.Echo "Verfügbare Speicherplatz: " & objDrive.AvailableSpace
5     Wscript.Echo "Laufwerksbuchstabe: " & objDrive.DriveLetter
6     Wscript.Echo "Laufwerkstyp: " & objDrive.DriveType
7     Wscript.Echo "Dateisystem: " & objDrive.FileSystem
8     Wscript.Echo "Bereit: " & objDrive.IsReady
9     Wscript.Echo "Pfad: " & objDrive.Path
10    Wscript.Echo "Stammordner: " & objDrive.RootFolder

```

```
11     Wscript.Echo "Seriennummer: " & objDrive.SerialNumber
12     Wscript.Echo "Freigabename: " & objDrive.ShareName
13     Wscript.Echo "Gesamtgröße: " & objDrive.TotalSize
14     Wscript.Echo "Volumenname: " & objDrive.VolumeName
15 Next
```

Wenn Sie Script 4.3 unter *CScript* ausführen, erhalten Sie eine ähnliche Ausgabe wie die folgende:

```
Verfügbarer Speicherplatz: 5422272512
Laufwerksbuchstabe: C
Laufwerkstyp: 2
Dateisystem: NTFS
Bereit: Wahr
Pfad: C:
Stammordner: C:\
Seriennummer: 2018221812
Freigabename:
Gesamtgröße: 15356563456
Volumenname: Festplatte
```

Prüfen, ob ein Laufwerk bereit ist

Script 4.3 hat eine potentielle Fehlerquelle. Wenn sich keine Diskette im Diskettenlaufwerk oder keine CD im CD-ROM befindet, dann schlägt das Script mit der Fehlermeldung *Laufwerk nicht bereit* fehl.

Wenn ein Laufwerk nicht bereit ist (normalerweise, weil kein Medium eingelegt ist), dann können Sie nur die folgenden Eigenschaften ohne Fehler abfragen:

- DriveLetter
- DriveType
- IsReady
- ShareName

Glücklicherweise haben Sie über die Eigenschaft *IsReady* die Möglichkeit festzustellen, ob ein Laufwerk tatsächlich nicht bereit ist. Wenn ihr Wert *True* ist, dann können Sie ohne Gefahr alle anderen Eigenschaften abrufen.

Script 4.4 ruft eine Collection mit den verfügbaren Laufwerken ab. Für jedes Laufwerk prüft das Script über die Eigenschaft *IsReady*, ob es die restlichen Eigenschaften ohne Fehler abfragen kann. Wenn dies möglich ist, gibt das Script den Laufwerksbuchstaben und den freien Speicherplatz des Laufwerks zurück. Wenn das Laufwerk nicht bereit ist, dann gibt das Script nur den Laufwerksbuchstaben zurück (eine der vier Eigenschaften, die auch in diesem Zustand gefahrlos abgefragt werden kann).

Script 4.4: Prüfen, ob ein Laufwerk bereit ist

```
1 Set objFSO = CreateObject("Scripting.FileSystemObject")
2 Set colDrives = objFSO.Drives
3 For Each objDrive in colDrives
4     If objDrive.IsReady = True Then
5         Wscript.Echo "Drive letter: " & objDrive.DriveLetter
6         Wscript.Echo "Free space: " & objDrive.FreeSpace
7     Else
8         Wscript.Echo "Drive letter: " & objDrive.DriveLetter
9     End If
10 Next
```

Anmerkung: Mit WMI gibt es solche Probleme nicht. Wenn sich kein Medium im Laufwerk befindet schlägt das Script hier nicht fehl – stattdessen gibt WMI den freien Speicherplatz einfach mit *Null* an.

Verwalten von Ordnern

Wenn Sie wissen in welchem Laufwerk eine Datei gespeichert ist, dann benötigen Sie im nächsten Schritt den Ordner, in der die Datei gespeichert ist. Bei vielen anderen Aufgaben der Systemadministration müssen Sie die Inhalte von Ordnern auflisten, Ordner kopieren, Ordner anlegen und sie auch wieder löschen.

Das Objekt *FileSystemObject* stellt Ihnen nicht nur detaillierte Informationen zu einem Ordner zur Verfügung, sondern bietet Ihnen auch Möglichkeiten zum Kopieren, Verschieben und Löschen an. Außerdem können Sie die Dateien und Ordner in einem Ordner auflisten.

Eine Referenz auf einen Ordner erstellen

In der Windows-Shell sind Ordner COM-Objekte. Das bedeutet, dass Sie vor einem Zugriff auf die Eigenschaften eines Ordners eine Objektreferenz auf diesen Ordner erstellen müssen. Dies können Sie über die Methode *FileSystemObject.GetFolder* durchführen.

Für die Methode *GetFolder* benötigen Sie den Pfad des Ordners. Er kann entweder als lokaler Pfad oder als UNC-Pfad angegeben werden (zum Beispiel *\\accounting\receivables*). Sie dürfen keine Wildcards verwenden. Außerdem können Sie keine Referenz auf mehrere Ordner erstellen. Die folgende Codezeile führt also zu einem Fehler:

```
objFSO.GetFolder("C:\FSO", "C:\Scripts")
```

Wenn Sie mit mehreren Ordnern arbeiten wollen, dann müssen Sie entweder WMI verwenden oder für jeden Ordner eine separate Referenz erstellen.

Script 4.5 erzeugt zum Beispiel eine Referenz auf den Ordner *C:\FSO* und weist diese der Variable *objFolder* zu.

Script 4.5: Erstellen einer Referenz auf einen Ordner

```
1 Set objFSO = CreateObject("Scripting.FileSystemObject")
2 Set objFolder = objFSO.GetFolder("C:\FSO")
```

Mit dem Punkt (.) können Sie eine Referenz auf den aktuellen Ordner erstellen, und mit zwei Punkten (..) erstellen Sie eine Referenz auf den dem aktuellen Ordner übergeordneten Ordner. Mit dem Backslash schließlich erstellen Sie eine Referenz auf den Stammordner. Die folgende Codezeile erstellt zum Beispiel eine Referenz auf den aktuellen Ordner:

```
Set objFolder = objFSO.GetFolder(".")
```

Prüfen, ob ein Ordner vorhanden ist

Bei den meisten Ordneroperationen muss der Ordner bereits vorhanden sein – ein Script kann einen Ordner, der nicht vorhanden ist nicht kopieren, verschieben oder löschen. Ein solcher Versuch wird zur Fehlermeldung *Pfad nicht gefunden* führen.

Um solche Probleme zu vermeiden, können Sie mit der Methode *FolderExists* überprüfen, ob ein Ordner schon vorhanden ist. Die Methode benötigt einen Parameter: den Pfad des Ordners. Sie gibt einen Boolean-Wert zurück. Wenn der Rückgabewert *True* ist, dann ist der Ordner vorhanden – bei *False* nicht.

Script 4.6 verwendet die Methode *FolderExists*, um die Existenz von Ordner *C:\FSO* zu überprüfen. Wenn die Methode den Wert *True* zurückgibt, dann erstellt das Script eine Referenz auf den Ordner. Andernfalls gibt das Script die Nachricht "Ordner ist nicht vorhanden zurück".

Script 4.6: Überprüfen, ob ein Ordner vorhanden ist

```
1 Set objFSO = CreateObject("Scripting.FileSystemObject")
2 If objFSO.FolderExists("C:\FSO") Then
3     Set objFolder = objFSO.GetFolder("C:\FSO")
4     Wscript.Echo "Referenz erstellt."
5 Else
6     Wscript.Echo "Ordner ist nicht vorhanden zurück?"
7 End If
```

Einen Ordner erstellen

Das Objekt *FileSystemObject* ermöglicht es Ihnen über Ihr Script neue Ordner anzulegen. Script 4.6 prüft zum Beispiel, ob ein bestimmter Ordner vorhanden ist. Wenn der Ordner existiert, dann verwendet es die Methode *GetFolder*, um eine Referenz auf den Ordner zu erstellen. Wenn der Ordner nicht vorhanden ist, dann gibt das Script eine Benachrichtigung aus.

Auch wenn das Script so daran gehindert wird einen Fehler zu produzieren, ist dies wohl kaum die beste Lösung. Statt den Benutzer einfach darüber zu informieren, dass der Ordner nicht vorhanden ist, wäre es möglicherweise besser, den Ordner anzulegen. Hierzu kann das Script zum Beispiel die Methode *FileSystemObject.CreateFolder* verwenden. Sie erwartet als Parameter den vollständigen Pfad des neuen Ordners. Script 4.7 erstellt zum Beispiel einen Ordner mit dem Namen *C:\FSO*.

Script 4.7: Einen neuen Ordner erstellen

```
1 Set objFSO = CreateObject("Scripting.FileSystemObject")
2 Set objFolder = objFSO.CreateFolder("C:\FSO")
```

Wenn der zu erstellende Ordner schon vorhanden ist, dann wird ein Fehler erzeugt ("Datei ist bereits vorhanden"). Daher sollten Sie vorher prüfen, ob der neue Ordner bereits vorhanden ist.

Anmerkung: Das Objekt *FileSystemObject* kann nur Ordner auf dem lokalen Computer erstellen. Wenn Sie Ordner auf einem Remotecomputer erstellen müssen, dann müssen Sie das Objekt *WshController* verwenden. Alternativ können Sie den Ordner lokal erstellen und ihn dann über WMI auf den Remotecomputer verschieben (den Ordner müssen Sie jedoch über *FileSystemObject* erstellen, da WMI keine Ordner erstellen kann).

Löschen eines Ordners

Manchmal ist es erforderlich, einen Ordner zu löschen. Zum Beispiel, wenn Sie einen temporären Ordner entfernen möchten. Hierzu können Sie die Methode *DeleteFolder* verwenden. Sie benötigt als Parameter den Pfad des zu löschenden Ordners. Script 4.8 löscht zum Beispiel den Ordner *C:\FSO* und seine gesamten Inhalte.

Script 4.8: Löschen eines Ordners

```
1 Set objFSO = CreateObject("Scripting.FileSystemObject")
2 objFSO.DeleteFolder("C:\FSO")
```

Die Methode *DeleteFolder* löscht den Ordner und alle Inhalte sofort. Es wird keine Bestätigung angefordert, und die Elemente werden auch nicht im Papierkorb abgelegt.

Ordner über Wildcards löschen

Ein Hauptvorteil der Verwaltung über Scripte ist, dass sie viele Aufgaben auf einmal erledigen können. Statt also viele Ordner einen nach dem anderen zu löschen, können Sie diese Aufgabe einfach in einem Schritt über ein Script durchführen.

Das Objekt *FileSystemObject* erlaubt es Ihnen, Ordner über Wildcards zu löschen. Stellen Sie sich zum Beispiel vor, Sie haben es mit der Ordnerstruktur aus Abbildung 4.1 zu tun und möchten alle Unterordner löschen, die mit einem *U* anfangen.



Abbildung 4.1: Beispiel-Ordnerstruktur

Eine solche Aufgabe können Sie über die folgende Scriptzeile ausführen:

```
objFSO.DeleteFolder("C:\FSO\U*")
```

Sie löscht die Ordner *Unterordner 1* und *Unterordner 2*.

Wildcards können nur im letzten Teil des Pfades verwendet werden. Die folgende Scriptzeile wird also zum Fehler "Pfad nicht gefunden" führen:

```
objFSO.DeleteFolder("C:\*\Unterordner 1")
```

Ordner und deren Inhalte kopieren

Über die Methode *CopyFolder* können Sie Ordner und deren Inhalte kopieren. Wenn Sie keine Wildcards verwenden, funktioniert die Methode wie der Befehl *Xcopy /E*: sie kopiert alle Dateien und alle Unterordner, inklusive aller leeren Unterordner. Die Methode benötigt zwei Parameter:

- **Quellordner** – der zu kopierende Ordner. Dieser Ordner kann entweder als lokaler Pfad (C:\Scripts) oder als UNC-Pfad ([\\helpdesk\scripts](#)) angegeben werden.
- **Zielordner** – der Ordner, in dem die Kopien erstellt werden. Auch er kann entweder als lokaler Pfad oder als UNC-Pfad angegeben werden. Wenn der Zielordner nicht vorhanden ist, dann erstellt das Script diesen automatisch.

Die Methode *CopyFolder* akzeptiert noch einen dritten optionalen Parameter: *Überschreiben*. Wenn Sie diesen Parameter auf *True* setzen (das ist die Standardeinstellung), dann werden alle bestehenden Ordner im Zielordner überschrieben. Wenn Sie den Parameter auf *False* setzen, werden die vorhandenen Ordner nicht überschrieben – stattdessen tritt ein Laufzeitfehler auf.

Anmerkung: Die Methode *CopyFolder* hält in dem Moment an, in dem sie auf einen Fehler stößt – auch wenn der Befehl *On Error Resume Next* verwendet wird. Wenn also von 100 zu kopierenden Ordnern erst drei kopiert wurden, bevor ein Fehler auftritt, dann werden die restlichen 97 Ordner nicht mehr kopiert.

Script 4.9 verwendet die Methode *CopyFolder*, um die Inhalte von C:\Scripts nach C:\FSO zu kopieren. Hierbei werden alle bestehenden Ordner im Zielordner überschrieben. Es wird kein Ordner mit dem Namen C:\FSO\Scripts erstellt – stattdessen enthält der Ordner C:\FSO alle Dateien und Ordner aus C:\Scripts. Um einen Ordner mit dem Namen C:\FSO\Scripts zu erstellen, müssen Sie als Zielordner C:\FSO\Scripts angeben.

Script 4.9: Einen Ordner kopieren

```
1 Const OverWriteFiles = True
2 Set objFSO = CreateObject("Scripting.FileSystemObject")
3 objFSO.CopyFolder "C:\Scripts" , "C:\FSO" , OverWriteFiles
```

Anmerkung: Da es sich bei *CopyFolder* um einen einzelnen Vorgang handelt, gibt es keine Möglichkeit, ihren Fortschritt zu überwachen. Sie können nur abwarten, bis die Kopieroperation beendet ist. Wenn Sie den Fortschritt der Kopieroperation überwachen möchten, dann sollten Sie stattdessen das Objekt *Shell.Application* verwenden. Dieses Objekt wird im Abschnitt *Dateien und Ordner* besprochen.

Ordner mit Hilfe von Wildcards kopieren

Die Methode *CopyFolder* kopiert alle Dateien und alle Unterordner eines Ordners. Dieses Verhalten kann jedoch problematisch sein – vielleicht möchten Sie nur die Dateien im Ordner C:\FSO und nicht die in C:\FSO\Subfolder1, C:\FSO\Subfolder2 und C:\FSO\Subfolder3 gespeicherten Dateien kopieren?

Unglücklicherweise gibt es keine einfache Methode, um nur die Dateien im Ordner ohne die Unterordner zu kopieren. Sie können jedoch Wildcards verwenden. Die folgende Scriptzeile kopiert zum Beispiel nur die Ordner, die mit den Buchstaben *log* beginnen:

```
objFSO.CopyFolder "C:\Scripts\Log*" , "C:\Archive", True
```

Wenn die Codezeile ausgeführt wird, dann werden die Ordner C:\Scripts\Logs und C:\Scripts\Logfiles kopiert – zusammen mit den in ihnen gespeicherten Dateien und Unterordnern. Die Dateien im Ordner C:\Scripts werden jedoch nicht kopiert (unabhängig von deren Namen).

Es ist mit *CopyFolder* nicht möglich, nur die Dateien ohne Ordner zu kopieren. Hierzu müssen Sie die *CopyFile* verwenden – sie wird später in diesem Kapitel besprochen.

Verschieben von Ordnern und deren Inhalten

Zum Verschieben von Ordnern verwenden Sie die Methode *MoveFolder*. Sie akzeptiert zwei Parameter:

- **Quellordner** – der zu verschiebende Ordner. Dieser Ordner kann entweder als lokaler Pfad oder als UNC-Pfad angegeben werden.
- **Zielordner** – der Ordner, in den die Quellordner verschoben werden. Auch er kann entweder als lokaler Pfad oder als UNC-Pfad angegeben werden. Wenn der Zielordner nicht vorhanden ist, dann erstellt das Script diesen automatisch.

Wenn der Zielordner noch nicht vorhanden ist, dann wird der Quellordner verschoben. Wenn der Zielordner schon existiert, schlägt die Methode fehl. Sie können mit der Methode keinen bestehenden Ordner überschreiben.

Script 4.10 verschiebt den lokalen Ordner *C:\Scripts* in die Freigabe *\\helpdesk\management*.

Script 4.10: Verschieben eines Ordners

```
1 Set objFSO = CreateObject("Scripting.FileSystemObject")
2 objFSO.MoveFolder "C:\Scripts" , "\\helpdesk\management"
```

Bedenken Sie, dass Sie bei einem Fehler keine Möglichkeit haben, die Aktionen von *MoveFolder* rückgängig zu machen. Wenn zum Beispiel während des Verschiebens in eine Netzwerkfreigabe die Netzwerkverbindung getrennt wird, bleiben einige Dateien auf Computer A, einige Dateien wurden auf Computer B verschoben und einige Dateien sind möglicherweise während des Verschiebens verloren gegangen. Es gibt keine Möglichkeit den ursprünglichen Zustand wiederherzustellen.

Daher sollten Sie den Verschiebevorgang besser über zwei einzelne Methoden durchführen: *CopyFolder* und *DeleteFolder*. Sie löschen den Quelleordner über die Methode *DeleteFolder* erst dann, wenn die Methode *CopyFolder* erfolgreich ausgeführt wurde.

Ordner umbenennen

Das Objekt *FileSystemObject* stellt keine Methode zum Umbenennen von Ordnern zu Verfügung. Sie können eine solche Operation jedoch über die Methode *MoveFolder* durchführen. Stellen wir uns vor, Sie haben den folgenden Pfad:

```
C:\Scripts\PerformanceMonitoring\Servers\Domain Controllers\Current Logs
```

Wenn Sie den Ordner über den Windows Explorer umbenennen, bleibt der Pfad gleich – das Einzige, was sich ändert, ist der letzte Teil:

```
C:\Scripts\PerformanceMonitoring\Servers\Domain Controllers\Archived Logs
```

Mit *MoveFolder* können Sie das gleiche erreichen, indem Sie den Ordner von *C:\Scripts\PerformanceMonitoring\Servers\Domain Controllers\Current Logs* nach *C:\Scripts\PerformanceMonitoring\Servers\Domain Controllers\Archived Logs* verschieben. Das Endergebnis ist exakt das gleiche.

Script 4.11 verwendet *MoveFolder*, um den Ordner *C:\FSO\Samples* in *C:\FSO\Scripts* umzubenennen.

Script 4.11: Ordner über die Methode *MoveFolder* umbenennen

```

1 Set objFSO = CreateObject("Scripting.FileSystemObject")
2 objFSO.MoveFolder "C:\FSO\Samples" , "C:\FSO\Scripts"

```

Verwenden von Ordereigenschaften

Da es sich bei Ordnern um COM-Objekte handelt, verfügen diese natürlich auch über Eigenschaften und sie können aufgelistet werden. Um Informationen über einen bestimmten Ordner abzurufen, können Sie das Objekt *Folder* verwenden – dieses finden Sie unter dem Objekt *FileSystemObject*. Die Eigenschaften des *Folder*-Objekts sehen Sie in Tabelle 4.3.

Tabelle 4.3: Eigenschaften des Objekts *Folder*

Eigenschaft	Beschreibung
Attributes	Ein Bitfeld mit den Attributen des Ordners. Weitere Informationen finden Sie im Abschnitt <i>Verwalten von Ordnerattributen</i> in diesem Kapitel.
DateCreated	Erstellungsdatum des Ordners.
DateLastAccessed	Letztes Zugriffsdatum.
DateLastModified	Letztes Bearbeitungsdatum.
Drive	Laufwerksbuchstabe mit Doppelpunkt (zum Beispiel C:) des Laufwerks, in dem der Ordner gespeichert ist.
Files	Eine Collection mit Objekten vom Typ <i>File</i> zum Zugriff auf die im Ordner gespeicherten Dateien.
IsRootFolder	Ein Boolean-Wert, der anzeigt, ob es sich um einen Stammordner (wie zum Beispiel C:\) handelt.
Name	Name des Ordners ohne Pfadangaben (zum Beispiel <i>System32</i>).
ParentFolder	Name des Ordners, in dem der Ordner gespeichert ist. Für den Ordner <i>C:\Scripts</i> wäre dies zum Beispiel <i>C:\</i> .
Path	Vollständiger Pfad des Ordners (zum Beispiel <i>C:\Windows\System32</i>).
ShortName	Ordnername in der MS-DOS-Syntax mit der 8.3-Namenskonvention. Der Ordner <i>C:\Windows\Programme</i> wird zum Beispiel als <i>Progra~1</i> angezeigt.
ShortPath	Pfadname des Ordners in der MS-DOS-Syntax. Der Ordner <i>C:\Windows\Programme</i> wird zum Beispiel als <i>C:\Windows\Progra~1</i> angezeigt.
Size	Gesamtgröße in Byte aller Inhalte des Ordners. Dies schließt die in dem Ordner gespeicherten Dateien und alle Dateien in Unterordnern ein.
SubFolders	Eine Collection mit den Unterordnern des Ordners. Die Unterordner in den Unterordnern sind nicht in dieser Collection enthalten.
Type	Ein String mit einer Beschreibung des Ordnertyps – meist <i>File Folder</i> .

Auflisten von Ordereigenschaften

Um die Eigenschaften eines Ordners abzufragen, muss ein Script folgendermaßen vorgehen:

1. Eine Instanz des Objekts *FileSystemObject* erstellen.
2. Eine Referenz auf den jeweiligen Ordner über die Methode *GetFolder* erzeugen.
3. Die Eigenschaften aus Tabelle 4.3 ausgeben.

Bei der Arbeit mit den Eigenschaften eines Ordners sollten Sie bedenken, dass die Eigenschaften *Files* und *Subfolders* beide eine Collection zurückgeben. Außerdem handelt es sich bei der

Eigenschaft *Attributes* um ein Bitfeld. Wie Sie mit diesen Eigenschaften umgehen, erfahren Sie in den folgenden Abschnitten dieses Kapitels.

Script 4.12 verwendet die Methode *GetFolder*, um eine Referenz auf den Ordner *C:\FSO* zu erstellen und dann die Eigenschaften des Ordners auszugeben.

Script 4.12: Eigenschaften eines Ordners abfragen

```
1 Set objFSO = CreateObject("Scripting.FileSystemObject")
2 Set objFolder = objFSO.GetFolder("C:\FSO")
3 Wscript.Echo "DateCreated: " & objFolder.DateCreated
4 Wscript.Echo "DateLastAccessed: " & objFolder.DateLastAccessed
5 Wscript.Echo "DateLastModified: " & objFolder.DateLastModified
6 Wscript.Echo "Drive: " & objFolder.Drive
7 Wscript.Echo "IsRootFolder: " & objFolder.IsRootFolder
8 Wscript.Echo "Name: " & objFolder.Name
9 Wscript.Echo "ParentFolder: " & objFolder.ParentFolder
10 Wscript.Echo "Path: " & objFolder.Path
11 Wscript.Echo "ShortName: " & objFolder.ShortName
12 Wscript.Echo "ShortPath: " & objFolder.ShortPath
13 Wscript.Echo "Size: " & objFolder.Size
14 Wscript.Echo "Type: " & objFolder.Type
```

Wenn Sie das Script unter *CScript* ausführen, erhalten Sie die folgende Ausgabe:

DateCreated: 28.03.2004 15:56:33

DateLastAccessed: 28.03.2004 15:56:3

DateLastModified: 28.03.2004 15:56:3

Drive: C:

IsRootFolder: Falsch

Name: fso

ParentFolder: C:

Path: C:\fso

ShortName: FSO

ShortPath: C:\FSO

Size: 0

Type: Dateiordner

Verwalten von Ordnerattributen

Wenn Sie mit der rechten Maustaste auf einen Ordner klicken und *Eigenschaften* auswählen, dann können Sie die Attribute des Ordners festlegen. Wie Sie in Abbildung 4.2 sehen, stehen Ihnen die folgenden Attribute zur Verfügung:

- Schreibgeschützt
- Versteckt
- Archiviert
- Komprimiert

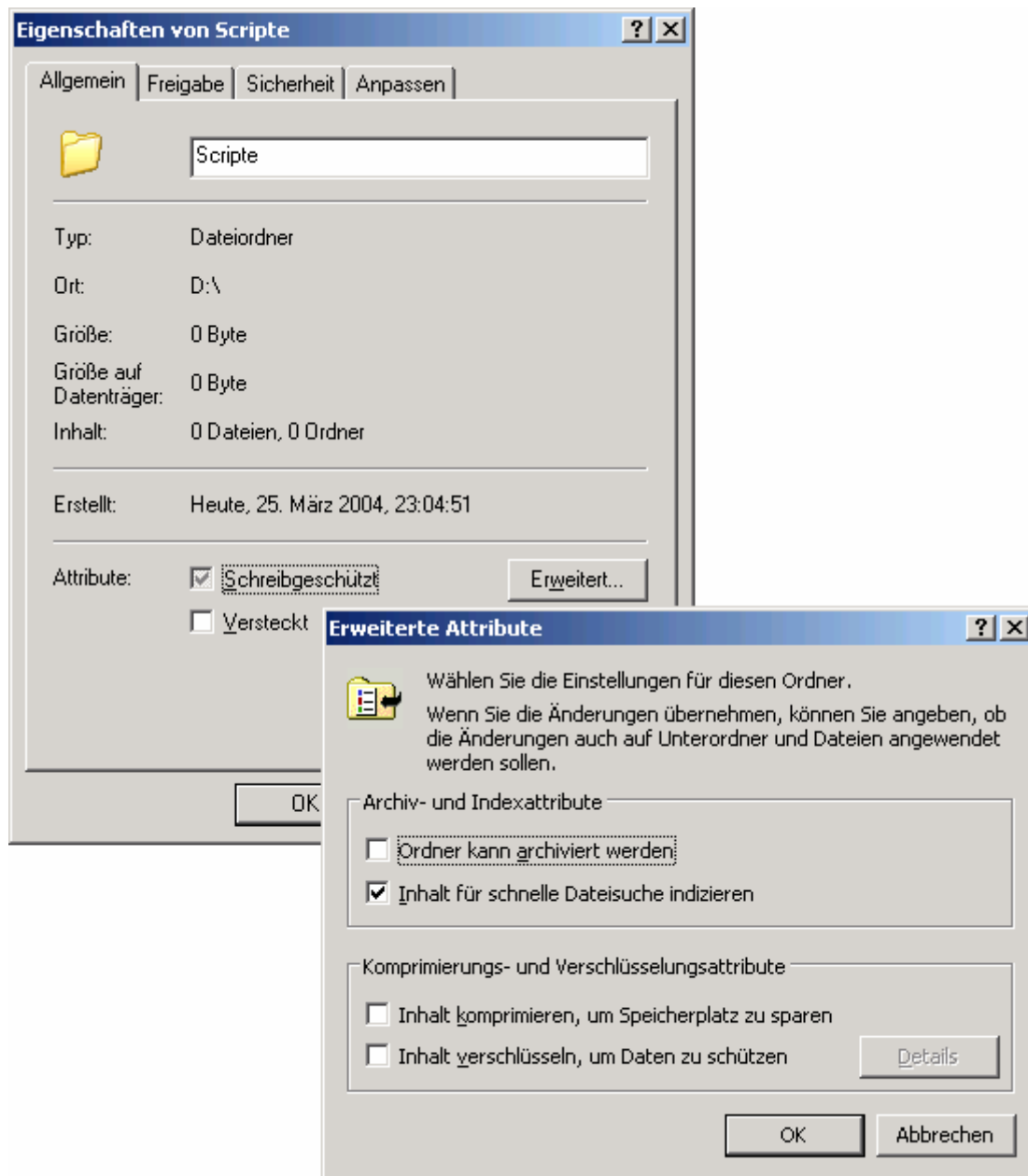


Abbildung 4.2: Eigenschaften eines Ordners

Die Attribute, die Sie über das Objekt *FileSystemObject* abfragen können, sehen Sie in Tabelle 4.4.

Tabelle 4.4: Über *FileSystemObject* abfragbare Ordnerattribute

Konstante	Wert	Beschreibung
Hidden	2	Versteckt - Zeigt an, dass der Ordner versteckt ist.
System	4	System - Zeigt an, dass es sich um einen Systemordner handelt.
Directory	16	Ordner - Der Standardwert für alle Ordner.
Archive	32	Archiv - Das Archiv-Attribut wird zum Beispiel von Sicherungsprogrammen verwendet. Diese stellt über das Attribut fest, welche Dateien und Ordner gesichert werden müssen.
Compressed	2048	Komprimiert - Zeigt an, ob der Ordner komprimiert ist.

Es könnte sein, dass Sie bei der Abfrage der Eigenschaft *Attributes* einige verwirrende Ergebnisse erhalten – zum Beispiel den Wert 20. Dieser Wert steht nicht in Tabelle 4.4. Außerdem erhalten Sie immer nur einen einzelnen Wert zurück – auch wenn der Ordner mehrere der in der Tabelle aufgelisteten Eigenschaften verwendet. Statt zum Beispiel einen Wert der Tabelle zu erhalten (2, 4, 16, 32, 2048), gibt die Abfrage der Eigenschaft den Wert 2102 zurück. Dieses Verhalten liegt daran, dass es sich bei dem zurückgegebenen Wert immer um ein Bitfeld handelt.

Ein Bitfeld ist wie eine Reihe Schalter, die entweder an oder aus sein können. Wenn ein bestimmter Schalter aus ist, dann hat dieser Schalter den Wert 0. Wenn ein bestimmter Schalter an ist, dann hat er im Fall eines Ordners den Wert aus Tabelle 4.4. Der Wert des gesamten Bitfeldes setzt sich aus der Gesamtsumme aller Schalter-Werte zusammen.

In Abbildung 4.3 sehen Sie zum Beispiel eine stark vereinfachte Darstellung eines Bitfeldes. In diesem Beispiel ist nur ein Schalter (*Ordner*) an – er hat den Wert 16 (siehe Tabelle 4.4). Da alle anderen Schalter aus sind haben Sie alle den Wert 0. Die Gesamtsumme aller Schalter ist also 16. Die Eigenschaft *Attribute* des entsprechenden Ordners hätte also den Wert 16.



Abbildung 4.3: Erstes Beispiel für ein Bitfeld

Im Gegensatz dazu sind in Abbildung 4.4 drei Schalter aktiviert. *Versteckt* (mit dem Wert 2), *Ordner* (mit dem Wert 16) und *Komprimiert* (mit dem Wert 248). Der Gesamtwert dieses Bitfeldes wäre also $2 + 16 + 2048 = 2066$. Für diesen Ordner hätte die Eigenschaft *Attributes* also den Wert 2066.



Abbildung 4.4: Zweites Beispiel für ein Bitfeld

Bitfelder arbeiten so, dass jeder mögliche Wert nur mit einer einzigen Schalterkombination erreicht werden kann – der Wert 2066 ist nur mit den Schaltern *Versteckt*, *Komprimiert* und *Ordner* möglich.

Daher können Sie über den Wert der Eigenschaft *Attributes* ermitteln, welcher Schalter gesetzt ist. Glücklicherweise müssen Sie hierzu keine mathematischen Berechnungen durchführen oder alle möglichen Werte kennen. Stattdessen können Sie den logischen Operator *AND* verwenden – mit ihm können Sie feststellen, welche Schalter an bzw. aus sind. Das folgende Codestück prüft zum Beispiel, ob ein Ordner versteckt ist. Wenn dies der Fall ist, dann gibt das Script die Nachricht "Versteckter Ordner" aus:

```
If objFolder.Attributes AND 2 Then  
    Wscript.Echo "Versteckter Ordner"  
End If
```

Der If-Befehl mag ein wenig komisch aussehen. Sie können ihn auch so lesen: "Wenn in der Eigenschaft *Attributes* der Schalter mit dem Wert 2 (Hidden – siehe Tabelle 4.4) gesetzt ist, dann ..."
Der folgende Befehl bedeutet zum Beispiel: "Wenn in der Eigenschaft *Attributes* der Schalter mit dem Wert 16 (System) gesetzt ist, dann ...":

```
If objFolder.Attributes AND 16 Then
```

Script 4.13 erstellt eine Referenz auf den Ordner C:\FSO und zeigt dessen Attribute an.

Script 4.13: Auflisten von Ordnerattributen

```
1 Set objFSO = CreateObject("Scripting.FileSystemObject")
2 Set objFolder = objFSO.GetFolder("C:\FSO")
3 If objFolder.Attributes AND 2 Then
4     Wscript.Echo "Versteckt"
5 End If
6 If objFolder.Attributes AND 4 Then
7     Wscript.Echo "System"
8 End If
9 If objFolder.Attributes AND 16 Then
10    Wscript.Echo "Ordner"
11 End If
12 If objFolder.Attributes AND 32 Then
13    Wscript.Echo "Archiv"
14 End If
15 If objFolder.Attributes AND 2048 Then
16    Wscript.Echo "Komprimiert"
17 End If
```

Ändern von Ordnerattributen

Sie können die im vorherigen Abschnitt kennen gelernten Attribute natürlich auch selbst aktivieren oder deaktivieren – ganz einfach indem Sie die Schalter anschalten oder abschalten. Am einfachsten geht das folgendermaßen:

1. Erstellen Sie eine Referenz auf den Ordner über die Methode *GetFolder*.
2. Prüfen Sie den Attributwert den Sie ändern möchten. Wenn Sie zum Beispiel das Attribut *Versteckt* eines Ordners ausschalten möchten, dann prüfen Sie, ob das Attribut im Moment angeschaltet ist.
3. Wenn der Ordner versteckt ist, dann verwenden Sie den logischen Operator *XOR*, um den Schalter auf *aus* umzustellen. Wenn der Ordner nicht versteckt ist, dann verwenden Sie *XOR* nicht – in diesem Fall würden Sie den Schalter sonst anschalten.

Script 4.14 verwendet den Operator *AND* mit dem Wert *2 (Hidden)*, um festzustellen, ob der Ordner *C:\FSO* versteckt ist. Wenn dies der Fall ist, dann verwendet das Script den logischen Operator *XOR*, um den entsprechenden Schalter auf *aus* umzustellen.

Script 4.14: Ändern von Ordnerattributen

```
1 Set objFSO = CreateObject("Scripting.FileSystemObject")
2 Set objFolder = objFSO.GetFolder("C:\FSO")
3 If objFolder.Attributes AND 2 Then
4     objFolder.Attributes = objFolder.Attributes XOR 2
5 End If
```

Auflisten der Dateien in einem Ordner

Nur zu wissen, dass ein Ordner vorhanden ist, ist meist eher nutzlos. Normalerweise wollen Sie wissen, welche Dateien in dem Ordner gespeichert sind. Hierzu stellt Ihnen das Objekt *Folder* eine Eigenschaft mit dem Namen *Files* zur Verfügung. Diese gibt eine Collection mit allen Dateien im entsprechenden Ordner zurück. Um diese Collection zu erhalten, gehen Sie folgendermaßen vor:

1. Erstellen Sie eine Instanz des Objekts *FileSystemObject*.
2. Verwenden Sie die Methode *GetFolder*, um eine Referenz auf den entsprechenden Ordner zu erstellen.
3. Erstellen Sie eine Objektreferenz auf die Eigenschaft *Files* des Ordnerobjekts.
4. Verwenden Sie eine For-Each-Schleife, um alle Dateien und deren Eigenschaften in der Collection *Files* zu durchlaufen (die Eigenschaften des Objekts *Files* sehen Sie in Tabelle 4.5.).

Script 4.15 ruft die Collection *Files* des Ordnerobjekts *C:\FSO* ab und gibt dann die Eigenschaften *Name* und *Größe* (in Byte) jedes *File*-Objekts in der Collection aus.

Script 4.15: Abfragen der Eigenschaften aller Dateien in einem Ordner

```
1 Set objFSO = CreateObject("Scripting.FileSystemObject")
2 Set objFolder = objFSO.GetFolder("C:\FSO")
3 Set colFiles = objFolder.Files
4 For Each objFile in colFiles
5     Wscript.Echo objFile.Name, objFile.Size
6 Next
```

Wie bei den meisten Collections haben Sie keine Kontrolle darüber, in welcher Reihenfolge die Informationen zurückgegeben werden – Sie können nicht angeben, dass die Dateien nach Name, Größe oder anderen Kriterien sortiert werden sollen. Wenn Sie die Dateien sortieren möchten, dann müssen Sie die Collection in ein *Array*, ein *Dictionary*-Objekt oder ein nicht verbundenes *Recordset* kopieren. Weitere Informationen hierzu finden Sie im Kapitel *Creating Enterprise Scripts* in diesem Buch.

Auflisten von Unterordnern

Zusätzlich zu den in einem Ordner gespeicherten Dateien interessieren Sie sich sicher für die Unterordner eines Ordners. Hierzu stellt Ihnen das *Folder*-Objekt eine Eigenschaft mit dem Namen *Subfolders* zur Verfügung. Über diese Eigenschaft erhalten Sie eine Collection mit den direkten Unterordnern des Ordners. Die Unterordner der Unterordner sind nicht in der Collection enthalten. Bei der folgenden Ordnerstruktur enthält die Collection für den Ordner *Scripte* also nur die Unterordner *Unterordner1* und *Unterordner2*. *Unterordner1A*, *Unterordner1B* usw. sind nicht in der Collection enthalten.

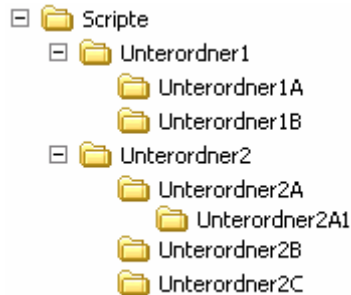


Abbildung 4.5: Eine Beispiel-Ordnerstruktur

Eine Collection mit den Unterordnern erhalten Sie folgendermaßen:

1. Sie erstellen eine Instanz des Objekts *FileSystemObject*.
2. Sie verwenden die Methode *GetFolder* um eine Referenz auf einen Ordner zu erhalten.
3. Sie erstellen eine Referenz auf die Eigenschaft *Subfolders* des Ordnerobjekts (dies ist erforderlich, da es sich bei Collections ja um Objekte handelt).

Nachdem Sie eine Referenz auf die Collection erstellt haben, können Sie die einzelnen Unterordner in der Collection mit einer For-Each-Schleife durchlaufen. Script 4.16 erstellt zum Beispiel eine Referenz auf den Ordner *C:\FSO* und gibt dann den Namen und die Größe der Unterordner aus. Die restlichen Eigenschaften eines Ordnerobjekts haben Sie in Tabelle 4.3 gesehen.

Script 4.16: Auflisten von Unterordnern

```
1 Set objFSO = CreateObject("Scripting.FileSystemObject")
2 Set objFolder = objFSO.GetFolder("C:\FSO")
3 Set colSubfolders = objFolder.Subfolders
4 For Each objSubfolder in colSubfolders
5     Wscript.Echo objSubfolder.Name, objSubfolder.Size
6 Next
```

Unterordner der Unterordner auflisten

Wahrscheinlich reicht es Ihnen nicht aus, die Unterordner eines Ordners zu kennen – stattdessen möchten Sie *alle* Unterordner auflisten. Die Collection *Subfolders* gibt Ihnen zwar die beiden Unterordner von *c:\Scripte* zurück, aber wie kommen Sie an deren Unterordner?

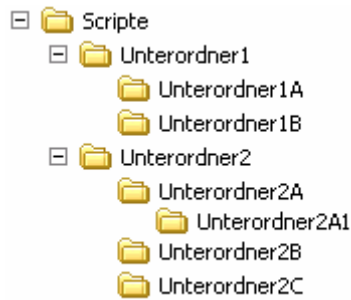


Abbildung 4.6: Eine Beispiel-Ordnerstruktur

Um alle Unterordner abzufragen, müssen Sie eine *rekursive* Funktion verwenden – eine Funktion, die sich selbst aufruft. Weitere Informationen über Rekursion finden Sie im Kapitel *VBScript Primer* in diesem Buch.

Script 4.17 ist ein Beispiel dafür, wie ein Script alle Unterordner eines Ordners (inklusive der Unterordner der Unterordner usw.) auflisten kann. Hierzu geht das Script folgendermaßen vor:

1. Er erstellt eine Instanz des Objekts *FileSystemObject*.
2. Es verwendet die Methode *GetFolder*, um eine Referenz auf *C:\Scripte* zu erstellen. Diese Referenz wird der Subroutine *ShowSubFolders* als Parameter übergeben. Die Subroutine listet alle Unterordner von *C:\Scripts* auf.
3. Das Script ruft eine Collection mit allen Unterordnern von *C:\Scripte* ab. Diese Collection hat zwei Elemente: *Unterordner1* und *Unterordner2*.
4. Das Script gibt den Ordnerpfad für das erste Element der Collection aus. Dann ruft sich die Subroutine mit dem Namen des Ordners als Parameter selbst auf (die Subroutine *ShowSubFolders* wird also jetzt mit dem Parameter *Unterordner1* aufgerufen).
5. Das Script ruft eine Collection mit allen Unterordnern von *Unterordner1* ab. Diese Collection hat zwei Elemente: *Unterordner1A* und *Unterordner1B*.
6. Das Script gibt den Ordnerpfad für das erste Element der Collection aus (*Unterordner1A*). Dann ruft sich die Subroutine mit dem Namen des Ordners als Parameter selbst auf (die Subroutine *ShowSubFolders* wird als mit dem Parameter *Unterordner1A* aufgerufen).
7. Da *Unterordner1A* keine Unterordner mehr hat, wird das nächste Elemente der Collection abgearbeitet. Die Subroutine ruft sich selbst mit dem Parameter *Subfolder1B* auf.
8. Damit ist die Rekursion durch *Unterordner1* abgeschlossen. Das Script macht mit dem zweiten Element der ersten Collection (aus Schritt 3) weiter und wiederholt den gesamten Vorgang.

Script 4.17: Rekursive Auflistung von Unterordnern

```
1 Set FSO = CreateObject("Scripting.FileSystemObject")
2 ShowSubFolders FSO.GetFolder("C:\Scripts")
3 Sub ShowSubFolders(Folder)
4     For Each Subfolder in Folder.SubFolders
5         Wscript.Echo Subfolder.Path
6         ShowSubFolders Subfolder
7     Next
```

8 End Sub

Wenn Sie das Script unter *CScript* ausführen, sollte die Ausgabe so aussehen:

```
C:\scripte\Unteroedner1
C:\scripte\Unteroedner1\Unteroedner1A
C:\scripte\Unteroedner1\Unteroedner1B
C:\scripte\Unteroedner2
C:\scripte\Unteroedner2\Unteroedner2A
C:\scripte\Unteroedner2\Unteroedner2A\Unteroedner 2A-1
C:\scripte\Unteroedner2\Unteroedner2B
C:\scripte\Unteroedner2\Unteroedner2C
```

Um zum Beispiel eine komplette Liste aller Ordner eines Laufwerkes abzufragen, beginnen Sie einfach mit dem Stammordner des entsprechenden Laufwerks (zum Beispiel *C:*).

Verwalten von Dateien

Zur Verwaltung des Dateisystems gehört auch die Verwaltung der einzelnen Dateien – diese müssen kopiert, verschoben, umbenannt und gelöscht werden. Das Objekt *FileSystemObject* bietet auch für diese Aufgaben die richtigen Methoden an.

Eine Referenz auf eine Datei erstellen

Das Objekt *FileSystemObject* stellt Ihnen einige Methoden zur Verfügung, über die ein Script mit Dateien arbeiten kann, ohne eine neue Instanz eines *File*-Objekts zu erstellen – unter anderem *CopyFile* und *DeleteFile*. Für andere Aufgaben ist jedoch ein *File*-Objekt erforderlich. Um zum Beispiel eine Liste der Dateieigenschaften abzurufen, muss ein Script erst eine Referenz auf ein *File*-Objekt erstellen.

Um eine solche Referenz zu erstellen, verwenden Sie die Methode *GetFile*. Das von *Getfile* zurückgegebene *File*-Objekt weisen Sie dann mit dem Schlüsselwort *Set* als Referenz zu einer Variablen zu. Als Parameter erwartet die Methode *GetFile* den Dateinamen (entweder ein lokaler Pfad oder ein UNC-Pfad).

Script 4.18 erstellt zu Beispiel eine Referenz auf die Datei *C:\FSO\ScriptLog.txt*.

Script 4.18: Eine Referenz auf eine Datei erstellen

```
1 Set objFSO = CreateObject("Scripting.FileSystemObject")
2 objFSO.GetFile("C:\FSO\ScriptLog.txt")
```

Im Allgemeinen ist es eine gute Idee, den absoluten Pfad einer Datei als Parameter zu verwenden. Dies stellt sicher, dass das Script immer auf die Datei zugreifen kann. Wenn sich die Datei im gleichen Ordner wie das Script befindet, dann reicht es auch, nur den Dateinamen anzugeben.

Die folgende Scriptzeile erzeugt zum Beispiel eine Referenz auf die Datei *ScriptLog.txt*, die sich im gleichen Ordner wie das Script befindet:

```
objFSO.GetFile("ScriptLog.txt")
```

Wenn sich die Datei im dem Scriptordner übergeordneten Ordner befindet, dann können Sie die folgende Syntax verwenden:

```
objFSO.GetFile(".\ScriptLog.txt")
```

Bitte denken Sie daran, dass das Objekt *FileSystemObject* nicht in den Pfaden der Umgebungsvariable *Path* nach den Dateien sucht. So können Sie beispielsweise den Windows-Rechner starten, indem Sie in der Eingabeaufforderung einfach **calc.exe** eingeben – egal wo Sie sich befinden. Mit *GetFile*-Methode funktioniert dies nicht. Hier müssen Sie immer den korrekten Pfad angeben (für den Windows-Taschenrechner wäre das zum Beispiel *C:\Windows\System32*)

Prüfen, ob eine Datei vorhanden ist

Manchmal ist es sehr wichtig zu prüfen, ob eine Datei vorhanden ist oder nicht. Zum Beispiel wenn Sie die Datei verschieben, kopieren oder löschen möchten. Wenn Sie versuchen eine solche Aktion mit einer nicht vorhandenen Datei durchzuführen, dann kommt es zu einem Laufzeitfehler.

Um solche Fehler zu vermeiden, können Sie vorher mit der Methode *FileExists* prüfen, ob die Datei vorhanden ist. Die Methode benötigt nur einen einzelnen Parameter (die Datei mit Pfad) und gibt einen Boolean-Wert zurück. Wenn die Datei vorhanden ist, dann ist der Rückgabewert *True* ansonsten *False*:

Script 4.19 prüft mit der Methode *FileExists*, ob die Datei *C:\FSO\ScriptLog.txt* vorhanden ist. Wenn dies der Fall ist, dann verwendet das Script die Methode *GetFile*, um eine Referenz auf die Datei zu erstellen. Wenn die Datei nicht vorhanden ist, dann gibt das Script die Fehlermeldung "Datei ist nicht vorhanden" aus.

Script 4.19: Existenz einer Datei überprüfen

```
1 Set objFSO = CreateObject("Scripting.FileSystemObject")
2 If objFSO.FileExists("C:\FSO\ScriptLog.txt") Then
3     Set objFile = objFSO.GetFile("C:\FSO\ScriptLog.txt")
4 Else
5     Wscript.Echo "Datei ist nicht vorhanden."
6 End If
```

Es ist nicht möglich Wildcards zu verwenden – zum Beispiel um zu prüfen, ob eine bestimmte Gruppe von Dateien vorhanden ist (beispielsweise *.txt). Die folgende Codezeile führt zwar nicht zu einem Fehler, gibt jedoch immer den Wert *False* zurück:

```
WScript.Echo objFSO.FileExists("C:\FSO\*.**")
```

Wenn Sie die Existenz einer Datei auf Basis bestimmter Kriterien prüfen müssen, dann haben Sie zwei Möglichkeiten:

- Verwenden Sie die Methode *GetFolder*, um eine Referenz auf den Ordner zu erstellen, rufen Sie die Eigenschaft *Files* ab und gehen Sie die gesamte Collection durch. So können Sie alle Dateien im Ordner überprüfen und feststellen, ob die gesuchte Datei vorhanden ist.
- Verwenden Sie WMI. Mit WMI haben Sie die Möglichkeit speziellere Abfragen zu erstellen – zum Beispiel "alle Dateien mit der Dateierweiterung .doc". Danach können Sie mit der Methode *Count* ermitteln, wie viele Dateien die Abfrage zurückgegeben hat.

Löschen einer Datei

Zum Löschen von Dateien erstellen Sie als erstes eine Instanz des Objektes *FileSystemObject*. Dann rufen Sie die Methode *DeleteFile* mit dem Dateinamen und deren Pfad als Parameter auf. Script 4.20 löscht als Beispiel die Datei *C:\FSO\ScriptLog.txt*.

Script 4.20: Löschen einer Datei

```
1 Set objFSO = CreateObject("Scripting.FileSystemObject")
2 objFSO.DeleteFile("C:\FSO\ScriptLog.txt")
```

Standardmäßig löscht die Methode *DeleteFile* keine schreibgeschützten Dateien – wenn Sie dies versuchen, kommt es zu einem Laufzeitfehler. Um solche Fehler zu vermeiden und um schreibgeschützte Dateien löschen zu können, können Sie den optionalen Parameter *Force* verwenden. Wenn Sie den Parameter auf *True* setzen, werden alle Dateien gelöscht:

```
objFSO.DeleteFile("C:\FSO\ScriptLog.txt", True)
```

Eine Gruppe von Dateien löschen

Innerhalb eines Ordners können Sie mehrere Dateien über Wildcards löschen. Wenn die Dateien über mehrere Ordner verteilt sind, ist dies mit einem einzigen Aufruf von *DeleteFile* jedoch nicht möglich. Um mehrere Dateien in unterschiedlichen Ordner mit einem Aufruf zu löschen (zum Beispiel alle *.TMP*-Dateien auf einem Computer), müssen Sie WMI verwenden.

Um mehrere Dateien in einem Ordner zu löschen, verwenden Sie die Methode *DeleteFile* und geben den Pfad und den Dateinamen einfach mit Wildcards an. Die folgende Codezeile löscht zum Beispiel alle *.doc*-Dateien im Ordner *C:\FSO*:

```
objFSO.DeleteFile("C:\FSO\*.doc")
```

Die folgende Codezeile löscht alle Dateien mit den Buchstaben *log* irgendwo im Dateinamen:

```
objFSO.DeleteFile("C:\FSO\*log.* ")
```

Wie bereits erwähnt, löscht *DeleteFile* standardmäßig keine Dateien, die schreibgeschützt sind. Bei einem solchen Versuch kommt es zu einem Laufzeitfehler und die Methode bricht ab – und zwar unabhängig davon, ob im Script der Befehl *On Error Resume Next* verwendet wurde oder nicht.

Script 4.21 löscht alle *.txt*-Dateien im Ordner *C:\FSO*. Um sicherzustellen, dass alle Dateien gelöscht werden (auch die schreibgeschützten), wird der Parameter *Force* mit dem Wert *DeleteReadOnly* verwendet.

Script 4.21: Löschen einer Gruppe von Dateien

```
1 Const DeleteReadOnly = True
2 Set objFSO = CreateObject("Scripting.FileSystemObject")
3 objFSO.DeleteFile("C:\FSO\*.txt"), DeleteReadOnly
```

Tipp: Was wenn Sie alle Dateien *außer* den schreibgeschützten löschen möchten? In diesem Fall rufen Sie über die Eigenschaft *Files* des *Folder*-Objekts eine Liste aller Dateien ab und gehen diese durch. Hierbei können Sie für jede Datei abfragen, ob sie schreibgeschützt ist und sie gegebenenfalls löschen.

Kopieren einer Datei

Die Methode *CopyFile* dient zum Kopieren von Dateien. Sie müssen zwei Parameter angeben – ein dritter Parameter ist optional:

- **Quelle** (erforderlich) – die zu kopierende Datei. Entweder ein lokaler Pfad oder ein UNC-Pfad.
- **Ziel** (erforderlich) – Pfad und Dateiname. Entweder ein lokaler Pfad oder ein UNC-Pfad. Um die Datei unter dem gleichen Namen in den Zielordner zu kopieren, lassen Sie einfach den Dateinamen weg:
objFSO.CopyFile "C:\FSO\ScriptLog.txt" , "D:\Archive\
Um der Datei am Zielort einen neuen Namen zu gegen verwenden Sie folgende Syntax:
objFSO.CopyFile "C:\FSO\ScriptLog.txt" , "D:\Archive\NewFileName.txt"
Wenn der Zielordner nicht vorhanden ist wird er automatisch erstellt.
- **Überschreiben** (optional) – Standardmäßig kopiert die Methode die Datei nicht, wenn im Zielordner schon eine Datei mit dem gleichen Namen vorhanden ist. Sie können das Überschreiben erzwingen, indem Sie den Parameter auf den Wert *True* setzen.

Script 4.22 kopiert die Datei *C:\FSO\ScriptLog.txt* in den Ordner *D:\Archive*. Um sicherzustellen, dass der Vorgang nicht fehlschlägt, wird der dritte Parameter auf *True* gesetzt.

Script 4.22: Eine Datei kopieren

```
1 Const OverwriteExisting = True
2 Set objFSO = CreateObject("Scripting.FileSystemObject")
3 objFSO.CopyFile "C:\FSO\ScriptLog.txt" , "D:\Archive\" , OverwriteExisting
```

Wenn Sie den Zielordner angeben, dürfen Sie das letzte Slash-Zeichen nicht vergessen. Wenn Sie das Zeichen weglassen, dann wird die Datei nicht in den Ordner kopiert. Stattdessen versucht die Methode eine neue Datei mit dem Ordernamen anzulegen.

Wenn Sie versuchen eine schreibgeschützte Datei zu überschreiben, wird die Methode fehlschlagen – auch dann, wenn Sie den Parameter *OverWrite* auf *True* gesetzt haben. In einem solchen Fall müssen Sie die Datei im Zielordner vorher löschen.

Eine Gruppe von Dateien kopieren

Solange sich die Dateien im gleichen Ordner befinden, können Sie mit Wildcards ganze Dateigruppen kopieren. Script 4.23 kopiert zum Beispiel alle *.txt*-Dateien im Ordner *C:\FSO* nach *D:\Archive*.

Script 4.23: Kopieren von mehreren Dateien

```
1 Const OverwriteExisting = True
2 Set objFSO = CreateObject("Scripting.FileSystemObject")
3 objFSO.CopyFile "C:\FSO\*.txt" , "D:\Archive\" , OverwriteExisting
```

Mit Wildcards und der Methode *CopyFile* haben Sie die Möglichkeit alle Dateien in einem Ordner zu kopieren – ohne die Unterordner. Mit *CopyFolder* hingegen werden sowohl Dateien als auch Unterordner kopiert. Die folgende Scriptzeile kopiert alle Dateien im Ordner *C:\FSO*:

```
objFSO.CopyFile "C:\FSO\*.*" , "D:\Archive\"
```

Verschieben einer Datei

Statt eine Datei zu kopieren, möchten Sie diese möglicherweise verschieben. Hierzu können Sie die Methode *MoveFile* nutzen. Sie arbeitet genauso wie die Methode *CopyFile*: Sie erstellen eine Instanz des Objekts *FileSystemObject* und rufen die Methode *MoveFile* mit zwei Parametern auf:

- Dem kompletten Pfad der zu verschiebenden Datei.
- Dem kompletten Pfad des neuen Speicherortes – inklusive des abschließenden Backslash-Zeichens.

Script 4.24 verschiebt die Datei *C:\FSO\ScriptLog.log* in den Ordner *Archive* auf Laufwerk *D*.

Script 4.24: Verschieben einer Datei

```
1 Set objFSO = CreateObject("Scripting.FileSystemObject")
2 objFSO.MoveFile "C:\FSO\ScriptLog.log" , "D:\Archive\"
```

Mehrere Dateien verschieben

Über Wildcards können Sie mehrere Dateien in einem Vorgang verschieben. Mit dem folgenden Parameter verschieben Sie zum Beispiel alle Dateien im Ordner *FSO*, die mit *data* anfangen:

```
C:\FSO\Data*.*
```

Script 4.25 verschiebt alle Protokolldateien (.log) aus dem Ordner *FSO* auf Laufwerk *C* in den Ordner *Archive* auf Laufwerk *D*.

Script 4.25: Verschieben von mehreren Dateien

```
1 Set objFSO = CreateObject("Scripting.FileSystemObject")
2 objFSO.MoveFile "C:\FSO\*.log" , "D:\Archive\"
```

Eine Datei umbenennen

Das Objekt *FileSystemObject* stellt keine Methode zum Umbenennen von Dateien bereit. Sie können eine Datei jedoch über den gleichen Vorgang umbenennen, den wir bereits beim Umbenennen von Ordnern kennen gelernt haben. Verwenden Sie die Methode *MoveFile* und belassen Sie die Datei in ihrem aktuellen Ordner.

Script 4.26 benennt die Datei *ScriptLog.txt* in *BackupLog.txt* um. Technisch verschiebt das Script die Datei *C:\FSO\ScriptLog.txt* nach *C:\FSO\BackupLog.txt*. Das Endergebnis ist jedoch eine umbenannte Datei.

Script 4.26: Umbenennen von Dateien über die Methode MoveFile

```
1 Set objFSO = CreateObject("Scripting.FileSystemObject")
2 objFSO.MoveFile "C:\FSO\ScriptLog.txt" , "C:\FSO\BackupLog.txt"
```

Abfragen von Dateieigenschaften

Auch Dateien haben einige Eigenschaften, die Sie abfragen können. Auch hierzu verwenden Sie das Objekt *FileSystemObject* – neben vielen anderen Dingen können Sie mit ihm die Eigenschaften einer

oder mehrere Dateien abfragen. Eine vollständige Liste der Eigenschaften des Objekts *File* finden Sie in Tabelle 4.5.

Tabelle 4.5: Eigenschaften des Objekts *File*

Eigenschaft	Beschreibung
Attributes	Ein Bitfeld mit den Attributen der Datei.
DateCreated	Das Erstellungsdatum der Datei.
DateLastAccessed	Das Datum des letzten Dateizugriffs.
DateLastModified	Das Datum der letzten Änderung an der Datei.
Drive	Laufwerksbuchstabe mit Doppelpunkt des Laufwerks, auf dem die Datei gespeichert ist.
Name	Dateiname ohne Pfadinformationen. Für die Datei <i>C:\Windows\System32\Scrrun.dll</i> wäre das zum Beispiel <i>Scrrun.dll</i> .
ParentFolder	Name des Ordners, in dem die Datei gespeichert ist. Für <i>C:\Windows\System32\Scrrun.dll</i> wäre das zum Beispiel <i>System32</i> .
Path	Der vollständige Pfad der Datei (zum Beispiel <i>C:\Windows\System32\Scrrun.dll</i>).
ShortName	Der MS-DOS-Name der Datei (in 8.3-Schreibweise). Für die Datei <i>C:\MySpreadsheet.xls</i> ist das zum Beispiel <i>MySpre~1.xls</i> .
ShortPath	Der Pfad in MS-DOS-Schreibweise (in 8.3-Schreibweise). Für die Datei <i>C:\Windows\Program Files\MyScript.vbs</i> wäre das zum Beispiel <i>C:\Windows\Progra~1\MyScript.vbs</i> .
Size	Gesamtgröße der Datei in Byte.
Type	Ein String mit der Dateiarart (zum Beispiel <i>Microsoft Word Document</i>).

Ein Script kann folgendermaßen auf die Dateieigenschaften zugreifen:

- Eine Instanz des Objekts *FileSystemObject* erstellen.
- Mit der Methode *GetFile* eine Objektreferenz auf eine bestimmte Datei erstellen (die Methode *GetFile* benötigt als Parameter den Pfad der Datei).
- Die Dateieigenschaften ausgeben oder ändern.

Script 4.27 verwendet die *GetFile*-Methode, um eine Referenz auf die Datei *C:\Windows\System32\Scrrun.dll* zu erstellen und deren Eigenschaften auszugeben.

Script 4.27: Auflisten von Dateieigenschaften

```
1 Set objFSO = CreateObject("Scripting.FileSystemObject")
2 Set objFile = objFSO.GetFile("c:\windows\system32\scrrun.dll")
3 Wscript.Echo "DateCreated: " & objFile.DateCreated
4 Wscript.Echo "DateLastAccessed: " & objFile.DateLastAccessed
5 Wscript.Echo "DateLastModified: " & objFile.DateLastModified
6 Wscript.Echo "Drive: " & objFile.Drive
7 Wscript.Echo "Name: " & objFile.Name
8 Wscript.Echo "ParentFolder: " & objFile.ParentFolder
```

```

9 Wscript.Echo "Path: " & objFile.Path
10 Wscript.Echo "ShortName: " & objFile.ShortName
11 Wscript.Echo "ShortPath: " & objFile.ShortPath
12 Wscript.Echo "Size: " & objFile.Size
13 Wscript.Echo "Type: " & objFile.Type

```

Wenn Sie das Script unter *CScript* ausführen erhalten Sie die folgende Ausgabe:

```

Set objFSO = CreateObject("Scripting.FileSystemObject")
Set objFile = objFSO.GetFile("c:\windows\system32\scrrun.dll")
Wscript.Echo "DateCreated: " & objFile.DateCreated
Wscript.Echo "DateLastAccessed: " & objFile.DateLastAccessed
Wscript.Echo "DateLastModified: " & objFile.DateLastModified
Wscript.Echo "Drive: " & objFile.Drive
Wscript.Echo "Name: " & objFile.Name
Wscript.Echo "ParentFolder: " & objFile.ParentFolder
Wscript.Echo "Path: " & objFile.Path
Wscript.Echo "ShortName: " & objFile.ShortName
Wscript.Echo "ShortPath: " & objFile.ShortPath
Wscript.Echo "Size: " & objFile.Size
Wscript.Echo "Type: " & objFile.Type

```

Auflisten der Dateiattribute

Auch Dateien haben Attribute. Wie bei Ordnern werden diese als Bitfeld zurückgegeben (weitere Informationen zu Bitfeldern finden Sie im Abschnitt *Verwalten von Ordnerattributen* dieses Kapitels). In Tabelle 4.6 sehen Sie die verfügbaren Dateiattribute.

Tabelle 4.6: Dateiattribute

Konstante	Wert	Beschreibung
Normal	0	Datei ohne Attribute.
Read-only	1	Datei kann nur gelesen werden.
Hidden	2	Versteckte Datei.
System	4	Datei ist Teil des Betriebssystems.
Archive	32	Datei ist zur Sicherung markiert.
Alias	64	Datei ist eine Verknüpfung, die auf eine andere Datei verweist.

Compressed 2048 Datei ist komprimiert.

Um die Dateiattribute abzufragen, verwenden Sie als erstes die Methode *GetFile*, um eine Referenz auf die Datei zu erstellen. Danach verwenden Sie den logischen Operator *AND*, um die Dateiattribute zu überprüfen.

Script 4.28 erstellt eine Referenz auf die Datei *C:\FSO\ScriptLog.txt* und prüft dann alle Dateiattribute.

Script 4.28: Auflisten von Dateiattributen

```
1 Set objFSO = CreateObject("Scripting.FileSystemObject")
2 Set objFile = objFSO.GetFile("C:\FSO\ScriptLog.txt")
3 If objFile.Attributes AND 0 Then
4     Wscript.Echo "Keine Attribute vorhanden."
5 End If
6 If objFile.Attributes AND 1 Then
7     Wscript.Echo "Read-only."
8 End If
9 If objFile.Attributes AND 2 Then
10    Wscript.Echo "Hidden."
11 End If
12 If objFile.Attributes AND 4 Then
13    Wscript.Echo "System."
14 End If
15 If objFile.Attributes AND 32 Then
16    Wscript.Echo "Archive."
17 End If
18 If objFile.Attributes AND 64 Then
19    Wscript.Echo "Alias."
20 End If
21 If objFile.Attributes AND 2048 Then
22    Wscript.Echo "Compressed."
23 End If
```

Konfigurieren von Dateiattributen

Natürlich haben Sie auch die Möglichkeit die Dateiattribute zu verändern. Dies ist jedoch nur für die folgenden Attribute möglich:

1. ReadOnly
2. Hidden
3. System
4. Archive

Um ein Dateiaattribut zu ändern, muss ein Script folgendermaßen vorgehen:

1. Eine Referenz auf die Datei über *GetFile* erstellen.
2. Die Attribute prüfen, die geändert werden sollen. Wenn Sie zum Beispiel das Attribut *read-only* setzen möchten, dann sollten Sie prüfen, ob das Attribut nicht eventuell bereits gesetzt ist.
3. Wenn das Attribut nicht gesetzt ist, dann verwendet das Script den logischen Operator *XOR*, um das Attribut "umzuschalten". Wenn das Attribut bereits gesetzt ist, dann dürfen Sie *XOR* nicht verwenden – Sie würden das Attribut so "ausschalten".

Script 4.29 verwendet den Operator *AND* um zu prüfen, ob der Schalter mit dem Wert 1 (*read-only*) gesetzt ist. Wenn dies nicht der Fall ist, dann wird der Schalter mit dem Operator *XOR* gesetzt.

Script 4.29: Konfigurieren von Dateiattributen

```
1 Set objFSO = CreateObject("Scripting.FileSystemObject")
2 Set objFile = objFSO.GetFile("C:\FSO\TestScript.vbs")
3 If objFile.Attributes AND 1 Then
4     objFile.Attributes = objFile.Attributes XOR 1
5 End If
```

Über die folgende Codezeile schalten Sie alle Attribute gleichzeitig aus:

```
objFile.Attributes = objFile.Attributes AND 0
```

Den Pfad einer Datei verarbeiten

Manchmal benötigen Sie den Pfad einer Datei oder nur einen Teil eines Pfades – in anderen Fällen vielleicht nur die Dateierweiterung. In all diesen Fällen müssen Sie in der Lage sein, die gewünschten Informationen abzufragen. Das Objekt *FileSystemObject* stellt Ihnen hierzu die in Tabelle 4.7 aufgelisteten Methoden zur Verfügung.

Tabelle 4.7: Methoden zur Abfrage von Dateipfaden

Methode	Beschreibung
GetAbsolutePathName	Gibt den vollständigen Pfad einer Datei zurück (zum Beispiel <i>C:\FSO\Scripts\ScriptLog.txt</i>).
GetParentFolderName	Gibt den Pfad des Ordners zurück, in dem die Datei gespeichert ist (zum Beispiel <i>C:\FSO\Scripts</i>).
GetFileName	Gibt den Dateinamen ohne Pfadinformationen zurück (zum Beispiel <i>ScriptLog.txt</i>).
GetBaseName	Gibt den Basisnamen der Datei zurück (der Dateiname ohne Erweiterung – zum Beispiel

ScriptLog).

`GetExtensionName` Gibt die Erweiterung der Datei zurück (zum Beispiel *txt*).

Script 4.30 fragt die Pfadinformationen der Datei *ScriptLog.txt* ab. Das Script funktioniert nur dann, wenn sich die Datei *ScriptLog.txt* im gleichen Ordner wie das Script befindet. Andernfalls müssten Sie der Methode *GetFile* den vollständigen Pfad zur Zielfeile angeben (zum Beispiel *C:\FSO\Scripts\ScriptLog.txt*).

Script 4.30: Abfragen von Pfadinformationen

```
1 Set objFSO = CreateObject("Scripting.FileSystemObject")
2 Set objFile = objFSO.GetFile("ScriptLog.txt")
3 Wscript.Echo "Absoluter Pfad: " & objFSO.GetAbsolutePathName(objFile)
4 Wscript.Echo "Überg. Ordner: " & objFSO.GetParentFolderName(objFile)
5 Wscript.Echo "Dateiname: " & objFSO.GetFileName(objFile)
6 Wscript.Echo "Basisname: " & objFSO.GetBaseName(objFile)
7 Wscript.Echo "Erweiterung: " & objFSO.GetExtensionName(objFile)
```

Wenn Sie das Script unter *CScript* ausführen, erhalten Sie eine Ausgabe wie die folgende:

```
Absoluter Pfad: C:\scriptlog.txt
Überg. Ordner: C:\
Dateiname: scriptlog.txt
Basisname: scriptlog
Erweiterung: txt
```

Abfragen der Dateiversion

Manchmal müssen Sie als Administrator feststellen, ob eine Datei veraltet ist. Hierzu benötigen Sie die Dateiversion, die Sie über die Methode *GetFileVersion* erhalten. Das Script muss folgendermaßen vorgehen:

1. Eine Instanz des Objekts *FileSystemObject* erstellen.
2. Die Methode *GetFileVersion* aufrufen und ihr als Parameter den Pfad zur Datei übergeben.

Script 4.31 demonstriert diesen Vorgang und fragt die Dateiversion von *Sccrun.dll* ab.

Script 4.31: Abfragen der Dateiversion

```
1 Set objFSO = CreateObject("Scripting.FileSystemObject")
2 Wscript.Echo objFSO.GetFileVersion("c:\windows\system32\sccrun.dll")
```

Wenn Sie das Script auf einem Computer unter Windows 2000 mit WSH 5.6 ausführen, erhalten Sie die in Abbildung 4.7 zu sehende Ausgabe.

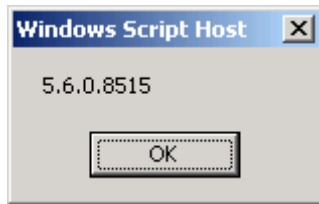


Abbildung 4.7: Versionsnummer der Datei *Scrrun.dll*

Versionsnummern setzen sich normalerweise aus vier Teilen zusammen. Die Versionsnummer *5.6.0.6626* enthält zum Beispiel die folgenden Informationen:

1. **5** - Die Hauptversionsnummer.
2. **6** - Die Unterversionsnummer. Hauptversion und Unterversion zusammen sind die Information, die normalerweise mit "Versionsnummer" gemeint ist. In unserem Beispiel würden Sie zum Beispiel wahrscheinlich eher von *Version 5.6* statt von *Version 5.6.0.6626* sprechen.
3. **0** – Die Buildnummer (normalerweise *0*).
4. **6626** – Die Dateinummer.

Nicht alle Dateien stellen Versionsnummern zur Verfügung. Bei den meisten ausführbaren Dateien und DLLs sind jedoch Versionsnummern vorhanden.

Textdateien lesen und schreiben

Eins der praktischsten Werkzeuge eines Systemadministrators ist die Textdatei. Das scheint im Zeitalter von 3D-Grafik und High-End-Datenbanken erst einmal schwer vorstellbar – trotzdem hat eine einfache Textdatei viele Vorteile. Sie ist klein und leicht zu pflegen, es wird keine zusätzliche Software benötigt (Notepad.exe reicht aus) und sie ist extrem portabel.

Mit Textdateien steht Ihnen ein einfacher Weg zur Verfügung, um Informationen in ein Script und aus einem Script heraus zu übertragen. Textdateien können Argumente enthalten, die Sie ansonsten direkt in das Script schreiben müssten. Sie können Scriptausgaben sehr einfach in einer Textdatei speichern (natürlich könnten Sie die Daten auch direkt in einer Datenbank schreiben – dies wäre jedoch viel komplizierter).

Das Objekt *FileSystemObject* bietet Ihnen daher sowohl Methoden zum Lesen als auch zum Schreiben von Textdateien an.

Textdateien erstellen

Sie haben die Möglichkeit mit vorhandenen Dateien zu arbeiten oder neue Textdateien anzulegen. Um eine neue Textdatei anzulegen benötigen Sie eine Instanz des Objekts *FileSystemObject*. Mit diesem rufen Sie dann die Methode *CreateTextFile* auf, und übergeben ihr den Dateinamen der neuen Datei als Parameter.

Script 4.32 erstellt eine neue Textdatei mit dem Namen *ScriptLog.txt* im Ordner *C:\FSO*.

Script 4.32: Erstellen einer Textdatei

```
1 Set objFSO = CreateObject("Scripting.FileSystemObject")  
2 Set objFile = objFSO.CreateTextFile("C:\FSO\ScriptLog.txt")
```

Wenn die Datei noch nicht vorhanden ist, dann erstellt die Methode *CreateTextFile* diese. Wenn die Datei bereits existiert, wird sie mit einer neuen, leeren Textdatei überschrieben. Wenn Sie nicht

möchten, dass vorhandene Dateien überschrieben werden, dann können Sie einen zweiten, optionalen Parameter verwenden: *Overwrite*. Wenn Sie diesen Parameter auf *True* setzen, dann werden vorhandene Dateien überschrieben (*True* ist auch der Standardwert). Mit *False* werden die vorhandenen Dateien nicht überschrieben. Die folgende Scriptzeile überschreibt die vorhandene Datei nicht:

```
Set objFile = objFSO.CreateTextFile("C:\FSO\ScriptLog.txt", False)
```

Wenn Sie den Parameter auf *False* setzen und die Datei bereits vorhanden ist, dann kommt es zu einem Laufzeitfehler. Daher sollten Sie vor dem Anlegen der Datei prüfen, ob diese schon vorhanden ist und in diesem Fall einen anderen Dateinamen wählen.

Dateinamen in einem Script erstellen

Ein Weg, um das oben beschriebene Problem mit vorhandenen Dateien zu umgehen, besteht darin, einen eindeutigen Dateinamen zu generieren. Da diese Dateinamen dann jedoch nicht sehr aussagekräftig sind, handelt es sich hierbei allerdings nicht um den besten Ansatz. Trotzdem kann ein eindeutiger Dateiname in einigen Situationen sehr nützlich sein – zum Beispiel bei temporären Dateien. Einen solchen Dateinamen erzeugen Sie mit der Methode *GetTempFileName*.

Das Script muss zuerst eine Instanz von *FileSystemObject* erstellen und ruft dann die Methode *GetTempName* auf (sie benötigt keine Parameter). Script 4.33 erstellt zum Beispiel in einer Schleife 10 zufällige Dateinamen.

Script 4.33: Generieren von Dateinamen

```
1 Set objFSO = CreateObject("Scripting.FileSystemObject")
2 For i = 1 to 10
3     strTempFile = objFSO.GetTempName
4     Wscript.Echo strTempFile
5 Next
```

Wenn Sie das Script unter *Cscript* ausführen, dann erhalten Sie eine ähnliche Ausgabe wie die folgende:

```
rad646E9.tmp
radEC50C.tmp
rad0C40A.tmp
radE866E.tmp
rad77F3D.tmp
rad19970.tmp
rad7A21A.tmp
radB9DDC.tmp
rad84930.tmp
rad92199.tmp
```

Anmerkung: Die von *GetTempName* generierten Dateien sind nicht auf jeden Fall eindeutig. Dies liegt teilweise an dem zur Generierung der Dateinamen verwendeten Algorithmus und teilweise an der endlichen Zahl der möglichen Dateinamen. Die generierten Dateinamen haben immer acht Zeichen – die ersten drei Zeichen sind immer *rad*. Es gibt also nur 9.894 eindeutige Dateinamen. Wenn Sie mehr Dateinamen erzeugen, kommt es zwangsläufig zu Dubletten.

Script 4.34 demonstriert die Verwendung von *GetTempName* bei der Erstellung einer Datei. Das Script geht folgendermaßen vor:

1. Es erstellt eine Instanz von *FileSystemObject*.
2. Es erstellt in der Variablen *strPath* eine Referenz auf den Ordner, in dem die Datei erstellt werden soll (*C:\FSO*).
3. Es verwendet die Methode *GetTempName*, um einen eindeutigen Dateinamen zu erstellen.
4. Es verwendet die Methode *BuildPath*, um den Ordernamen mit dem Dateinamen zu kombinieren und so einen vollständigen Pfad für die neue Datei zu erstellen. Dieser Pfad wird in der Variable *strFullName* gespeichert.
5. Es ruft die Methode *CreateTextFile* auf und verwendet die Variable *strFullName* aus dem Parameter für die Methode.

Nachdem die Datei erstellt wurde, schließt das Script diese sofort wieder. In einem Produktionsscript würde das Script nun in die Datei schreiben.

Script 4.34: Erstellen und benennen einer Textdatei

```
1 Set objFSO = CreateObject("Scripting.FileSystemObject")
2 strPath = "C:\FSO"
3 strFileName = objFSO.GetTempName
4 strFullName = objFSO.BuildPath(strPath, strFileName)
5 Set objFile = objFSO.CreateTextFile(strFullName)
6 objFile.Close
```

Öffnen von Textdateien

Die Arbeit mit Textdateien ist ein Prozess mit drei Schritten. Bevor Sie etwas anderes machen, müssen Sie die Datei öffnen (alternativ können Sie auch eine neue Textdatei erstellen – diese ist dann automatisch geöffnet). Durch das Öffnen erhalten Sie ein Objekt vom Typ *TextStream*.

Nachdem Sie über eine Referenz auf das *TextStream*-Objekt verfügen, können Sie aus der Datei lesen oder in die Datei schreiben. Es ist nicht möglich, beide Vorgänge gleichzeitig durchzuführen. Sie können zum Beispiel eine Datei nicht öffnen, Daten lesen und dann sofort Daten in die geöffnete Datei schreiben. Stattdessen müssen Sie die Datei nach dem Lesevorgang schließen, sie neu öffnen und dann in die Datei schreiben. Dies liegt daran, dass eine Textdatei entweder zum Lesen oder zum Schreiben geöffnet wird. Wenn Sie eine neue Textdatei erstellen, dann ist diese automatisch zum Schreiben geöffnet (zu lesen gibt es ja normalerweise auch nichts).

Im letzten Schritt müssen Sie die Datei dann noch schließen. Die ist zwar nicht unbedingt erforderlich (die Datei wird auch geschlossen, wenn das Script beendet wird), um ein korrektes Script zu schreiben sollten Sie diesen Schritt aber trotzdem durchführen.

Um eine Textdatei zu öffnen, gehen Sie folgendermaßen vor:

1. Erstellen Sie eine Instanz von *FileSystemObject*.
2. Verwenden Sie die Methode *OpenTextFile*, um eine Textdatei zu öffnen. Sie benötigt zwei Parameter – den Pfadnamen der Datei und einen der folgenden Modi:
 - **ForReading (1)** – Die Datei wird im Lesemodus geöffnet. Schreiben in die Datei ist nicht möglich (hierzu müssten Sie die Datei schließen und dann im Modus *ForWriting* oder *ForAppending* neu öffnen).
 - **ForWriting (2)** – In diesem Modus werden die vorhandenen Dateien der Textdatei mit den neuen Daten überschrieben.
 - **ForAppending (8)** – In diesem Modus werden die neuen Dateien an die bestehenden Daten der Datei angehängt.

Wenn Sie versuchen im Modus *ForReading* in eine Datei zu schreiben, erhalten Sie den Fehler "Falscher Dateimodus". Auch wenn Sie versuchen andere Dateien als Textdateien zu öffnen, kommt es zu diesem Fehler (auch bei HTML- und XML-Dateien handelt es sich um Textdateien).

Als zweiten Parameter können Sie entweder den Wert (zum Beispiel 1) oder eine Konstante (zum Beispiel *ForReading*) angeben. Das folgende Script demonstriert beide Methoden:

```
Const ForReading = 1

Set objFSO = CreateObject("Scripting.FileSystemObject")

Set objFile = objFSO.OpenTextFile("C:\FSO\ScriptLog.txt", ForReading)

Set objFile2 = objFSO.OpenTextFile("C:\FSO\ScriptLog2.txt", 1)
```

Allerdings müssen Sie die Konstante erst definieren – das liegt daran, dass VBScript leider keinen Zugriff auf die Konstanten von COM-Objekten hat (und *ForReading* ist eine Konstante des COM-Objekts *FileSystemObject* – im Gegensatz zum Beispiel zur Konstante *True*, bei der es sich um eine Konstante von VBScript handelt). Das folgende Script bricht mit der Fehlermeldung "Ungültiger Prozeduraufruf oder Argument" ab, da die Konstante *ForReading* nicht definiert wurde:

```
Set objFSO = CreateObject("Scripting.FileSystemObject")

Set objFile = objFSO.OpenTextFile("C:\FSO\ScriptLog.txt", ForReading)
```

Script 4.35 öffnet die Datei *C:\FSO\ScriptLog.txt* zum Lesen und definiert hierzu die Konstante *ForReading* mit dem Wert 1.

Script 4.35: Eine Textdatei zum Lesen öffnen

```
1 Const ForReading = 1

2 Set objFSO = CreateObject("Scripting.FileSystemObject")

3 Set objFile = objFSO.OpenTextFile("C:\FSO\ScriptLog.txt", ForReading)
```

Schließen von Textdateien

Alle vom Script geöffneten Textdateien werden beim Beenden des Scripts automatisch geschlossen. Daher müssen Sie dies nicht unbedingt selbst durchführen. Sie sollten sich jedoch trotzdem selbst darum kümmern. Nicht nur, weil es eine saubere Programmierung so erfordert, sondern auch um zu vermeiden, dass folgende Probleme mit der geöffneten Datei auftreten:

- **Löschen der Datei** – Wenn Sie versuchen eine geöffnete Datei zu löschen, tritt der Laufzeitfehler "Zugriff verweigert" auf.
- **Neu-Einlesen der Datei** – Es könnte sein, dass Sie im selben Script ein zweites Mal auf die Textdatei zugreifen möchten. Wenn Sie versuchen eine bereits geöffnete Datei ein zweites Mal zu öffnen, erhalten Sie keinen Laufzeitfehler. Das Ergebnis wird allerdings auch nicht Ihren Erwartungen entsprechen. Das folgende Script liest zum Beispiel aus einer Textdatei und gibt die gelesenen Informationen aus. Dann versucht es den Vorgang zu wiederholen, ohne die Datei vorher zu schließen:

```
Set objFSO = CreateObject("Scripting.FileSystemObject")

Set objFile = objFSO.OpenTextFile("C:\FSO\ScriptLog.txt", 1)

Wscript.Echo "Erstes Lesen der Datei:"

strContents = objFile.ReadAll

Wscript.Echo strContents

Wscript.Echo "Zweites Lesen der Datei:"

Do While objFile.AtEndOfStream = False

    strLine = objFile.ReadLine

    Wscript.Echo strLine

Loop
```

Wenn Sie das Script unter *Cscript* ausführen, erhalten Sie die folgende Ausgabe:

```
Erstes Lesen der Datei:

Zeile 1.

Zeile 2.

Zeile 3.

Zweites Lesen der Datei:
```

Bei erstem Lesen werden die erwarteten Dateiinhalte ausgegeben. Bei zweitem Lesen kommt es jedoch zu keinerlei Ausgabe. Dies liegt daran, dass das Ende der Datei bereits beim ersten Lesen erreicht wurde. Um die Datei ein zweites Mal zu lesen, müssen Sie sie erst schließen und dann neu öffnen. Sie können nicht einfach zum Anfang der Datei springen und dann von vorne beginnen.

Um eine Textdatei zu schließen, wird die Methode *Close* des Objekts *TextStreamObject* verwendet. Script 4.36 demonstriert dies, indem es erst eine Instanz von *FileSystemObject* erstellt, dann eine Textdatei öffnet (*C:\FSO\ScriptLog.txt*) und diese dann wieder schließt.

Script 4.36: Öffnen und Schließen einer Textdatei

```
1 Set objFSO = CreateObject("Scripting.FileSystemObject")

2 Set objFile = objFSO.OpenTextFile("C:\FSO\ScriptLog.txt", 1)

3 objFile.Close
```

Lesen von Textdateien

Lesen aus einer Textdatei ist ein Standardverfahren bei vielen Administrationsaufgaben. Zum Beispiel:

- **Lesen von Kommandozeilenargumenten** – In der Textdatei könnte zum Beispiel eine Liste der Computer gespeichert sein, auf die das Script zugreifen soll.
- **Eine Protokolldatei automatisch nach bestimmten Einträgen durchsuchen** – Sie können zum Beispiel nach Fehlereinträgen suchen.

Mit dem Objekt *FileSystemObject* können Sie aus Textdateien lesen. Hierbei sollten Sie jedoch die folgenden Einschränkungen bedenken:

- **Sie können nur ASCII-Textdateien lesen** – Sie können keine Unicode-Dateien oder Binärdateien (zum Beispiel Microsoft Word oder Microsoft Excel lesen).
- **Sie können nur vom Anfang zum Ende der Textdatei lesen** – Sie können nicht rückwärts lesen oder zu Zeilen oder Zeichen in der Textdatei zurückspringen.
- **Sie können eine Datei nicht für gleichzeitiges Lesen und Schreiben öffnen**

Die zum Lesen verfügbaren Dateien von *FileSystemObject* sehen Sie in Tabelle 4.8. Die Beispiele gehen von einer Textdatei aus, die folgendermaßen aussieht:

```
Alerter, Share Process, Running, Auto, LocalSystem,  
AppMgmt, Share Process, Running, Manual, LocalSystem,  
Ati HotKey Poller, Own Process, Stopped, Auto, LocalSystem,
```

Tabelle 4.8: *FileSystemObject*-Methoden zum Lesen aus Textdateien

Methode Beschreibung

Read Liest die angegebene Anzahl an Zeichen. Der folgende Befehl liest zum Beispiel die ersten 12 Zeichen der ersten Zeile ("Alerter,Shar") und speichert diese in der Variable *strText*:

```
strText = objTextFile.Read(12)
```

ReadLine Liest eine Zeile auf der Textdatei. Der folgende Befehl liest zum Beispiel die erste Zeile ("Alerter,Share Process,Running,Auto,LocalSystem") und speichert sie in der Variable *strText*:

```
strText = objTextFile.ReadLine
```

ReadAll Liest die gesamte Textdatei.

Skip Überspringt die angegebene Anzahl an Zeichen. Der folgende Befehl überspringt zum Beispiel die ersten 12 Zeichen. Alle weiteren Leseoperationen beginnen also bei Zeichen 13 ("e Process,Running,Auto,LocalSystem"):

```
objTextFile.Skip(12)
```

Überspringt eine Zeile. Der folgende Code liest zum Beispiel die erste und die dritte Zeile:

SkipLine

```
strText = objTextFile.Readline objTextFile.SkipLine strText =  
objTextFile.Readline
```

Die Größe einer Datei prüfen

Manchmal sind die von Windows erstellten Textdateien leer – das bedeutet, dass in der Datei keine Zeichen enthalten sind und sie eine Größe von 0 Byte hat. Dies kommt zum Beispiel bei Protokollen vor, die so lange leer sind bis ein Ereignis aufgetreten ist.

Leere Dateien sind ein Problem für Scriptautoren, da VBScript einen Laufzeitfehler erzeugt, wenn Sie versuchen aus einer leeren Datei zu lesen.

Wenn die Möglichkeit besteht, dass die Datei leer sein könnte, dann vermeiden Sie den Fehler, indem Sie vor einem Leseversuch die Dateigröße testen.

Script 4.37 erstellt eine Referenz auf die Datei *C:\Windows\Netlogon.log*. Dann prüft das Script die Größe der Datei – wenn diese größer 0 ist, wird die Datei geöffnet und gelesen. Andernfalls wird eine Fehlermeldung ausgegeben.

Script 4.37: Prüfen der Dateigröße

```
1 Set objFSO = CreateObject("Scripting.FileSystemObject")
2 Set objFile = objFSO.GetFile("C:\Windows\Netlogon.log")
3 If objFile.Size > 0 Then
4     Set objReadFile = objFSO.OpenTextFile("C:\Windows\Netlogon.log", 1)
5     strContents = objReadFile.ReadAll
6     Wscript.Echo strContents
7     objReadFile.Close
8 Else
9     Wscript.Echo "Die Datei ist leer."
10 End If
```

Lesen einer gesamten Textdatei

Das einfachste Verfahren, um eine gesamte Datei einzulesen, ist die Methode *ReadAll*. Sie rufen die Methode einfach auf, und speichern ihren Rückgabewert in einer Variablen.

Auch wenn Sie mehrere Textdateien zusammenfassen möchten, ist die Methode *ReadAll* die beste Wahl. Stellen Sie sich zum Beispiel vor, Sie haben eine Gruppe von täglichen Protokolldateien, die Sie in einer wöchentlichen Protokolldatei zusammenfassen möchten. Ein solches Script würde folgendermaßen vorgehen:

1. Es öffnet die Textdatei für Montag, liest die gesamten Dateien über *ReadAll* ein und speichert diese in einer Variablen.
2. Es öffnet die wöchentliche Protokolldatei im Modus *Append* und schreibt die Daten aus der Variablen in diese Datei.
3. Es wiederholt die Schritte 1 und 2 für alle verbleibenden täglichen Protokolldateien.

Die Methode *ReadAll* benötigt keine Parameter. Script 4.38 öffnet die Textdatei *C:\FSO\ScriptLog* und liest den gesamten Inhalt der Datei über *ReadAll* aus.

Script 4.38: Lesen einer Textdatei über die Methode *ReadAll*

```
1 Const ForReading = 1
2 Set objFSO = CreateObject("Scripting.FileSystemObject")
3 Set objFile = objFSO.OpenTextFile("C:\FSO\ScriptLog.txt", ForReading)
4 strContents = objFile.ReadAll
```

```
5 Wscript.Echo strContents
6 objFile.Close
```

Zeilenweises Lesen einer Textdatei

Bei der Systemadministration haben Sie es meist mit Textdateien zu tun, die als "Datenbankersatz" verwendet werden. Jede Textzeile ist hier ein Datensatz. Ein Script könnte zum Beispiel die Servernamen, auf die es zugreifen soll, aus der Textdatei lesen. Diese Textdatei wird dann so oder ähnlich aussehen:

```
atl-dc-01
atl-dc-02
atl-dc-03
atl-dc-04
```

In einem solchen Fall möchten Sie die Textdatei wahrscheinlich zeilenweise auslesen. Hierzu steht Ihnen die Methode *ReadLine* zur Verfügung. Sie verwenden die Methode, indem Sie eine Do-Loop-Schleife so lange ausführen, bis die Eigenschaft *AtEndOfStream* den Wert *True* hat (das bedeutet, dass das Ende der Datei erreicht wurde). In der Schleife lesen Sie jeweils eine einzelne Textzeile über die Methode *ReadLine* ein und führen die gewünschte Aktion aus.

Script 4.39 demonstriert dieses Verfahren, indem es die Datei *C:\FSO\ServerList.txt* öffnet und diese dann zeilenweise einliest.

Script 4.39: Zeilenweises Einlesen einer Datei über die Methode *ReadLine*

```
1 Set objFSO = CreateObject("Scripting.FileSystemObject")
2 Set objFile = objFSO.OpenTextFile("C:\FSO\ServerList.txt", 1)
3 Do Until objFile.AtEndOfStream
4     strLine = objFile.ReadLine
5     Wscript.Echo strLine
6 Loop
7 objFile.Close
```

Eine Textdatei von unten nach oben "lesen"

Wie schon erwähnt, können Sie mit dem Objekt *FileSystemObject* eine Datei nur von vorne nach hinten lesen. Sie können nicht hinten beginnen und rückwärts lesen. Manchmal ist das ein Problem – zum Beispiel bei der Arbeit mit Protokolldateien. Stellen Sie sich vor, Sie möchten die neusten Einträge einer Protokolldatei (also die letzten) zuerst lesen und sich dann zu den ältesten vorarbeiten.

Ein Script muss in einem solchen Fall folgendermaßen vorgehen:

1. Es erstellt einen Array, der die einzelnen Zeilen der Textdatei speichert.
2. Es liest die einzelnen Zeilen über die Methode *ReadLine* ein und speichert diese jeweils im Array.

3. Es ruft den Inhalt des Array ab, beginnt jedoch mit dem letzten Element des Arrays (also mit der Zeile, die als letztes gelesen wurde bzw. die als letzte Zeile in der Datei steht).

Script 4.40 demonstriert dieses Verfahren mit der Datei *C:\FSO\ScriptLog.txt*. Es speichert die einzelnen Zeilen im Array *arrFileLines*. Nachdem alle Zeilen der Datei gelesen wurden, werden die einzelnen Elemente des Arrays in umgekehrter Reihenfolge ausgegeben. Hierzu wird eine For-Schleife verwendet. Sie beginnt mit dem letzten Element (die Obergrenze des Arrays – Upper Bound – UBOUND) und arbeitet sich zum ersten Elemente vor (die Untergrenze des Arrays – Lower Bound – LBOUND).

Script 4.40: Umgekehrtes "Lesen" aus einer Textdatei

```
1 Dim arrFileLines()  
2 i = 0  
3 Set objFSO = CreateObject("Scripting.FileSystemObject")  
4 Set objFile = objFSO.OpenTextFile("C:\FSO\ScriptLog.txt", 1)  
5 Do Until objFile.AtEndOfStream  
6     Redim Preserve arrFileLines(i)  
7     arrFileLines(i) = objFile.ReadLine  
8     i = i + 1  
9 Loop  
1(objFile.Close  
1:For l = Ubound(arrFileLines) to LBound(arrFileLines) Step -1  
1:    Wscript.Echo arrFileLines(l)  
1:Next
```

Nehmen wir an, die Textdatei hat den folgenden Inhalt:

```
6/19/2002    Success  
6/20/2002    Failure  
6/21/2002    Failure  
6/22/2002    Failure  
6/23/2002    Success
```

Dann würde die Ausgabe des Scripts folgendermaßen aussehen:

```
6/23/2002    Success  
6/22/2002    Failure  
6/21/2002    Failure  
6/20/2002    Failure
```

Eine Textdatei Zeichen für Zeichen einlesen

Eine Textdatei kann auch einzelne Werte enthalten. In einem solchen Fall hat zum Beispiel jeder Werte eine bestimmte Länge. Wert 1 ist 15 Zeichen lang, Wert 2 ist 10 Zeichen lang, usw. Eine solche Textdatei könnte zum Beispiel folgendermaßen aussehen:

Server	Value	Status
atl-dc-01	19345	OK
atl-printserver-02	00042	OK
atl-win2kpro-05	00000	Failed

Wenn Sie nun einzelne Werte aus dieser Datei benötigen, dann müssen Sie die Datei zeichenweise einlesen (eine Zeile enthält ja schließlich mehrere Werte). Nehmen wir an, Sie benötigen jeweils den dritten Wert aus der oben gezeigten Textdatei. Dieser Wert beginnt in jeder Zeile beim 26. Zeichen und ist nicht länger als 5 Zeichen. Sie müssen alle die Zeichen 26, 27, 28, 29 und 30 jeder Zeile lesen.

Mit der Methode *Read* ist genau dies möglich. Sie liest die als Parameter übergebene Anzahl von Zeichen ein. Die folgende Zeile liest zum Beispiel die nächsten 7 Zeichen aus der Textdatei und speichert diese in der Variable *strCharacters*:

```
strCharacters = objFile.Read(7)
```

Indem Sie die Methoden *Skip* und *SkipLine* verwenden, haben Sie die Möglichkeit Zeichen zu überspringen und so nur bestimmte Zeichen einzulesen. Script 4.41 liest zum Beispiel nur das sechste Zeichen jeder Zeile ein. Hierzu geht es folgendermaßen vor:

1. Es überspringt die ersten fünf Zeichen mit dem Befehl *Skip(5)*.
2. Es liest sechs Zeichen über den Befehl *Read(1)* ein.
3. Es springt zur nächsten Zeile.

Script 4.41: Einlesen von bestimmten Zeichen

```
1 Set objFSO = CreateObject("Scripting.FileSystemObject")
2 Set objFile = objFSO.OpenTextFile("C:\FSO\ScriptLog.txt", 1)
3 Do Until objFile.AtEndOfStream
4     objFile.Skip(5)
5     strCharacters = objFile.Read(1)
6     Wscript.Echo strCharacters
7     objFile.SkipLine
8 Loop
```

Um zu veranschaulichen wie das Script arbeitet, nehmen wir an, die Textdatei *C:\FSO\ScriptLog.txt* hätte den folgenden Inhalt:

```
XXXXX1XXXXXXXXXXXXXXXXXXXX
```

```
XXXXX2XXXXXXXXXXXXXXXXXXXX
```

XXXXX3XXXXXXXXXXXXXX

XXXXX4XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Die ersten fünf Zeichen jeder Zeile sind ein X, das sechste Zeichen ist eine Zahl und die restlichen Zeichen der Zeile sind wieder X. Wenn das Script ausgeführt wird, dann geht es folgendermaßen vor:

1. Es öffnet die Textdatei und beginnt in der ersten Zeile mit dem Lesen.
2. Es überspringt die ersten fünf Zeichen.
3. Es verwendet die Methode *Read*, um das sechste Zeichen zu lesen.
4. Es gibt das Zeichen aus.
5. Es springt zur nächsten Zeile und wiederholt den Prozess so lange bis alle Zeilen gelesen wurden.

Wenn Sie das Script unter CScript ausführen, erhalten Sie die folgende Ausgabe:

1
2
3
4

In Textdateien schreiben

Textdateien sind eine sehr gute Möglichkeit Daten dauerhaft zu speichern. Außerdem können Sie die von einem Script ausgeführten Aktionen in einer Textdatei protokollieren – dies ist gerade beim Erstellen von Scripten und bei der Fehlersuche sehr praktisch.

Um in eine Datei zu schreiben, muss ein Script folgendermaßen vorgehen:

1. Es erstellt eine Instanz von *FileSystemObject*.
2. Es verwendet die Methode *OpenTextFile*, um eine Datei zu öffnen. Hierbei gibt es zwei Möglichkeiten:
 - **ForWriting (2)** – In diesem Modus werden die vorhandenen Dateien der Textdatei mit den neuen Daten überschrieben.
 - **ForAppending (8)** – In diesem Modus werden die neuen Dateien an die bestehenden Daten der Datei angehängt.
3. Es schreibt über die Methode *Write*, *WriteLine* oder *WriteBlankLines* in die Datei.
4. Es schließt die Datei.

Die drei Methoden zum Schreiben in Textdateien sehen Sie in Tabelle 4.9.

Tabelle 4.9: Methoden zum Schreiben in Textdateien

Methode	Beschreibung
	Schreibt Daten in eine Textdatei, ohne einen Zeilenumbruch anzuhängen. Der folgende Code schreibt zum Beispiel zwei Strings in eine Textdatei:
Write	<pre>objFile.Write ("Die ist Zeile 1.") objFile.Write ("Die ist Zeile 2.")</pre> Die Textdatei sieht dann so aus:

Die ist Zeile 1.Die ist Zeile 2.

Schreibt Daten in eine Textdatei und hängt einen Zeilenumbruch an. Der folgende Code schreibt zum Beispiel zwei Strings in eine Textdatei:

```
objFile.WriteLine ("Die ist Zeile 1.")
```

WriteLine objFile.WriteLine ("Die ist Zeile 2.")

Das Ergebnis sieht so aus:

Die ist Zeile 1.

Die ist Zeile 2.

Schreibt eine leere Zeile in die Textdatei. Der folgende Code schreibt zwei Strings in die Textdatei und trennt diese durch einen Leerzeile:

```
objFile.WriteLine ("Dies ist Zeile 1.")
```

```
objFile.WriteLine (1)
```

WriteBlankLines objFile.WriteLine ("Dies ist Zeile 2.")

Das Ergebnis sieht so aus:

Dies ist Zeile 1.

Dies ist Zeile 2.

Außerdem können Sie beim Schreiben in eine Textdatei die VBScript-Konstante *VbTab* verwenden. Mit dieser Konstante fügen Sie an der entsprechenden Position ein Tabulatorzeichen ein. Die folgende Codezeile demonstriert dies:

```
objTextFile.WriteLine(objService.DisplayName & vbTab & objService.State)
```

Eine Schwachstelle von *FileSystemObject* ist, dass es nicht möglich ist, bestimmte Zeilen in einer Textdatei zu verändern. Sie können zum Beispiel kein Script schreiben, das einige Zeilen überspringt und dann eine Zeile mit einem bestimmten Text überschreibt. Stattdessen müssen Sie folgendermaßen vorgehen:

1. Sie lesen alle Zeilen der Datei ein (zum Beispiel in ein Array).
2. Sie ändern die entsprechenden Zeilen im Array.
3. Sie schreiben das Array wieder zurück in die Datei

Überschreiben vorhandener Daten

Stellen Sie sich vor, Sie möchten dass ein Script eine bestimmte nächtliche Aufgabe protokolliert, damit Sie das Protokoll jeweils am nächsten Tag überprüfen können. In diesem Fall reicht es Ihnen, wenn Ihnen das Protokoll vom Vortag zur Verfügung steht. Es ist nicht nötig, dass das Script die aktuellen Daten an die alten Dateien anhängt. Das Script kann die alten Daten vom Vortag einfach überschreiben.

Wenn Sie eine Datei im Modus *ForWriting* öffnen, dann werden die in der Datei vorhandenen Daten mit den neu geschriebenen Daten überschrieben. Nehmen wir zum Beispiel an, in einer einzelnen Textdatei haben Sie die letzte Kopie der gesammelten Werke von Shakespeare gespeichert. Nun öffnet Ihr Script diese Datei im Modus *ForWriting* und schreibt ein einzelnes Zeichen in die Datei. In diesem Fall sind alle Werke von Shakespeares für immer verloren, und die Datei enthält nur noch das eine neue Zeichen.

Script 4.42 öffnet die Datei *C:\FSO\ScriptLog.txt* im Modus *ForWriting* und schreibt das aktuelle Datum und die Uhrzeit in die Datei. Jedes Mal wenn Sie das Script ausführen, wird der gesamte Inhalt der Datei überschrieben. Die Datei enthält niemals mehr als ein Datum und eine Uhrzeit – egal wie oft es auch ausgeführt wird.

Script 4.42: Überschreiben von vorhandenen Daten

```
1 Const ForWriting = 2
2 Set objFSO = CreateObject("Scripting.FileSystemObject")
3 Set objFile = objFSO.OpenTextFile("C:\FSO\ScriptLog.txt", ForWriting)
4 objFile.Write Now
5 objFile.Close
```

Neue Daten an bestehende Daten anhängen

Stellen Sie sich vor, Sie können die Protokolldateien aus dem Beispiel oben nur einmal in der Woche überprüfen. In diesem Fall möchten Sie sicher nicht, dass das Protokoll vom Vortag mit dem aktuellen Protokoll überschrieben wird. Stattdessen soll das Script die aktuellen Daten an die alten Daten anhängen.

Dies können Sie durchführen, indem Sie die Textdatei im Modus *ForAppending* öffnen. Statt die Daten zu überschreiben, werden die neuen Daten am Ende der Datei angehängt. Script 4.43 schreibt zum Beispiel das aktuelle Datum und die Uhrzeit in eine Textdatei. Da die Datei aber im Modus *ForAppending* geöffnet wird, werden diese neuen Daten an die alten angehängt. Bei jeder Ausführung kommt so eine Zeile zur Textdatei hinzu. Die Textdatei sieht nach einigen Ausführungen des Scripts zum Beispiel so aus:

```
6/25/2002 8:49:47 AM
6/25/2002 8:49:48 AM
6/25/2002 8:50:33 AM
6/25/2002 8:50:35 AM
```

Script 4.43: Daten an eine Textdatei anhängen

```
1 Const ForAppending = 8
2 Set objFSO = CreateObject("Scripting.FileSystemObject")
3 Set objFile = objFSO.OpenTextFile("C:\FSO\ScriptLog.txt", ForAppending)
4 objFile.WriteLine Now
5 objFile.Close
```

Da das Script die Methode *WriteLine* verwendet, werden Datum und Uhrzeit immer in eine neue Zeile geschrieben. Wäre die Methode *Write* verwendet worden, dann sähe die Textdatei so aus:

```
6/25/2002 8:49:47 AM6/25/2002 8:49:48 AM6/25/2002 8:50:33 AM6/25/2002 8:50:35 AM
```

Das Dictionary-Objekt

Oftmals erhält ein Script Informationen von einer externen Quelle – zum Beispiel aus einer Textdatei oder einer Datenbank. Diese Informationen müssen irgendwo gespeichert werden. Natürlich haben Sie die Möglichkeit solche Informationen in einzelnen Variablen oder in einem Array zu speichern – sehr effektiv ist dies jedoch nicht. Die Alternative wäre ein *Dictionary-Objekt* (ein Verzeichnis-Objekt).

Ein *Dictionary-Objekt* funktioniert wie ein assoziativer Array. Das heißt, es speichert Wert und Schlüssel in Paaren. Ein Beispiel soll dies verdeutlichen. Ein Array speichert Werte so (die Zahl stellt den Arrayindex dar):

0 — München
1 — Hannover
2 — Wiesbaden

Ein Dictionary-Objekt könnte die Werte zum Beispiel so speichern:

Bayern — München
Niedersachsen — Hannover
Hessen — Wiesbaden

Alternativ können Sie sich ein Dictionary-Objekt auch wie ein Telefonbuch vorstellen. Es speichert Namen und die dazugehörigen Telefonnummern. Über die Namen können Sie dann die Telefonnummern abrufen.

Stellen Sie sich vor, das Script verwendet mehrere Argumente. Sie können diese Argumente zwar in einem Array speichern, das Dictionary-Objekt bietet jedoch mehrere Vorteile gegenüber einem Array:

- Sie müssen nicht festlegen, wie viele Objekte gespeichert werden sollen. Bei einem Array müssen Sie das Array im Voraus erstellen (oder später vergrößern).
- Sie müssen keine Indexnummer zum Zugriff auf ein Element verwenden. Stattdessen können Sie über einen Schlüssel(begriff) auf das Element zugreifen (im Beispiel oben wären das die Bundesländer – sprich: "Gib mir die Landeshauptstadt von Hessen zurück" statt „Gib mir das Element mit der Nummer 2“).

Damit ist das Dictionary-Objekt ideal um Informationen zu speichern und diese später wieder abzufragen.

Ein Dictionary-Objekt erstellen

Da ein Dictionary-Objekt ein COM-Objekt ist, müssen Sie wie bei allen anderen COM-Objekten auch erst eine Instanz des Objekts erstellen. Die folgende Codezeile führt dies durch:

```
Set objDictionary = CreateObject("Scripting.Dictionary")
```

Nachdem Sie das Dictionary-Objekt erstellt haben, können Sie dessen Eigenschaften konfigurieren und Elemente zum Dictionary-Objekt hinzufügen.

Das Dictionary-Objekt hat nur eine konfigurierbare Eigenschaft: *CompareMode*. Diese Eigenschaft beeinflusst, wie Sie die Einträge im Dictionary-Objekt finden können. Standardmäßig arbeitet das Dictionary-Objekt im Binärmodus. Das bedeutet, dass die einzelnen Schlüssel als ASCII-Werte gespeichert werden. Bei ASCII-Werten wird zwischen Großbuchstaben und Kleinbuchstaben unterschieden. Im Binärmodus könnten Sie zum Beispiel die beiden folgenden Werte als Schlüssel anlegen:

alerter

ALERTER

Mit anderen Worten: Im Binärmodus ist es durchaus möglich versehentlich mehrere Einträge für dasselbe Element anzulegen. Außerdem ist die Suche in diesem Modus schwieriger. Wenn Sie zum Beispiel nach dem Schlüssel *Alerter* suchen, dann erhalten Sie kein Ergebnis. Das liegt daran, weil keiner der beiden Einträge dieser Buchstabenkombination entspricht (großes A, Rest klein).

Im zweiten Modus, dem Textmodus, werden Groß- und Kleinbuchstaben gleich behandelt. In diesem Modus können Sie keinen Schlüssel mit dem Namen *ALERTER* anlegen, wenn bereits ein Schlüssel mit dem Namen *alerter* vorhanden ist. Auch das Suchen ist viel einfacher. Wenn Sie nach dem Schlüssel *alerter* suchen, erhalten Sie die Einträge für *Alerter* und *ALERTER* zurück.

Um den Modus eines Dictionary-Objekts zu konfigurieren, erstellen Sie zuerst eine Instanz des Objekts. Dann setzen Sie die Eigenschaft *CompareMode* auf einen der beiden folgenden Werte:

0 — Setzt den Modus auf *Binär*. Dies ist auch der Standardwert.

1 — Setzt den Modus auf *Text*.

Script 4.44 setzt das Dictionary-Objekt beispielsweise auf den Modus *Text*.

Script 4.44: Dictionary-Objekt konfigurieren

```
1 Const TextMode = 1
2 Set objDictionary = CreateObject("Scripting.Dictionary")
3 objDictionary.CompareMode = TextMode
```

Wenn das Dictionary-Objekt bereits Elemente enthält, können Sie die Eigenschaft *CompareMode* nicht mehr verändern. Das liegt daran, weil es sonst zu Inkonsistenzen im Dictionary-Objekt kommen könnte (zum Beispiel wenn Einträge vorhanden sind, die mit dem neuen Modus nicht kompatibel sind). Die folgenden Schlüssel könnten im Modus *Binär* zum Beispiel im gleichen Dictionary-Objekt gespeichert werden:

apple

Apple

APPLE

Im *Textmodus* wären die drei Schlüssel jedoch identisch, da in diesem Modus ja nicht zwischen Groß- und Kleinschreibung unterschieden wird. Sie müssen also erst alle Elemente aus dem Objekt löschen, bevor Sie dessen Modus ändern können.

Einträge zu einem Dictionary-Objekt hinzufügen

Mit der Methode *Add* können Sie neue Einträge zu einem Dictionary-Objekt hinzufügen. Die Methode benötigt zwei Parameter: den *Schlüsselnamen*, über den später auf den Eintrag zugegriffen werden soll und den *Wert* für diesen Eintrag.

Script 4.45 erstellt ein Dictionary-Objekt und fügt diesem dann die Einträge aus Tabelle 4.10 hinzu.

Tabelle 4.10: Beispieleinträge

Schlüssel

Wert

Drucker1	Druckt
Drucker2	Offline
Drucker3	Druckt

Script 4.45: Einen Eintrag zum Dictionary-Objekt hinzufügen

```

1 Set objDictionary = CreateObject("Scripting.Dictionary")
2 objDictionary.Add "Drucker 1", "Druckt"
3 objDictionary.Add "Drucker 2", "Offline"
4 objDictionary.Add "Drucker 3", "Druckt"

```

Die Schlüssel der einzelnen Einträge müssen jeweils eindeutig sein – sie dürfen nicht mehrmals vorkommen. Der folgenden Scriptcode würde daher zum Beispiel zu einem Fehler führen:

```

objDictionary.Add "Drucker 1", "Druckt"
objDictionary.Add "Drucker 1", "Offline"

```

Versehentlich einen Schlüssel zu einem Dictionary-Objekt hinzufügen

Ein potentiell Problem bei Dictionary-Objekten ist es, wenn Sie versuchen auf ein Element zuzugreifen, das noch nicht vorhanden ist. Statt den Wert für dieses Objekt zurückzugeben, wird in diesem Fall nämlich ein neuer Eintrag mit diesem Schlüssel angelegt. Das folgende Script versucht zum Beispiel auf das nicht vorhandene Element mit dem Schlüssel *Printer 4* zuzugreifen:

```

Set objDictionary = CreateObject("Scripting.Dictionary")
objDictionary.Add "Drucker 1", "Druckt"
objDictionary.Add "Drucker 2", "Offline"
objDictionary.Add "Drucker 3", "Druckt"
Wscript.Echo objDictionary.Item("Drucker 4")

```

Wenn das Script versucht auf das nicht vorhandene Element (*Drucker 4*) zuzugreifen, wird leider kein Laufzeitfehler ausgelöst. Stattdessen wird ein neues Element mit dem Schlüssel *Drucker 4* zum Dictionary-Objekt hinzugefügt. Der Wert dieses Elements ist *Null*. Da der Wert des neuen Eintrags auch gleich angezeigt wird, sieht die Ausgabe des Scripts wie in Abbildung 4.9 aus.

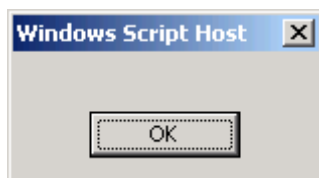


Abbildung 4.9: Ausgabe eines versehentlich hinzugefügten Elements zu einem Dictionary-Objekt

Um dieses Problem zu umgehen, müssen Sie vor einem Zugriff auf ein Element prüfen, ob es das Element gibt.

Bearbeiten von Schlüsseln und Werten in einem Dictionary-Objekt

Zu den Aufgaben, die Sie mit einem Dictionary-Objekt durchführen können, gehören unter anderem die folgenden:

- Prüfen, wie viele Einträge im Dictionary-Objekt vorhanden sind.
- Die Einträge im Dictionary-Objekt auflisten.
- Prüfen, ob ein bestimmter Eintrag vorhanden ist oder nicht.
- Einen Wert oder einen Schlüssel ändern.
- Einträge aus dem Dictionary-Objekt entfernen.

Diese Aktionen werden in den folgenden Abschnitt besprochen.

Die Zahl der Einträge in einem Dictionary-Objekt abfragen

Wie die meisten anderen Collections hat auch das Dictionary-Objekt eine Eigenschaft mit dem Namen *Count*. Diese gibt die Zahl der Elemente der Collection zurück. Script 4.46 erstellt eine Instanz des Dictionary-Objekts, fügt diesem drei Einträge hinzu und gibt dann die Anzahl der Einträge über die Eigenschaft *Count* zurück.

Script 4.46: Prüfen, wie viele Einträge in einem Dictionary-Objekt vorhanden sind

```
1 Set objDictionary = CreateObject("Scripting.Dictionary")
2 objDictionary.Add "Drucker 1", "Printing"
3 objDictionary.Add "Drucker 2", "Offline"
4 objDictionary.Add "Drucker 3", "Printing"
5 Wscript.Echo objDictionary.Count
```

Wenn Sie das Script ausführen, erhalten Sie den Wert 3 zurück (die Anzahl der Einträge).

Die Elemente eines Dictionary-Objekts aufzählen

Über die Methoden *Keys* und *Items* können Sie die Schlüssel oder Werte aller Einträge eines Dictionary-Objekts in einem Array zurückgeben. Nachdem Sie eine der Methoden aufgerufen haben, können Sie das Array dann mit einer For-Each-Schleife durchgehen.

Script 4.47 erstellt ein einfaches Dictionary-Objekt mit drei Einträgen. Dann verwendet es die Methode *Keys*, um alle Schlüssel des Dictionary-Objekts im Array *colKeys* zu speichern und die Elemente des Arrays in einer For-Each-Schleife auszugeben. Danach wird für die Werte des Dictionary-Objekts derselbe Vorgang über die Methode *Items* ausgeführt.

Script 4.47: Auflisten der Schlüssel und Werte eines Dictionary-Objekts

```
1 Set objDictionary = CreateObject("Scripting.Dictionary")
2 objDictionary.Add "Drucker 1", "Druckt"
3 objDictionary.Add "Drucker 2", "Offline"
4 objDictionary.Add "Drucker 3", "Druckt"
5
6 colKeys = objDictionary.Keys
```

```

7 For Each strKey in colKeys
8     Wscript.Echo strKey
9 Next

1(
1:colItems = objDictionary.Items
1:For Each strItem in colItems
1:    Wscript.Echo strItem
1:Next

```

Wenn Sie das Script unter *CScript* ausführen, erhalten Sie die folgende Ausgabe:

```

Drucker 1
Drucker 2
Drucker 3

Druckt
Offline
Druckt

```

Um den Wert eines bestimmten Eintrags anzuzeigen, verwenden Sie die Methode *Item*. Die folgende Zeile zeigt zum Beispiel den Wert des Eintrags mit dem Schlüssel *Drucker 3* an:

```
Wscript.Echo objDictionary.Item("Printer 3")
```

Die Existenz eines Schlüssels prüfen

Das Dictionary-Objekt bietet Ihnen gegenüber einem Array oder einer Collection einen großen Vorteil: Sie können sehr einfach feststellen, ob ein bestimmter Schlüssel vorhanden ist. Nehmen wir zum Beispiel einmal an, dass Sie in einer Liste mit Dateien nach bestimmten DLLs suchen möchten. Mit einer Collection oder einem Array müssen Sie die gesamte Liste in einer Schleife durchlaufen und jedes Element einzeln prüfen.

Im Gegensatz dazu können Sie bei einem Dictionary-Objekt die Methode *Exists* verwenden – mit dieser stellen Sie fest, ob ein bestimmter Schlüssel vorhanden ist. Die Methode erwartet einen Parameter (den Namen des Schlüssels). Wenn der Schlüssel existiert, dann gibt sie den Boolean-Wert *True* zurück – andernfalls ist der Rückgabewert *False*.

Script 4.48 erstellt zum Beispiel ein Dictionary-Objekt und fügt diesem drei Elemente hinzu (Drucker 1, Drucker 2 und Drucker 3). Dann prüft das Script, ob ein Schlüssel mit dem Namen *Drucker 4* vorhanden ist und gibt das Ergebnis dieser Prüfung aus.

Script 4.48: Ein Dictionary-Objekt auf einen bestimmten Schlüssel prüfen

```

1 Set objDictionary = CreateObject("Scripting.Dictionary")
2 objDictionary.Add "Drucker 1", "Druckt"

```

```

3 objDictionary.Add "Drucker 2", "Offline"
4 objDictionary.Add "Drucker 3", "Druckt"
5 If objDictionary.Exists("Drucker 4") Then
6     Wscript.Echo "Drucker 4 ist vorhanden."
7 Else
8     Wscript.Echo "Drucker 4 ist nicht vorhanden."
9 End If

```

Ein Element in einem Dictionary-Objekt ändern

Die Elemente eines Dictionary-Objekts sind nicht in Stein gemeißelt. Sie haben die Möglichkeit die Elemente zu ändern. Script 4.49 erstellt ein Dictionary-Objekt mit drei Schlüsseln: *atl-dc-01*, *atl-dc-02* und *atl-dc-03*. Der Wert jedes Elements wird auf den Text "No status." gesetzt.

Danach werden die Werte der Elemente über die Methode *Item* geändert. Die Methode erwartet als Parameter den Schlüssel des zu ändernden Elements.

Script 4.49: Ändern des Werts eines Elements in einem Dictionary-Objekt

```

1 Set objDictionary = CreateObject("Scripting.Dictionary")
2 objDictionary.Add "atl-dc-01", "Kein Status"
3 objDictionary.Add "atl-dc-02", "Kein Status"
4 objDictionary.Add "atl-dc-03", "Kein Status"
5
6 colKeys = objDictionary.Keys
7 For Each strKey in colKeys
8     Wscript.Echo strKey, objDictionary.Item(strKey)
9 Next
10
11 objDictionary.Item("atl-dc-01") = "Verfügbar"
12 objDictionary.Item("atl-dc-02") = "Verfügbar"
13 objDictionary.Item("atl-dc-03") = "Nicht verfügbar"
14
15 colKeys = objDictionary.Keys
16 For Each strKey in colKeys

```

```
17     Wscript.Echo strKey, objDictionary.Item(strKey)
18 Next
```

Wenn Sie das Script unter *CScript* ausführen, erhalten Sie die folgende Ausgabe:

```
atl-dc-01 Kein Status
atl-dc-02 Kein Status
atl-dc-03 Kein Status
atl-dc-01 Verfügbar
atl-dc-02 Verfügbar
atl-dc-03 Nicht verfügbar
```

Elemente aus einem Dictionary-Objekt entfernen

Es gibt zwei Methoden, um Elemente aus einem Dictionary-Objekt zu entfernen:

- **RemoveAll** – entfernt alle Elemente.
- **Remove** – entfernt ein bestimmtes Element.

Alle Elemente aus einem Dictionary-Objekt entfernen

Um alle Elemente aus einem Dictionary-Objekt zu entfernen, können Sie die Methode *RemoveAll* verwenden. Script 4.50 demonstriert dies bei einem Dictionary-Objekt mit drei Elementen. Nachdem es alle Elemente angezeigt hat, entfernt es die Elemente über den folgenden Befehl:

```
objDictionary.RemoveAll
```

Um zu überprüfen ob alle Elemente entfernt wurden, gibt es noch einmal alle Elemente aus.

Script 4.50: Alle Elemente aus einem Dictionary-Objekt entfernen

```
1 Set objDictionary = CreateObject("Scripting.Dictionary")
2 objDictionary.Add "Drucker 1", "Druckt"
3 objDictionary.Add "Drucker 2", "Offline"
4 objDictionary.Add "Drucker 3", "Druckt"
5 colKeys = objDictionary.Keys
6 Wscript.Echo "Erster Durchlauf: "
7 For Each strKey in colKeys
8     Wscript.Echo strKey
9 Next
10 objDictionary.RemoveAll
```

```
11 colKeys = objDictionary.Keys
12 Wscript.Echo VbCrLf & "Zweiter Durchlauf: "
13 For Each strKey in colKeys
14     Wscript.Echo strKey
15 Next
```

Wenn Sie das Script unter *CScript* ausführen, erhalten Sie die folgende Ausgabe:

Erster Durchlauf:

```
Drucker 1
Drucker 2
Drucker 3
```

Zweiter Durchlauf:

Bestimmte Einträge aus einem Dictionary-Objekt entfernen

Stellen Sie sich vor, Sie haben ein Dictionary-Objekt mit den folgenden Schlüsseln:

```
atl-dc-01
atl-dc-02
atl-dc-03
atl-dc-04
atl-dc-05
```

Sie verwenden das Dictionary-Objekt, um auf die Computer zuzugreifen. Beim Zugriff auf zwei Computer (*atl-dc-03* und *atl-dc-04*) stellt das Script fest, dass diese nicht erreichbar sind. Das Script muss also später erneut versuchen diese Computer zu erreichen. Wie können Sie diese Aufgabe bewältigen? Ein Weg wäre alle erreichten Computer aus dem Dictionary-Objekt zu entfernen. Nach dem ersten Durchlauf wären also noch folgende Elemente vorhanden:

```
atl-dc-03
atl-dc-04
```

Bei nächsten Durchlauf werden wieder alle erfolgreich kontaktierten Computer entfernt, usw. Wenn das Dictionary-Objekt keine Einträge mehr enthält, dann wurden alle Computer erfolgreich kontaktiert.

Um ein Element zu entfernen, verwenden Sie die Methode *Remove*. Ihr übergeben Sie als einzigen Parameter den Schlüssel des zu entfernenden Eintrags:

```
objDictionary.Remove("atl-dc-02")
```

Script 4.51 erstellt ein Dictionary-Objekt mit drei Elementen und gibt dann alle Schlüssel aus. Danach entfernt es den Eintrag mit dem Schlüssel *Drucker 2* und gibt die Elemente erneut aus.

Script 4.51: Einen bestimmten Eintrag entfernen

```
1 Set objDictionary = CreateObject("Scripting.Dictionary")
2 objDictionary.Add "Drucker 1", "Druckt"
3 objDictionary.Add "Drucker 2", "Offline"
4 objDictionary.Add "Drucker 3", "Druckt"
5 colKeys = objDictionary.Keys
6 Wscript.Echo "Erster Durchlauf: "
7 For Each strKey in colKeys
8     Wscript.Echo strKey
9 Next
10 objDictionary.Remove("Drucker 2")
11 colKeys = objDictionary.Keys
12 Wscript.Echo VbCrLf & "Zweiter Durchlauf: "
13 For Each strKey in colKeys
14     Wscript.Echo strKey
15 Next
```

Wenn Sie das Script unter *CScript* ausführen, erhalten Sie die folgende Ausgabe:

Erster Durchlauf:

Drucker 1

Drucker 2

Drucker 3

Zweiter Durchlauf:

Drucker 1

Drucker 2