Windows Scripting

- Für alle Windows-Versionen (inkl. XP, Server 2003 R2 und Vista)
- Visual Basic, Visual Basic Script und PowerShell Language
- Über 900 Praxisbeispiele

ADDISON-WESLEY

CD

5 Einführung

5.1 Der Automatisierungsbedarf

GUIs

Microsoft spielt auf dem Weltmarkt der Betriebssysteme eine zentrale Rolle, sowohl im Unternehmenseinsatz als auch im Heimbereich, sowohl auf dem Client als auch dem Server. Ein Grund für den Siegeszug von Windows sind die grafischen Benutzeroberflächen (engl. Graphical User Interfaces, kurz GUIs), die eine sehr einfache Bedienung ermöglichen. Fast alle administrativen Funktionen des Windows-Betriebssystems lassen sich durch ein GUI verwalten. Die Windows-GUIs zeichnen sich durch eine hohe Konsistenz aus, d.h., sie sind nach dem gleichen Prinzip aufgebaut: Sie haben alle ähnliche Menüs, Symbolleisten und Dialogfenster.

Die Installation und Konfiguration eines Windows-Systems ist daher vergleichsweise einfach und intuitiv. Sofern grundsätzliche Erfahrung in der Administration eines Windows-Systems vorhanden ist, fällt die Einarbeitung in neue Aufgaben leicht. Auch Personen, die nur selten administrative Aufgaben ausführen müssen, können diese schnell erledigen, ohne komplexe Befehle beherrschen zu müssen.

Automatisierungsbedarf

Ein gutes GUI ist aber nur ein Aspekt der Systemadministration. Auf der anderen Seite der Medaille stehen Aufgaben, die sich nicht oder nur schlecht durch ein GUI lösen lassen:

- Zum einen sind dies Aufgaben, die unbeaufsichtigt, d.h. ohne Beisein eines Menschen, von der Maschine automatisch ausgeführt werden sollen (z.B. Überwachungsaufgaben, Systemstart-Scripts sowie rechenintensive Prozesse, die nur nachts durchgeführt werden können).
- Zweitens sind dies wiederkehrende Administrationsaufgaben, die in definierten Intervallen ausgeführt werden sollen (z.B. Backup, Datenabgleich, Login-Scripts).
- In die dritte Gruppe gehören Administrationsaufgaben, die zu einem bestimmten Zeitpunkt in großer Menge anfallen (z.B. Benutzereinrichtung bei einer Systemumstellung).
- Viertens will ein Administrator auch bestimmte Aufgaben an andere Personen delegieren. Diesen Personen möchte er eine Routine zur Verfügung stellen, die abseits der vielfältigen Möglichkeiten eines GUI in einem fest vorgeschriebenen Dialogpfad eine bestimmte isolierte Aufgabe erledigt.
- Schließlich wird es immer wichtiger, Anpassbarkeit und Erweiterbarkeit in Betriebssysteme und Anwendungen zu integrieren, da es immer komplexere und individuellere Wünsche der Anwender und Administratoren gibt, die man als Softwareanbieter nicht alle in die Softwareprodukte integrieren kann.

CLI Derartige Aufgaben wollen Systemadministratoren üblicherweise durch ein Command Line Interface (CLI) ausführen bzw. dort automatisiert ablaufen lassen. Während unter Linux das GUI mehr als ein Add-on für das CLI zu verstehen ist, stand bei Windows immer das GUI im Mittelpunkt. Es existieren verschiedene Kommandozeilenwerkzeuge, die im Folgenden kurz besprochen werden.

5.2 Was ist Scripting?

Scripting und Scriptsprachen Scripting ist das Schreiben eines Programms mit Hilfe einer Scriptsprache; das Programm wird in diesem Zusammenhang Script genannt. Diese Definition führt zu der Frage, was eine Scriptsprache ist. Die Antwort darauf ist jedoch nicht einfach. [FIS99] nennt folgende Kriterien zur Unterscheidung einer Scriptsprache von anderen Sprachen:

- Die Sprache wird interpretiert (keine Kompilierung notwendig).
- Scripts werden normalerweise im Quellcode gespeichert und weitergegeben.
- Scriptsprachen haben in der Regel einen geringen Sprachumfang. Viele typische Funktionalitäten sind in externe Bibliotheken und Komponenten ausgelagert.
- Die Syntax ist einfach zu verwenden und zu erlernen.
- Es gibt nur ein sehr schwaches Typsystem.
- Die Abstraktion von technischen Details wie z.B. Zeigern und Speicherverwaltung ist hoch.
- Eine Scriptsprache ist der Maschinensprache und der Computer-Hardware ferner als eine normale Programmiersprache und kann daher einen Computer nicht so leicht zum Absturz bringen.
- Die Sprache dient dem Ad-hoc-Gebrauch.

Beispiele für Scriptsprachen sind REXX, Perl, Python, AppleScript, PHP, JavaScript/JScript und VBScript. Gemäß obiger Definition sind auch die Unix-Shellsprachen wie *sh* und *csh* als Scriptsprachen zu betrachten.

ALP Zur Betonung der Anwendungsnähe und Abstraktion von technischen Details wird Scripting häufig als Application Level Programming (ALP) bezeichnet

Glue Code für Komponenten Scriptsprachen kommt in Zusammenhang mit komponentenorientierter Softwareentwicklung oft die Rolle zu, als Verbindung (so genannter *Glue Code*) zwischen Komponenten zu fungieren. Diese Rolle nehmen die Scriptsprachen auch beim Active Scripting ein.

Oftmals sprechen Fachleute auch von *Scripting*, wenn keine Scriptsprache im engeren (oben definierten) Sinne eingesetzt wird. Dann wird Scripting mit Automatisierung gleichgesetzt, selbst wenn zur Implementierung der Automatisierungslösung eine Sprache eingesetzt wird, die keine Scriptsprache ist.

Automatisierbarkeit Eine Anwendung wird *automatisierbar* (synonym: *fernsteuerbar*, *programmierbar* oder *scriptable*) genannt, wenn es möglich ist, die Anwendung durch Programmcode zu steuern.

Scriptsprache versus Batchsprache versus Makrosprache

Die Begriffe Scriptsprache und Batchsprache werden leider nicht einheitlich verwendet und oft vermischt. In diesem Buch wird die Grenze zwischen Scripting und Batch hinsichtlich der Ausdrucksfähigkeit der Programmiersprache bezogen.

Eine Scriptsprache ist eine vollständige Programmiersprache, die verschiedene Vereinfachungen gegenüber „echten" Programmiersprachen für die Softwareentwicklung besitzt.

Eine Batchsprache hingegen ist eine Aneinanderreihung von eigenständigen Befehlen. Dabei fehlen oft typische Programmierkonstrukte wie Bedingungen, Schleifen und Variablen.

Makrosprachen sind Scriptsprachen, die innerhalb von Anwendungen zur Automatisierung der jeweiligen Anwendung dienen.

5.3 Automatisierungslösungen auf der Windows-Plattform

Das lange vernachlässigte Thema der kommandozeilenbasierten Systemadministration hat Microsoft Mitte der 90er Jahre stärker in den Fokus genommen. Microsoft bietet folgende Lösungsansätze für die automatisierte Systemadministration durch Scripting an (Stand September 2006):

- DOS-Batchsprache (nur im erweiterten Sinne als Scriptsprache zu bezeichnen, vgl. Erläuterung im vorherigen Kapitel)
- Active Scripting-Architektur, insbesondere der Windows Script Host (WSH)
- Scripting mit .NET Framework-basierten Programmiersprachen wie C# und Visual Basic .NET
- Microsoft Powershell

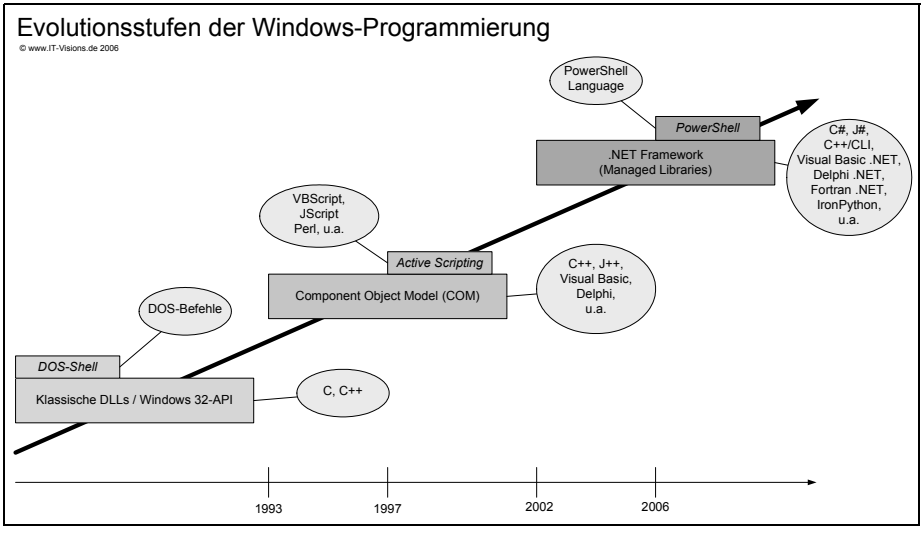


Bild 5.1: Entwicklungsstufen der Programmierschnittstellen in Windows

Da Microsoft die Schwächen der Automatisierung mit Hilfe der DOS-Batchsprache lange Zeit nicht erkannt hat, haben sich eine Reihe anderer Ansätze entwickelt. Zu nennen sind hier insbesondere:

- KiXtart [KIX04]
- Perl for Win32 [ACT00]
- REXX [IBM01]
- AutoIt [AUT04]
- WinBatch [WINB04]
- WinRobots [WINR04]
- GnuWin32 (Portierung von Unix-Werkzeugen auf Windows) [GNUW04] und
- Bash-Shell im Rahmen von Cygwin, einer Linux-ähnlichen Umgebung für Windows [CYG04].

In diesem Buch werden Sie das von Microsoft entwickelte Active Scripting, das .NET Framework und die Microsoft Shell (MSH) als Alternative kennen lernen.

5.4 Die DOS-Batchsprache

DOS-Batch

Die Microsoft DOS-Batchsprache, die in Einzelbefehlen im Befehlszeilenfenster oder in Form von *.bat*-Dateien ausgeführt werden kann, gibt es seit den Anfängen von MS-DOS. Sie war damals – zu einer Zeit, als es noch keine grafische Benutzeroberfläche namens Windows gab – ein adäquates Instrument zur Systemadministration. Sie ist auch heute noch in allen Windows-Versionen integriert, in Form des Kommandozeilenfensters (alias DOS-Fenster oder DOS Command Prompt).

Die DOS-Batchsprache hat in all diesen Jahren allerdings nur wenige Veränderungen erfahren, obwohl die Anforderungen stets gestiegen sind. Windows war daher in diesem Bereich gegenüber Unix lange Zeit im Nachteil. In Unix gibt es mächtige Shells (z.B. bash, Bourne-Shell, Korn-Shell, C-Shell, tcsh), in denen so genannte Shell-Scripts ausgeführt werden können, mit denen sich alle administrativen Aufgaben durchführen lassen.

Schwachpunkte

Die wesentlichen Schwachpunkte der DOS-Batchprogrammierung sind:

- Die DOS-Batchsprache ist keine vollständige Programmiersprache. Daher sind viele Aufgaben nur umständlich zu bewältigen.
- Die Kommunikation zwischen Befehlen ist textorientiert.
- Ein- und Ausgabe sind zeilenorientiert.
- Es gibt keine Möglichkeit, Programmierschnittstellen (Application Programming Interfaces – APIs) des Betriebssystems oder von Anwendungen (weder komponentenbasierte noch nichtkomponentenbasierte) anzusprechen.
- Die Sprache ist nur über neue *.cmd*- oder *.exe*-Dateien erweiterbar.
- Die DOS-Befehle decken zwar die Anforderungen der DOS-Ebene ab, auf viele GUI-Funktionen des Windows-Betriebssystems gibt es jedoch keinen Zugriff.

Die mangelnden Fähigkeiten der DOS-Batchsprache hat Microsoft immer wieder durch neue Kommandozeilenwerkzeuge auszugleichen versucht. Insbesondere ist hier die Werk-

zeugsammlung zu nennen, die Microsoft unter dem Namen Support Tool oder Resource Kit zu den verschiedenen Betriebssystemen ausgeliefert hat.

Microsoft macht kein Geheimnis daraus, dass man im Zuge der Entwicklung des Windows Server 2003 einigen Unix-Administratoren über die Schulter geschaut hat, um Ideen für die Verbesserung der befehlsbasierten Administration zu sammeln. Die Erkenntnisse aus dieser Studie manifestieren sich in einer Vielzahl neuer Kommandozeilenwerkzeuge im Windows Server 2003. Die langfristige Erkenntnis dieser Studie zeigt sich jedoch in der neuen Microsoft Shell (siehe Kapitel 10).

Eine gute Darstellung der Anwendung der DOS-Sprache ist das nachfolgend empfohlene Buch.

Buchtipp

Armin Hanisch:

Windows 2003 Shell Scripting

Abläufe automatisieren ohne Programmierkenntnisse

Addison-Wesley 2006, ISBN 978-3-8273-2413-9

5.5 Die Active Scripting-Architektur

ActiveX Scripting

Interessanterweise hat sich Microsoft des Themas Windows Scripting erst im Zuge der Besinnung auf das Internet und der dortigen Popularität von Scriptsprachen angenommen. Seit Mitte der 90er Jahre stellen die Redmonder eine eigene modulare Scripting-Architektur für Internetanwendungen und Windows bereit.

Die Windows Scripting-Architektur heißt bei Microsoft auch *ActiveX Scripting, Active Scripting* oder *Windows Script.* In diesem Buch wird vorzugsweise der Begriff Active Scripting verwendet. Grundlage der gesamten Architektur ist Microsofts Komponentenarchitektur – das *Component Object Model (COM)*.

Das *Component Object Model* ist Microsofts Technologie für die Entwicklung und Nutzung von objektorientierten Softwarekomponenten (zum Begriff Softwarekomponente siehe Anhang A), die *COM-Komponenten* genannt werden. ActiveX ist ein Marketingbegriff für einen Teil dieser Komponentenarchitektur. COM wird ausführlich in Kapitel 7 vorgestellt.

COM ist objektbasiert. Daher dreht sich auch beim Active Scripting alles um Objekte und Klassen.

Die Active Scripting-Architektur besteht aus folgenden drei Bausteinen:

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

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

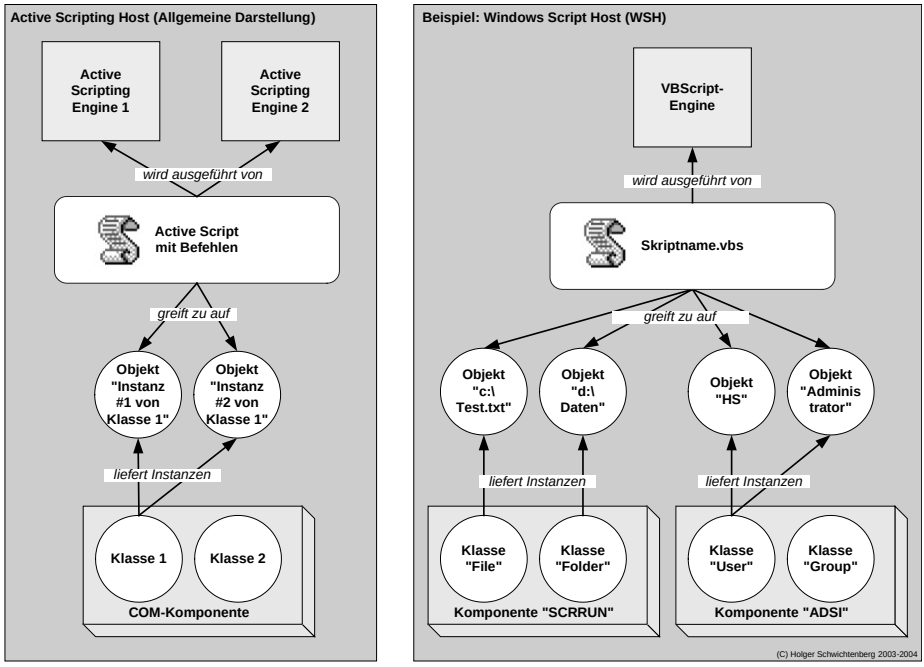


Bild 5.2: Die Active Scripting-Architektur von Microsoft

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

COM versus ActiveX Auch Active Scripting Hosts und Active Scripting Engines sind COM-Komponenten, die spezielle Schnittstellen implementieren. Dass die Architektur *ActiveX Scripting* und nicht *COM Scripting* heißt, beruht darauf, dass Microsoft aus Marketinggründen den Begriff ActiveX gefördert hat. ActiveX wurde eine Zeit lang mit dem Begriff COM sogar völlig gleichgesetzt. In Kapitel 7 erhalten Sie eine detaillierte Einführung in COM.

Host versus Sprache Die Trennung in Host und Sprache ist in vielen Scriptsprachen nicht gegeben. So steht der Begriff *Personal Home Page Tools* (PHP) sowohl für eine Sprache als auch für einen Scripting Host. Gleiches gilt für viele Unix-Shells.

Vielfältige Möglichkeiten

Die Möglichkeiten der Windows-Automatisierung mit der Active Scripting-Architektur sind vielfältig, deshalb kommt es besonders auf die Auswahl von Scriptsprache und Komponenten an, wenn es um die Frage geht, wie aufwändig ein Automatisierungsprojekt wird. Eine intensive Recherche nach vorhandenen COM-Komponenten ist ein entscheidender Erfolgsfaktor.

Mit der zunehmenden Verbreitung des Komponentengedankens werden Make-or-Buy-Entscheidungen zu einem ständigen Begleiter im Softwareentwicklungsprozess. Entwickler werden sich fragen lassen müssen, ob es notwendig war, eigene Routinen zu entwickeln, anstatt auf dem Markt verfügbare Komponenten zu einer Anwendung zusammenzubauen. Gerade der Active Scripting-Bereich zeigt, dass eine unterlassene Internetrecherche dazu führt, dass man sich tagelang mit Problemen beschäftigt, die andere mit wesentlich geringeren Kosten längst gelöst haben. In Kapitel 12 erhalten Sie einen Leitfaden für die Recherche nach Komponenten für das Scripting.

5.5.1 Active Scripting Hosts

Ablaufumgebung

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

Verfügbare Scripting Hosts

Microsoft integriert Scripting Hosts in immer mehr Produkte, insbesondere in die Produkte des „Windows Server System“ (früher: „.NET Enterprise Server“ und davor „BackOffice“). Aktuell sind folgende Scripting Hosts verfügbar:

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

Script Control

Scripting Hosts werden inzwischen auch von anderen Anbietern bereitgestellt. Mit dem *Script Control* bietet Microsoft zudem die Möglichkeit, auf einfache Weise einen Scripting Host in eigene Anwendungen zu integrieren.

VBA

Zwar sind die VBA-Umgebungen (VBA steht für *Visual Basic for Applications*) nicht nach der Windows Scripting-Architektur konstruiert, in der Praxis sind sich die Architekturen aber sehr nahe: Ein Scriptprogrammierer, der auf Visual Basic Script (VBScript) setzt, kann

mit Cut&Paste des Quelltexts sowie mit ein paar einfachen Änderungen seine Scripts auch in VBA laufen lassen. Mit anderen Scriptsprachen geht das allerdings nicht.

5.5.2 Active Scripting Engines

Sprachinterpreter

Eine Active Scripting Engine ist ein Sprachinterpreter für eine Scriptsprache mit der Nebenbedingung, dass der Interpreter

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

COM-fähige Sprache versus Active Scripting-Sprache Die Anforderungen an eine Active Scripting-Sprache sind abzugrenzen von dem allgemeinen Begriff einer COM-fähigen Programmiersprache (Microsoft spricht von „COM-enabled Languages" – ins Deutsche zum Teil mit „COM-aktivierten Sprachen" übersetzt). Eine COM-fähige Sprache unterstützt die Nutzung von COM-Komponenten. Nicht jede COM-fähige Sprache ist auch eine Active Scripting-fähige Sprache. Beispielsweise unterstützen auch Delphi und PHP4 die Nutzung von COM-Komponenten. Dennoch sind beide Sprachen nicht im Rahmen des Active Scripting als Scriptsprachen einsetzbar.

Sprachen von Microsoft

Verfügbare Scriptsprachen Microsoft selbst hat bislang zwei Active Scripting Engines veröffentlicht:

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

Sprachen von anderen Herstellern

Es gibt weitere Scriptsprachen von anderen Anbietern (zum Großteil als Free- oder Shareware) in Form von Active Scripting Engines:

- PerlScript, Active Scripting-fähige Perl-Implementierung der Firma ActiveState [ACT00]
- PScript, Active Scripting-fähige Perl-Implementierung der Firma MKS [MKS00]
- IBM unterstützt im Rahmen seiner REXX-Implementierung unter dem Namen Object REXX Active Scripting. *Object REXX* ist seit Version 2.1 eine Active Scripting Engine [IBM01] [CAW01].
- PythonScript, Active Scripting-fähige Version von Python [PYT00a] [PYT00b]
- HaskellScript, Active Scripting-fähige Version der funktionalen Scriptsprache Haskell [HAS00]
- ActiveScriptRuby, Active Scripting-fähige Implementierung der objektorientierten Scriptsprache Ruby [RUB01a] [RUB01b]
- LUAScript, Active Scripting-fähige Implementierung der in Brasilien entwickelten Sprache LUA [LUA01a] [LUA01b]
- TclScript ist ein Active Scripting-Wrapper für die Nutzung des Tcl-Interpreters [TCL04].
- ForthScript von Mark Baker [BAK04]

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

Welche Scriptsprachen auf einem System installiert sind, erfährt man aus der Registrierungsdatenbank. Details dazu können Sie in Kapitel 7 nachlesen.

VBScript

Hinweise zur Auswahl der Scriptsprache in diesem Buch In diesem Buch wird durchgängig VBScript benutzt. VBScript ist die am häufigsten verwendete Sprache beim Windows Scripting. Auch auf Grund der weitgehenden Kompatibilität mit der Vollversion von Visual Basic ist VBScript die erste Wahl bei den Scriptsprachen unter Windows. Kapitel 8 stellt die Sprache ausführlich vor.

5.5.3 COM-Komponenten

Verfahren für den Systemzugriff

Ein Script benötigt den Zugriff auf das es umgebende System, um administrative Aufgaben und die Interaktion mit dem Anwender durchzuführen. Grundsätzlich gibt es für eine Scriptsprache zwei Möglichkeiten, wie sie diesen Zugriff herstellen kann: Zum einen können in der Sprache selbst Sprachkonstrukte und Funktionen integriert sein, die den Zugriff auf das System ermöglichen. Zum anderen kann die Sprache aber auch einen Mechanismus bereitstellen, um vorhandene Programmierschnittstellen (Application Programming Interfaces – APIs) anzusprechen.

Zugriffsmöglichkeiten auf APIs

Die erste Möglichkeit wird von fast allen Scriptsprachen hinsichtlich rudimentärer Ein- und Ausgabebefehle genutzt. Die Bereitstellung darüber hinausgehender Systemfunktionen bereitet jedoch Unannehmlichkeiten. Einerseits kann die Sprache kaum plattformunabhängig sein, da jedes Betriebssystem seine eigenen spezifischen Systemfunktionen bereitstellt. Andererseits muss die Sprache ständig erweitert werden, um mit den Veränderungen der System-APIs Schritt halten zu können. Beliebter ist daher die zweite Möglichkeit (Bereitstellung eines API-Zugriffsmechanismus), die jedoch dann beschwerlich ist, wenn unterschiedliche Arten von Programmierschnittstellen unterstützt werden müssen. Microsoft geht in der ActiveX Scripting-Architektur den zweiten Weg, mit der Prämisse der Einschränkung auf eine einzige Art von Programmierschnittstellen, nämlich auf *automationsfähige COM-Softwarekomponenten.* Man spricht auch von *Scripting-fähigen Komponenten* oder einfach *Scripting-Komponenten.* COM steht – wie schon oben erwähnt – für „Component Object Model".

COM-Komponenten

Im Kapitel zu COM werden Sie den Unterschied zwischen automationsfähigen und nichtautomationsfähigen COM-Komponenten genauer kennen lernen. So viel vorweg: COM-Automation ist in etwa gleichzusetzen mit einem späten Binden zur Laufzeit.

Das Active Scripting mit COM ist objektbasiert („COM-Objekte") mit instanziierbaren Klassen, die aus Methoden und Attributen bestehen. Die Klassen sind in der Regel in hierarchischen Objektmodellen angeordnet; für das Script ist die Komponente eine Objektbibliothek. Auf einen Zugriff auf Nicht-COM-APIs (z. B. DLLs, die keine COM-Komponenten sind) hat Microsoft ausdrücklich verzichtet. Es gibt jedoch inzwischen Ansätze, dies zu ermöglichen (vgl. DynaWrap).

Die Arbeit mit COM-Objekten verlangt einer Programmiersprache die Unterstützung einiger grundlegender Mechanismen ab. Sofern diese jedoch implementiert sind, kann die Sprache mit einer Vielzahl unterschiedlicher COM-Objekte aus unterschiedlichen COM-Komponenten zusammenarbeiten. Dies wird im Kapitel zu COM erklärt.

Auch wenn viele COM-Komponenten COM-Automation unterstützen, gibt es dennoch Komponenten, die den Dienst nicht anbieten und daher im Windows Scripting nicht verwendbar sind.

Typisierung

Komponententypen Komponenten erweitern die eingebauten Funktionen der Scriptsprachen und lassen sich aus Scripting-Sicht in zwei Typen einteilen:

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

Zahlreiche Komponenten

Es sind bereits zahlreiche Komponenten für den Zugriff auf unterschiedliche Betriebssystem- und Anwendungsfunktionen verfügbar: So ermöglichen COM-Komponenten unter Windows beispielsweise den Zugriff auf

- Betriebssystemfunktionen wie Windows-Benutzeroberfläche, Verzeichnisdienste, Dateisystem, Registrierungsdatenbank, Eventlog, Hardware, Scheduler, MTS/COM+, Dokumente (z.B. Text, HTML, XML) und Netzwerkprotokolle (z.B. TCP, IP, HTTP, FTP),
- Anwendungen wie Microsoft Office, Microsoft Exchange Server, Microsoft SQL Server, Internet Information Server, aber auch auf Produkte wie Lotus Notes, Corel Draw und SAP R/3.

Nicht alle Komponenten stammen von Microsoft selbst: Es gibt inzwischen unzählige Komponenten anderer Anbieter – zum Teil auch als Share- und Freeware. Auch selbst entwickelte COM-Komponenten können unabhängig von der Programmiersprache, in der sie implementiert wurden, verwendet werden. Anwendungen, die direkt komponentenbasiert entwickelt werden, lassen sich sehr einfach per Script ansteuern. Das Ende dieses Einleitungskapitel liefert einen Überblick über die wichtigsten Komponenten.

WSC

Windows Script Components (WSCs) *Windows Script Components (WSCs)* sind in Scriptsprachen geschriebene COM-Komponenten. Der Begriff WSC steht weder allgemein für Komponenten, die von Scripts aus genutzt werden können, noch für die Bausteine der Windows Scripting-Architektur. WSCs werden in Kapitel 19 behandelt.

Ausblick auf die Scripting-Komponenten in diesem Buch In den Kapiteln 10 bis 17 werden verschiedene COM-Komponenten zur Automatisierung administrativer Aufgaben in den Windows-Betriebssystemen sowie in Produkten der Microsoft Server System-Produktfamilie (ehemals BackOffice-Server und .NET Enterprise Server) beschrieben. Die Ausführlichkeit der Darstellung in diesem Buch richtet sich vor allem nach Bedeutung und Komplexität der Komponenten. Das *Active Directory Service Interface (ADSI)* zur Verzeichnisdienstverwaltung und die *Windows Management Instrumentation (WMI)* als übergreifender Ansatz zum Systemmanagement nehmen daher den größten Raum ein. In Wichtigkeit und Umfang folgen die *Scripting Runtime Library* für den Dateisystemzugriff und die

WSH Runtime. Darüber hinaus werden in den oben genannten Kapiteln auch zahlreiche weitere (kleinere) Scripting-Komponenten beschrieben.

Ein repräsentativer Ausschnitt

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

Auf eigener Suche Mit Sicherheit werden Sie nach der Lektüre dieses Kapitels noch die eine oder andere Funktionalität vermissen. Auf der Suche nach Komponenten hilft Ihnen das in Kapitel 18 vorgestellte Vorgehensmodell.

5.5.4 Werkzeugunterstützung

Editoren, Debugger und andere Tools

Die Werkzeugunterstützung der Scriptentwicklung unter Windows ist noch verbesserungswürdig. Mit Visual InterDev liefert Microsoft zwar eine Scriptentwicklungsumgebung; diese unterstützt aber bislang fast nur die Scriptprogrammierung im Web. In vielen Scripting Hosts (z.B. SQL Server Agent, Microsoft Outlook) stehen nur sehr primitive Editoren bereit, so dass die Scriptprogrammierung sehr mühsam ist.

Auch hinsichtlich des Debugging ist die Werkzeugunterstützung noch nicht optimal, wenn man die Entwicklungsumgebungen wie Visual C++ 6.0 und Visual Basic 6.0 als Maßstab nimmt. Inzwischen gibt es zum Teil bessere Lösungen von Drittanbietern. Einen Überblick über die verfügbaren Werkzeuge liefert Kapitel 18.

5.5.5 Active Scripting versus VBA

Das in Microsoft Office und vielen Anwendungen anderer Hersteller enthaltene Visual Basic for Applications (VBA) kann zum Teil als eine Konkurrenztechnologie zum Active Scripting angesehen werden. Auch VBA ist eine Interpreter-Sprache, die es ermöglicht, auf einfache Weise Anwendungen um Automatisierungsfähigkeit zu erweitern.

Die Unterschiede zwischen VBA und Active Scripting aus der Sicht eines Softwareherstellers, der seine Software anpassbar und erweiterbar machen möchte, zeigt die folgende Tabelle.

Visual Basic for Applications (VBA)	Active Scripting
• nur eine Programmiersprache (Visual Basic for Applications) • komfortable Entwicklungsumgebung wird von Microsoft bereitgestellt • Softwarehersteller zahlt an Microsoft	• viele Programmiersprachen (alle Active Scripting-fähigen Sprachen) • keine Entwicklungsumgebung (muss gegebenenfalls vom Softwarehersteller selbst erstellt werden) • kostenlos

5.6 Scripting im .NET Framework

Das .NET Framework

Das Microsoft .NET Framework ist eine Programmierplattform, die Microsoft im Jahre 2002 als Nachfolgeprodukt für das Component Object Model (COM) eingeführt hat. Das .NET Framework bietet gegenüber COM viele neue Möglichkeiten, umfasst jedoch andererseits in den ersten Versionen noch nicht alle Einsatzgebiete von COM. Echtes Scripting (interpretierter Code zur Ad-hoc-Programmierung) ist im .NET Framework grundsätzlich möglich, wird aber von dem Framework bisher nur rudimentär unterstützt. Es gibt von Microsoft noch keinen „Windows Script Host .NET (WSH.NET)", der auf dem .NET Framework aufsetzend eine Scripting-Architektur bereitstellt.

Scripting im .NET Framework ist durch zwei Ansätze möglich:

- Der Autor dieses Buchs hat selbst einen Scripting Host auf Basis des .NET Framework entwickelt, den DOTNET Scripting Host (DSH). Der DSH ist hinsichtlich seiner Nutzung dem COM-basierten WSH sehr ähnlich, steht aber auf einer ganz anderen technologischen Basis und bietet daher viele Vorteile. Der DSH ist Freeware; die Version 1.1 finden Sie auf der CD im Verzeichnis [CD:/install/dotnet/DSH]. Beschrieben ist der DSH in der 4. Auflage dieses Buchs. Zugunsten der Microsoft PowerShell ist dieses Kapitel in der 5. Auflage entfallen.
- Microsofts Ansatz zum Scripting mit dem .NET Framework ist die Microsoft PowerShell (PS), siehe nächstes Kapitel.

5.7 Die Microsoft Powershell

Das Active Scripting ist einigen Administratoren zu komplex, weil es viel Wissen über objektorientiertes Programmieren und das Component Object Models (COM). Die vielen Ausnahmen und Ungereimheiten (daher ist dieses Buch auch so dick) erschweren das Erlernen von Windows Script Host und seinen Host-Kollegen.

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

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

In einer ersten Beta-Version wurde die neue Shell schon unter dem Codenamen „Monad" auf der Professional Developer Conference (PDC) im Oktober 2003 in Los Angeles vorgestellt. Nach den Zwischenstufen „Microsoft Shell (MSH)" und „Microsoft Command Shell" trägt die neue Scriptumgebung seit Mai 2006 den Namen „Windows PowerShell". Derzeit aktuell ist der so genannte „Release Candidate", der den Ausführungen in diesem Buch zu Grunde liegt.

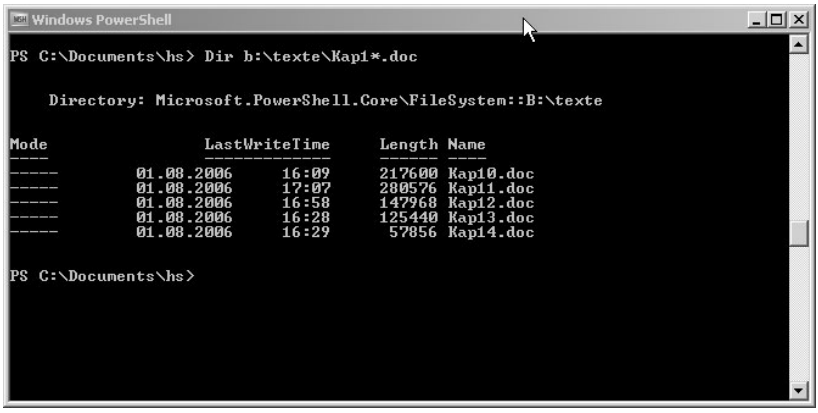


Bild 5.3: Verzeichnisauflistung in der Microsoft PowerShell Version 1.0

Die PowerShell sieht auf den ersten Blick aus wie ein DOS-Kommandozeilenfenster (siehe Abbildung 5.3), nur mit blauem statt schwarzem Hintergrund. Unter der Haube bietet sie jedoch folgende Highlights:

- eine den Unix-Shells ähnliche Scriptsprache,
- wahlweise starke oder schwache Typisierung sowie eine dynamische Erweiterung von Typen,
- über einhundert vordefinierte Befehle, die Commandlets genannt werden,
- typisiertes Pipelining (Objekt-Pipelining) zwischen Commandlets,
- einheitliches Navigationsmodell für Datenmengen (sowohl flache als auch hierarchische Datenmengen, z.B. Dateisystem, Registrierungsdatenbank, Active Directory, Umgebungsvariablen),
- Zugriff auf alle Objekte, die das .NET Framework, COM-Komponenten und die Windows Management Instrumentation (WMI) zur Verfügung stellen,
- eingebaute Vereinfachungen für ausgewählte Komponenten (insbesondere ADSI, WMI und System.Xml),
- eingebaute Sicherheitsfunktionen zum Schutz vor „bösen" Scripts,
- eigene Commandlets zur Erweiterung der Fähigkeiten der Powershell können in jeder .NET-basierten Programmiersprache geschrieben werden.

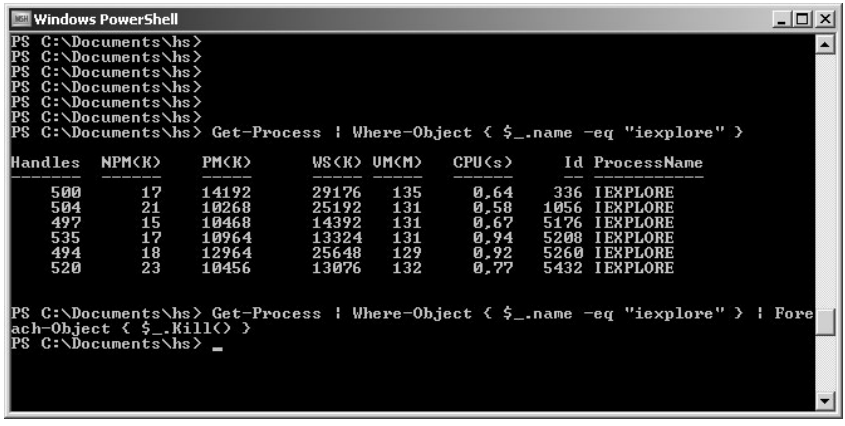
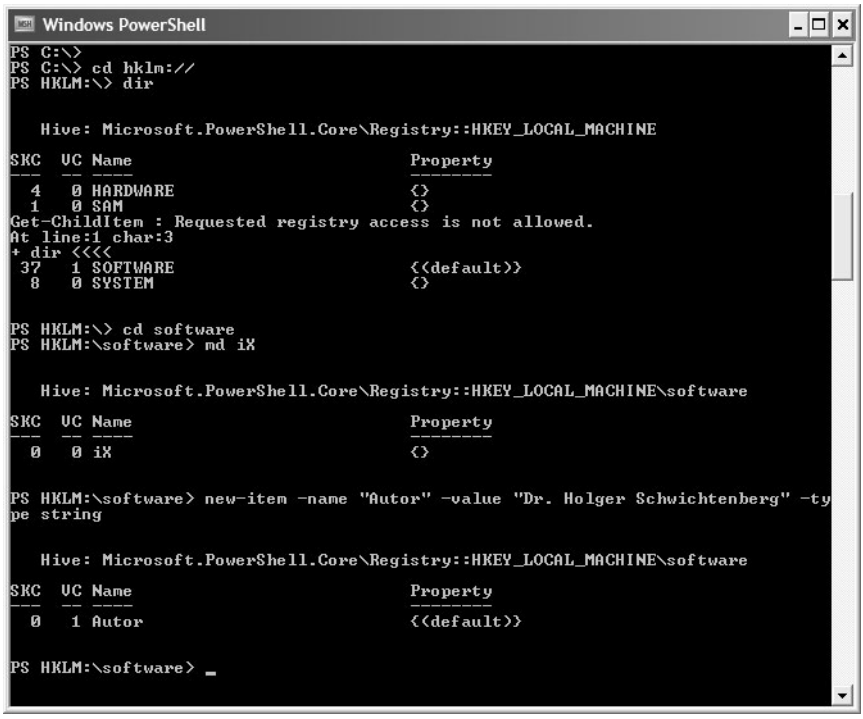


Bild 5.4: Objekt-Pipelining in Aktion in der PowerShell

Bild 5.5: Navigieren in der Registrierungsdatenbank



Die PowerShell wird ausführlich im Buchteil C behandelt.

5.8 Neuerungen in Windows Vista im Überblick

Fünf Jahre nach Windows XP erscheint Anfang 2007 endlich das neue Windows-Clientbetriebssystem mit Namen „Windows Vista", das vorher den Codenamen „Longhorn" trug.

Die folgenden Ausführungen basieren auf dem Release Candidate 1 von Windows Vista und der bis zum 7.9.2006 verfügbaren Dokumentation.

5.8.1 Scriptumgebungen

Der **Windows Script Host (WSH)** trägt in Vista die Versionsnummer 5.7 (gegenüber 5.6 in Windows XP und Windows Server 2003). Neue Funktionen sind aber im Release Candidate 1 weder erkennbar noch dokumentiert. Als neue Active Scripting-basierte Scriptumgebung sind die **Microsoft Gadgets** hinzugekommen. Gadgets sind scriptbasierte Mini-Anwendungen für Desktop und die neue Sidebar.

Die **Microsoft Powershell**, die einmal geplant wurde als ein Teil von Windows Vista, ist weder im Installationsumfang von Vista noch auf der CD-ROM enthalten, sondern muss separat aus dem Internet heruntergeladen und installiert werden.

5.8.2 Scriptbibliotheken

Windows Vista enthält zahlreiche neue Programmierschnittstellen. Interessanterweise sind darunter nicht nur moderne .NET-basierte Schnittstellen, sondern auch viele Schnittstellen, die auf alten Techniken wie C++ und COM basieren. Im Folgenden werden nur die Neuerungen genannt, die für das Thema dieses Buchs relevant sind.

COM-basierte Schnittstellen

- Erweiterungen in der **Windows Management Instrumentation** (**WMI**) sowohl beim Metaobjektmodell als auch bei den WMI-Klassen
- **Windows Remote Management** (**WinRM**) zum entfernten Zugriff auf WMI-Objekte über Webservices (XML/HTTP) (auch verfügbar für Windows Server 2003 Release 2)
- **Windows RSS Platform** („Microsoft Feeds, Version 1.0“) zum Zugriff auf Nachrichtenkanäle, die Really Simple Syndication (RSS) (auch für Windows XP und Windows Server 2003, wenn Internet Explorer 7.0 installiert wird)
- **Task Scheduler 2.0 Scripting Objects** zum Zugriff auf den neuen Zeitplandienst in Vista
- **OLEDB Provider for Windows Search** (**MSIDXS**) zur Suche im Vista-Dateisystem

.NET-basierte Schnittstellen

Die .NET-basierten Schnitsttellen in Windows Vista sind im **.NET Framework 3.0** zusammengefasst. Gegenüber dem .NET Framework 2.0 bietet die Version 3.0 folgende neuen Bibliotheken:

- Die **Windows Presentation Foundation** (**WPF**) für grafische Benutzerschnittstellen (Codename Avalon)
- Die **Windows Communication Foundation** (**WCF**) für Fernaufruf und Anwendungskopplung (Codename Indigo)
- Mit der **Windows Workflow Foundation** (**WF**) können Entwickler eigene Anwendungen um rechnergesteuerte Arbeitsabläufe erweitern.
- **Windows Cardspaces** ist eine Bibliothek zur Verwaltung digitaler Identitäten.

.NET Framework 3.0

Das .NET Framework 3.0 besteht aus der gleichen Laufzeitumgebung (Common Language Runtime – CLR) wie das .NET Framework 2.0. Auch die Syntax der Programmiersprachen ist gleich. Neu sind lediglich die zusätzlichen Bibliotheken, die optional verwendet werden können.

Das .NET Framework 2.0 kann auch auf älteren Windows-Versionen als Add-on installiert werden. Allerdings lassen sich die neuen oben genannten Bibliotheken nur auf Windows XP und Windows Server 2003 installieren.

6 Scripting-Schnellstart

Wenn Sie noch nie ein Script unter Windows erstellt haben, werden Ihnen die folgenden vier Beispiele erste Erfolgserlebnisse bereiten.

WSH 2.0 und IE ab 4.0

Voraussetzung für das erste Beispiel ist, dass Sie den Windows Scripting Host installiert haben. Sie sollten in Ihrem Windows-Verzeichnis eine Datei *WScript.exe* finden. Wenn Sie Windows 2000 benutzen, können Sie fast sicher sein, dass Sie den WSH 2.0 auf Ihrem System haben. Unter Windows 98 ist der WSH in der Version 1.0 eine Installationsoption; unter NT 4.0 gibt es den WSH nur als separates Add-on. Windows XP und Windows Server 2003 enthalten den WSH 5.6. Wenn der WSH 2.0 nicht vorhanden ist, installieren Sie ihn bitte von der Buch-CD aus dem Verzeichnis */install/hosts/wsh*. Voraussetzung für das zweite Beispiel ist ein installierter Internet Explorer ab Version 5.0.

Die beiden Scripts befinden sich natürlich auch auf der Buch-CD [CD:/code/Einfuehrung/]. Jedoch sollten Sie sich an dieser Stelle durchaus die Mühe machen, die Scriptdateien selbst zu erstellen.

6.1 Ein einfaches Script für den Windows Script Host

Ihr erstes Script

So erstellen Sie Ihr erstes Script für den Windows Scripting Host in der Sprache Visual Basic Script:

- Legen Sie eine Textdatei an, indem Sie irgendwo auf dem Desktop oder in einem Verzeichnis im Dateisystem im Kontextmenü *Neu|Textdatei* wählen. Es erscheint eine Datei *Neue Textdatei.txt*.
- Benennen Sie die Datei in *ErstesSkript.vbs* um. Bestätigen Sie die Nachfrage des Betriebssystems, ob die Dateierweiterung wirklich geändert werden soll.
- Wählen Sie aus dem Kontextmenü der Datei *Bearbeiten,* so dass sich der Notepad öffnet. (Sofern Sie einen anderen Editor installiert haben, mag jetzt dieser gestartet werden.)
- Geben Sie Folgendes in die erste Zeile ein:

```
MsgBox "Ab heute kann ich skripten!"
```

- Speichern Sie die Änderungen ab. Sie können den Editor schließen, müssen es aber nicht.

- Doppelklicken Sie auf die Datei *ErstesSkript.vbs*. Wenn Sie alles richtig gemacht haben und das System Ihnen wohlgesonnen ist, wird die nachstehend abgebildete Dialogbox erscheinen.

Bild 6.1: Ausgabe des Scripts „Erstes-Skript.vbs“

6.2 Ein komplexeres Script mit zwei Sprachen für den Internet Explorer

Ihr zweites Script

Das zweite Beispiel wird Ihnen bereits zeigen, wie Sie zwei Scriptsprachen innerhalb einer Scriptdatei mischen können. Als Scripting Host wird der Internet Explorer eingesetzt.

- Legen Sie eine Textdatei an, indem Sie irgendwo auf dem Desktop oder in einem Verzeichnis im Dateisystem im Kontextmenü *Neu|Textdatei* wählen. Es erscheint eine Datei *Neue Textdatei.txt*.
- Benennen Sie die Datei um in *ZweitesSkript.htm*. Bestätigen Sie die Nachfrage des Betriebssystems, ob die Dateierweiterung wirklich geändert werden soll.
- Wählen Sie aus dem Kontextmenü *Öffnen mit* und dort *Notepad*. (Wenn Sie einen HTML-Quelltexteditor auf Ihrem System haben, können Sie auch diesen nutzen.)
- Geben Sie die folgenden Zeilen ein:

```
<HTML>Beispiel für die Mischung von ActiveX-Sprachen
<SCRIPT language="JavaScript">
// ----- Unterroutine in Jscript
function jadd(a,b)
{ return(a+b) }
</script>
<script language="VBScript">
' ------ Hauptprogramm in VBScript
x = 5
y = 6
Ergebnis = jadd(x,y) ' JScript zur Addition nutzen
msgbox x & " + " & y & " = " & ergebnis
</SCRIPT></HTML>"
```

- Speichern Sie die Änderungen ab. Sie können den Editor schließen, müssen es aber nicht.
- Doppelklicken Sie auf die Datei *ZweitesSkript.htm*. Wenn Sie alles richtig gemacht haben und das System korrekt arbeitet, wird der Internet Explorer mit nachstehend abgebildeter Dialogbox erscheinen.

Bild 6.2: Ausgabe des Scripts im Internet Explorer

6.3 Benutzer aus dem Active Directory exportieren

Zur Einstimmung auf die großen Möglichkeiten im Scripting des Active Directory soll hier ein Beispiel dienen, das eine Liste aller Benutzer in einem bestimmten Active Directory-Container, z.B. dem Users-Container, oder einer Organisationseinheit (OU) ausgibt. Das Script umfasst die folgenden Schritte:

1. Festlegung des aufzulistenden Ordners in einer Konstanten zu Beginn des Scripts. In dieser Konstante müssen Sie einen gültigen Active Directory-Container Ihres eigenen Netzwerks in Form eines LDAP-Pfads eintragen. Der Pfad muss mit `LDAP://` beginnen, wobei die komplette Großschreibweise des Worts LDAP zu beachten ist.
2. `GetObject()` holt eine Referenz auf den Verzeichniscontainer und speichert diese in der Variablen `oContainer`.
3. Der Befehl `oContainer.Filter = Array("user")` dient dazu, die folgende Ausgabe auf die Benutzerkonten zu beschränken. Sonst würden auch in dem Container enthaltene Benutzergruppen und andere Verzeichnisdienstobjekte ausgegeben. Sowohl Gruppen als auch Benutzer erhalten Sie mit `oDomain.Filter = Array("user", "group")`.
4. `WScript.Echo` gibt den Namen des aufzulistenden Containers aus.
5. `For Each...Next` bildet eine Schleife, in der nacheinander alle Benutzerkonten an die Variable `oUser` gebunden werden.
6. Innerhalb der Schleife wird zu jedem Benutzer der Verzeichnisname, der Anzeigename und der Beschreibungstext ausgegeben.

```
Const Pfad = "LDAP://ou=Mitarbeiter,dc=it-visions,dc=de"
Set oContainer = GetObject(Pfad)
oContainer.Filter = Array("user")
wscript.echo "Liste der Benutzer in: " & oContainer.Name
For Each oUser In oContainer
    wscript.echo oUser.Name & ";" & _
    oUser.DisplayName & ";" & _
    oUser.Description
Next
```

Listing 6.1: Liste der Benutzer in einem Active Directory-Container [Benutzerliste.vbs]

Schritte Geben Sie das Script in einem Editor ein (oder verwenden Sie das Script auf der Buch-CD-ROM). Vergessen Sie nicht, den Pfad in der ersten Zeile auf Ihre Umgebung anzupassen. Speichern Sie das Script unter dem Namen *benutzerliste.vbs*. Das Script sollte man unbedingt mit *cscript.exe* in der DOS-Eingabeaufforderung starten, also z.B. mit

```
cscript.exe benutzerliste.vbs
```

Würde man das Script mit dem Standardhost *wscript.exe* starten, würde für jedes Benutzerkonto ein Dialogfenster erscheinen. Die nächste Abbildung zeigt die mögliche Ausgabe, die man auch sehr einfach in eine Textdatei umleiten kann:

```
cscript.exe benutzerliste.vbs >Ausgabedatei.cvs
```

Bild 6.3: Ausgabe der Liste der Benutzer im Active Directory-Container „Users“

Wenn Microsoft Excel installiert ist, lädt ein Doppelklick auf die durch Semikola getrennte Datei die Daten in Excel. Aus Microsoft Access heraus kann die Datei sehr einfach mit der Import-Funktion in eine Datenbank eingelesen werden.

Das obige Script im Listing kann auch in NT 4.0-Domänen eingesetzt werden. Dann ist als Pfad aber „WinNT://domaenenname“ anzugeben. Auch hier ist die Groß-/Kleinschreibung von WinNT exakt zu beachten.

6.4 Ein PowerShell-Beispiel

Für dieses Beispiel müssen Sie zunächst einmal die Microsoft PowerShell installieren. Die PowerShell ist auf der Buch-CD-ROM enthalten (*/install/powershell*). Da zum Redaktionsschluss noch nicht die finale Version vorlag, sollten Sie die aktuelle Version bei Microsoft herunterladen (deren HTTP-Adresse natürlich auch noch nicht bekannt ist).

1. Starten Sie die PowerShell.
2. Geben Sie folgenden Befehl ein:

```
get-wmiobject Win32_useraccount | where-object {$_.passwordexpires -eq 0 } |
select-object Name,Domain
```

3. In der Ausgabe sehen Sie dann eine Liste der Benutzerkonten, deren Kennwort nicht abläuft.

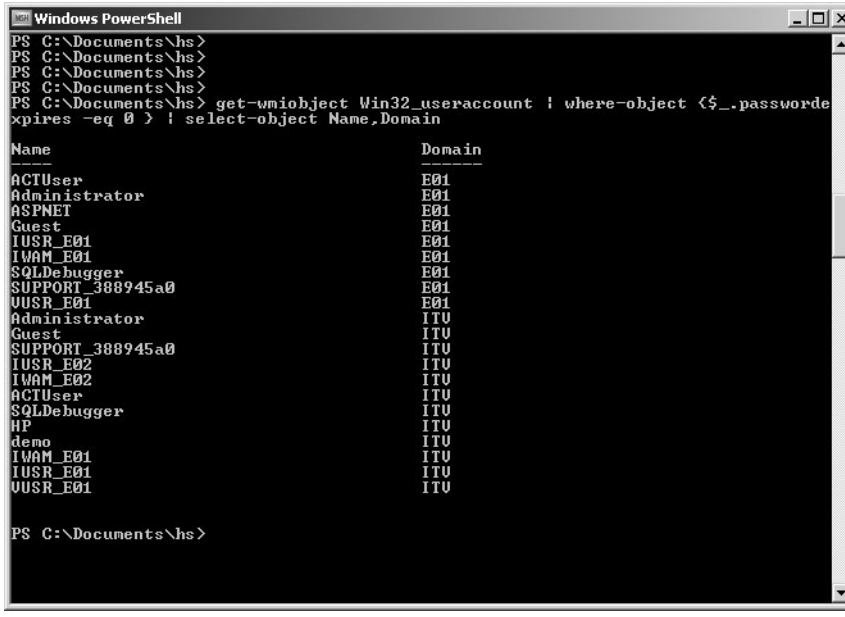


Bild 6.4: Liste der Benutzerkonten, bei denen man das Kennwort nicht wechseln muss