Microsoft®

Tworzenie skryptów w Microsoft® Windows®

Podręcznik do samodzielnej nauki

Ed Wilson

Tworzenie skryptów w Microsoft Windows. Podręcznik do samodzielnej nauki
Edycja polska Microsoft Press
Tytuł oryginału: Microsoft® Windows® Scripting Self-Paced Learning Guide

Polish edition by APN PROMISE Sp. z o. o. Warszawa 2005

APN PROMISE Sp. z o. o., biuro: 00-108 Warszawa, ul. Zielna 39
tel. (022) 351 90 00, faks (022) 351 90 99
e-mail: mspress@promise.pl

ISBN: 83-88440-70-5

Przekład:
Dominik Gotojuch
Redakcja:
Marek Włodarz
Korekta: Dorota Wojdanowicz
Skład i łamanie: Marek Włodarz

1 Początki

Rozdział rozpoczynający naukę o automatyzacji Microsoft Windows Server 2003 na początku zawiera analizę kilkunastu skryptów napisanych w Microsoft Visual Basic Scripting Edition (VBScript). Później Czytelnik podejmie pierwsze próby samodzielnego pisania skryptów. Materiał z tego rozdziału będzie potrzebny do zrozumienia kolejnych omawianych zagadnień, a także może okazać się przydatny w pracy administratora sieci, zatem Czytelnik musi mieć pewność, że zrozumiał go, nim przystąpi do dalszej lektury.

Zanim rozpoczniesz

Do zrozumienia zawartości tego rozdziału wymagana jest znajomość:

- Podstawowej administracji Windows Server 2003

Po ukończeniu tego rozdziału Czytelnik będzie umiał:

- Radzić sobie z podstawowymi błędami
- Połączyć się z obiektem systemu plików
- Rozróżnić cztery części skryptu
- Deklarować zmienne
- Tworzyć wersję wykonywalną
- Odczytywać rejestr
- Uruchamiać skrypty
- Korzystać z klauzuli *Option Explicit*

Uruchomienie pierwszego skryptu

Musisz odpowiedzieć na pytanie, czy serwer ma zainstalowane narzędzia administracyjne, ale nie masz czasu ani siły na manualne sprawdzanie. A do sprawdzenia masz setki serwerów... Rozwiązaniem jest napisanie prostego skryptu, który sprawdza pojedynczy serwer:

```
Set objShell = CreateObject("Shell.Application")
Set colTools = objShell.Namespace(47).Items
For Each objTool in colTools
    WScript.Echo objTool
Next
```

Krok po kroku:

▶ **Aby uruchomić istniejący skrypt:**

1. Otworzyć linię poleceń (w menu Start wybieramy Run\CMD)
2. Zmienić katalog na **\BookScripts\ch1**.
3. Wpisać **CScript CheckAdminTools.vbs** i nacisnąć Enter.

Najlepszym sposobem nauki pisania skryptów jest ich czytanie. Czym jest skrypt? W omawianym kontekście to nic innego, jak zbiór poleceń, zawartych w pliku tekstowym. Upodabnia to skrypty do plików wsadowych, używanych przez administratorów sieciowych od czasów DOS. Podobnie jak pliki wsadowe, skrypty mogą być pisane w prostym Notatniku Windows. Różni je jednak elastyczność i potęga języka skryptowego. W tej części zostanie dokonana analiza i rozróżnienie wspólnych elementów różnych skryptów. Pomoże to Czytelnikowi zrozumieć zasadę działania elementów każdego skryptu.

Krok po kroku:

▶ **Aby otworzyć istniejący skrypt:**

1. Otworzyć Notatnik
2. Z menu plików wybrać Otwórz. W ramce Typ plików na liście rozwijanej wybrać Wszystkie pliki.
3. Wybrać katalog, w którym znajduje się VBScript.
4. Wybrać plik i polecenie Otwórz.

W dalszej części rozdziału zostanie omówiony następujący skrypt:

```
Option Explicit
On Error Resume Next
Dim objShell
Dim regActiveComputerName, regComputerName, regHostname
Dim ActiveComputerName, ComputerName, Hostname

regActiveComputerName = "HKLM\SYSTEM\CurrentControlSet\Control\" & _
    "ComputerName\ActiveComputerName\ComputerName"
regComputerName = "HKLM\SYSTEM\CurrentControlSet\Control\" & _
    "ComputerName\ComputerName\ComputerName"
regHostname = _
    "HKLM\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters\Hostname"

Set objShell = CreateObject("WScript.Shell")
ActiveComputerName = objShell.RegRead(regActiveComputerName)
ComputerName = objShell.RegRead(regComputerName)
Hostname = objShell.RegRead(regHostname)
```

```
WScript.Echo ActiveComputerName & " is active computer name"
WScript.Echo ComputerName & " is computer name"
WScript.Echo Hostname & " is host name"
```

Zawiera on dość dużo informacji. Dla ułatwienia można go rozbić na części:

- Część nagłówkowa
- Część odniesieniowa
- Część wykonawcza
- Część wyjściowa

Część nagłówkowa

Część nagłówkowa jest częścią administracyjną skryptu. Na ogół można pozostawić ją pustą, bez utraty funkcjonalności skryptu. W przedstawionym powyżej skrypcie tak naprawdę jest zupełnie niepotrzebna i można ją bez wahania usunąć (czynność tę opisano w ramach jednego z ćwiczeń na końcu rozdziału). Mimo że część nagłówkowa nie jest konieczna do poprawnego działania skryptu, czyni go bardziej przejrzystym i umożliwia kontrolę działania skryptu (innego niż domyślne). Więcej na ten temat znajduje się w punktach dotyczących poleceń *Option Explicit* i *On Error Resume Next*. W podanym skrypcie częścią nagłówkową jest następujący fragment kodu:

```
Option Explicit
On Error Resume Next
Dim objShell
Dim regActiveComputerName, regComputerName, regHostname
Dim ActiveComputerName, ComputerName, Hostname
```

Mimo pozornej złożoności wykorzystano tu tylko trzy polecenia: *Option Explicit*, *On Error Resume Next* i *Dim*. Każde z nich omówimy w kolejnych punktach.

Krótko

P. Podać jeden ze sposobów uruchamiania VBScript?
O. Wpisać CScript przed nazwą pliku .vbs w linii poleceń..

P. Jakim narzędziem można odczytać zawartość pliku .vbs?
O. Notatnikiem.

P. Jakie polecenia znajdują się w części nagłówkowej VBScript?
O. *Option Explicit*, *On Error Resume Next* i *Dim*.

Option Explicit

Polecenie *Option Explicit* określa, iż każda wykorzystana w skrypcie zmienna zostanie wcześniej zadeklarowana.

Jeśli w części nagłówkowej VBScript znajduje się polecenie *Option Explicit*, zmienna musi zostać przed wykorzystaniem zadeklarowana. Jeśli polecenie *Option Explicit* zostanie pominięte, VBScript domyślnie przyjmuje, że nierozpoznawalne wyrażenia są zmiennymi. Deklaracja zmiennej następuje przez użycie polecenia *Dim*, jak w skrypcie przedstawionym wcześniej. *Dim* jest skrótem od *dimension*. Nazwa ta wynika ze sposobu operowania zmiennymi (zarezerwowanie części pamięci, przeznaczonej do przechowywania danych).

Uwaga Zmienna jest definiowana jako nazwana przestrzeń, przechowująca dane, które mogą zostać zmodyfikowane podczas pracy programu. Dla uproszczenia można założyć, że zmienne to odpowiednio nazwane informacje, przechowywane w skrypcie.

On Error Resume Next

Polecenie *On Error Resume Next* służy do wykonania przez skrypt konkretnej czynności w razie wykrycia błędu. Tutaj *Resume Next* kontynuuje pracę skryptu, odczytując kolejną linię kodu. Takie pominięcie linii kodu wywołujących błędy nazywane jest obsługą błędów. Warto wykorzystać to polecenie w skryptach logowania, gdyż możliwe błędy mogą całkowicie zablokować dostęp do serwera. Oczywiście każdy skrypt powinien zostać parokrotnie sprawdzony przed implementacją, jednak zawsze istnieje możliwość pominięcia jakiegoś szczegółu. W dalszych punktach znajduje się dokładniejszy opis obsługi występujących błędów.

Uwaga Mimo że polecenie *On Error Resume Next* jest niezwykle przydatne w skryptach, nie powinno się z niego korzystać w trakcie nauki. Jego działanie blokuje komunikaty o ewentualnych błędach, więc jeśli skrypt z tym poleceniem nie działa poprawnie, należy w pierwszej kolejności usunąć to polecenie.

Dim

Powyższy kod wielokrotnie wykorzystuje polecenie *Dim*. Jak już wiadomo, służy ono do deklarowania zmiennych takich, jak *objShell* i wszystkich innych, znajdujących się w skrypcie na końcu tej części (z wyjątkiem *Dim*). Mogą one się nazywać dowolnie, nawet *a*, *b*, *c*, *d*, co skraca wstawianie ich w kodzie. Jednak należy pamiętać, że odpowiednia nazwa zmiennej czyni skrypt bardziej przejrzystym i bardziej zrozumiałym. W poniższym przykładzie łatwo się domyśleć, że zmienna o nazwie *ComputerName* przechowuje nazwę komputera. Zmienne *regActiveComputerName*, *regComputerName* i *regHostName* różnią się od zmiennych *ComputerName*, *ComputerName* i *HostName* jedynie przedimkiem, który porządkuje je według przeznaczenia. Pierwsze trzy zmienne przechowują klucze rejestru, a trzy następne – ich wartości.

```
Dim objShell
Dim regActiveComputerName, regComputerName, regHostName
Dim ActiveComputerName, ComputerName, Hostname
```

Krótko

P. **Do czego służy polecenie *Option Explicit*?**
O. Informuje VBScript, że zmienne będą deklarowane przed wykorzystaniem.

P. **Do czego stosuje się polecenie *On Error Resume Next*?**
O. Do prostej obsługi błędów.

P. **Jakie zastosowanie ma polecenie *Dim*?**
O. Służy do deklaracji zmiennych.

Część odniesieniowa

Część odniesieniowa skryptu umożliwia przypisywanie wartości zmiennym zdeklarowanym w części nagłówkowej. Jedną z przyczyn korzystania ze zmiennych jest utworzenie prostej nazwy dla złożonych danych. Ułatwia to pracę z kodem skryptu. W podanym kodzie zdeklarowanym w części nagłówkowej zmiennym przypisano odpowiednie wartości.

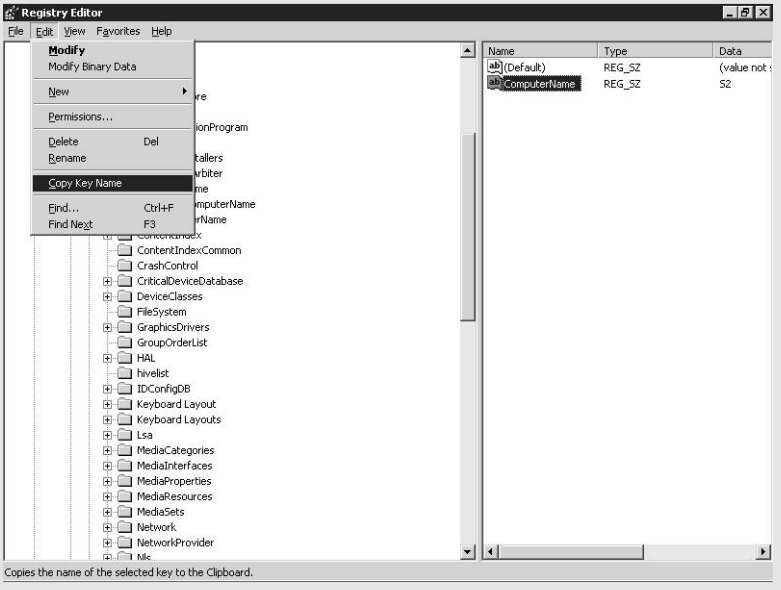
```
regActiveComputerName = "HKLM\SYSTEM\CurrentControlSet\Control\" &_
    "ComputerName\ActiveComputerName\ComputerName"
regComputerName = "HKLM\SYSTEM\CurrentControlSet\Control" &_
    "\ComputerName\ComputerName\ComputerName"
regHostname = "HKLM\SYSTEM\CurrentControlSet\Services" &_
    "\Tcpip\Parameters\Hostname"
```

Łatwo zauważyć, że elementy o tym samym znaku po prawej stronie przypominają klucze rejestru. Właśnie dlatego w kodzie pojawiają się wspomniane zmienne z przedimkiem *reg*. Każda z nich zostaje przypisana do klucza rejestru. Na przykład z pierwszej linii kodu wynika, że *regActiveComputerName* przypisany jest bardzo długi łańcuch *HKLM\SYSTEM\CurrentControlSet\Control\ComputerName\ActiveComputerName\ComputerName*. (*HKLM* to skrót od HKEY_LOCAL_MACHINE. VBScript odczytuje ten skrót, nie trzeba więc nic dodatkowo pisać). Część odniesieniowa:

- Ogranicza ilości wpisów w kodzie, zapewniając tym samym mniejsze prawdopodobieństwo wystąpienia błędu. Dzięki temu nie trzeba powtarzać długich wpisów.
- Czyni skrypt bardziej przejrzystym. Jeśli zmienna występuje w kodzie parokrotnie, wystarczy raz przypisać jej wartość.
- Ułatwia późniejsze modyfikacje skryptu. Na przykład zapisany wyżej skrypt odczytuje nazwy komputerów. Jeśli zmieni się klucz rejestru, skrypt może odczytywać z niego jakiekolwiek inne informacje.

Uzyskanie odpowiedniego klucza rejestru

Aby zapewnić skryptom uzyskanie odpowiedniego klucza rejestru, najlepiej wykorzystać funkcję Copy Key Name(Kopiowanie Nazwy Klucza) w programie Registry Editor (Regedit.exe)., jak to jest przedstawione na rysunku 1-1. Należy wybrać klucz rejestru zawierający odpowiednie informacje, otworzyć menu edycji, wybrać Copy Key Name z listy. Cała nazwa klucza zostaje skopiowana do schowka, skąd można ją wkleić do skryptu.



Rysunek 1-1 Funkcja Copy Key Name w programie Registry Editor

Część wykonawcza

Ta część, jak sugeruje nazwa, wykonuje instrukcje zawarte w skrypcie, korzystając ze zadeklarowanych i użytych wcześniej zmiennych.

Uwaga Dotychczas nie pojawiły się żadne opisy dotyczące WScript, wykorzystywanego do tworzenia obiektów, oraz sposobu tworzenia obiektów systemu plików. Zagadnienia te zostaną wyjaśnione w dalszych rozdziałach. Na razie Czytelnik powinien skupić się na działaniu i funkcjonalności skryptu..

Oto kolejny przykład:

```
Set objShell = CreateObject("WScript.Shell")
Set objFileSystem = CreateObject("Scripting.FileSystemObject")
```

```
ActiveComputerName = objShell.RegRead(regActiveComputerName)
ComputerName = objShell.RegRead(regComputerName)
Hostname = objShell.RegRead(regHostname)
```

Czytelnik powinien już rozpoznawać zmienne w kodzie. Są nimi między innymi *objShell* i *objFileSystem*, które mają przypisane jakieś wartości. Można dowiedzieć się, jakie to wartości, analizując pierwszą linię kodu:

```
Set objShell = CreateObject("WScript.Shell")
```

Rozpoczynające linię wyrażenie *Set* jest komendą VBScript, przypisującą odniesienie do obiektu do zmiennej. Aby VBScript odczytywał rejestr, musi się z nim połączyć. Działa to podobnie, jak w wypadku baz danych – do odczytu wymagane jest połączenie się z bazą. Tworzenie odniesienia do obiektu następuje po użyciu polecenia *Set.*

VBScript korzysta z możliwości innych programów za pomocą obiektów automatyzacji. Umożliwia to tworzenie potężniejszych skryptów, nadających się do zarządzania złożonymi środowiskami sieciowymi. Dzięki tym możliwościom można wyświetlać informacje z wykorzystaniem technologii wizualizacji i formatowania programów z Microsoft Office System, zamiast ograniczać się do czarno-białej, wyłącznie tekstowej linii poleceń.

Do powstałego odniesienia można przypisać zmienną objShell stosując polecenie *CreateObject*. Znak równości występujący za *objShell* wskazuje, iż zmienna ta powinna być równa jakiejś wartości. Tutaj jest to dalsza część kodu za znakiem równości lub *CreateObject("WScript.Shell")*. Czasownik *Create* (tworzyć) w poleceniu *CreateObject* sugeruje, że wykonana zostanie jakaś czynność. Ta linia kodu przydziela połączenie zmiennej *objShell*, umożliwiając skryptowi odczyt rejestru.

Uwaga Zamiast typowego *CreateObject* VBScript, aby przypisać odwołania do obiektu można także posłużyć się *WScript.CreateObject*. Obydwie te metody działają bez zarzutu.

Teraz zmienne *ActiveComputerName* i *regActiveComputerName* można wykorzystać do odczytu rejestru, korzystając z możliwości zmiennej *objShell*. Zmienna *regActiveComputerName* została wcześniej zadeklarowana jako klucz rejestru, zawierający dane o aktywnym komputerze. Teraz należy zmodyfikować zmienną *ActiveComputerName* tak, aby zwracała wartość klucza rejestru. To samo trzeba zrobić z dwoma pozostałymi kluczami rejestru.

Dotychczas udało się zachować w pamięci nazwy trzech komputerów, wykorzystując zmienne *ActiveComputerName*, *ComputerName* i *Hostname*. Żeby zmienne przechowywały nazwy komputerów, muszą one wcześniej być odczytane z trzech różnych kluczy rejestru. Na potrzeby tego odczytu stworzyliśmy trzy zmienne o nazwach *regActiveComputerName*, *regComputerName* i *regHostname*. Przedimek *reg* oznacza zmienne bezpośrednio powiązane z kluczami rejestru. Następnie zastosowaliśmy polecenie *RegRead* zmiennej *objShell,* przypisanej do odniesienia do obiektu za pomocą polecenia *CreateObject*. Zgromadzone przez skrypt informacje można wykorzystać na wiele sposobów. Kolejny skrypt przykładowy przedstawia możliwości wykonawcze VBScript.

Część wyjściowa

Zgromadzenie informacji z rejestru okazuje się nieprzydatne, jeśli nie można z nimi zrobić nic więcej. Możliwości ich użycia daje część wykonawcza skryptu. Naturalnie skrypt nie musi służyć do generowania wyniku pracy. Może on wykorzystać zgromadzone informacje chociażby do monitorowania działania usługi i restartowania jej w razie błędu, jednak większość administratorów sieciowych i tak chciałaby zobaczyć wpis dotyczący restartu. W kolejnym skrypcie wynik zostanie przedstawiony za pomocą serii poleceń *Echo*. Sposób korzystania z polecenia *Wscript.Echo* obrazuje kod:

```
WScript.Echo activecomputername & " is active computer name"
WScript.Echo ComputerName & " is computer name"
WScript.Echo Hostname & " is host name"
```

Polecenie *WScript.Echo* wypisuje tekst w linii poleceń lub wywołuje okno z wiadomością, w zależności od tego, jak wykonywany jest VBScript. Jeśli uruchamia się go z wykorzystaniem CScript, jak było to opisane wcześniej w procedurze „Krok po kroku: Aby uruchomić istniejący skrypt", zapis odbywa się w interpretatorze poleceń.

Ustanowione zmienne są równoważne informacjom kluczy rejestru z ostatniej części skryptu. *Echo* jest tu funkcją powtarzania. Jako że zmienne powiązane są z łańcuchami, będącymi częścią kluczy rejestru (poprzez część odniesieniową), możemy skorzystać z *WScript.Echo* do zapisu wartości przechowywanych obecnie w zmiennych. Za znakiem „&", oznaczającym „i", w kodzie znajduje się wyrażenie w cudzysłowie. Łączy się ono wraz z wartością zmiennej, po lewej stronie znaku „&". Takie łączenie zwane jest *konkatenacją*. Funkcja *Echo* operuje na przechowywanych wartościach zmiennych, a do każdej z nich dołączany jest tekst, wyjaśniający, jakie role pełnią. Uruchomienie skryptu powinno zaowocować wynikami, przedstawionymi na rysunku 1-2.

Rysunek 1-2 Wyniki pracy DisplayComputerNames.vbs

Już trzy okna z wiadomościami zapełniają ekran, a co dopiero gdyby były ich setki. Istnieją skrypty, które zwracają listy, składające się z ponad tysiąca elementów (na przykład skrypt średniej wielkości domeny, ustawiający użytkowników w kolejce). Wystarczy wymyślić efektywniejszą metodę rozpisania wyników. Istnieje kilkanaście sposobów, jak chociażby wykorzystanie okien wiadomości VBScript, ale o tym będzie dopiero w rozdziale 2.

Udoskonalanie skryptu

W celu udoskonaleniu funkcjonalności skryptu, gotowy skrypt można odpowiednio modyfikować. Spróbujemy dodać do skryptu poniższe funkcje:

- Tworzenie dokumentacji, zawierającej elementy przećwiczone w poprzednich paragrafach
- Zapis danych, innych niż dotychczas przechowywane nazwy komputerów

Dokument, który dzwoni do domu

Na początku zostanie utworzona dokumentacja wykonywanych ćwiczeń.

Dokumentację dodajemy wpisując informacje wewnątrz skryptu. Trzeba jednak zaznaczyć, iż dodajemy tekst, aby uniknąć błędów skryptu. Można to zrobić na parę sposobów. Najefektywniejszym z nich jest poprzedzenie każdej notki apostrofem ('), po którym następuje tekst wyjaśniający (komentarz). Przedstawiony poniżej skrypt jest efektem dodania objaśnionych powyżej elementów:

```
' Skrypt odczytuje nazwy komputerów z rejestru
Option Explicit     'Zmusza skryptującego do deklaracji zmiennych
On Error Resume Next 'Wydaje VBScript polecenie przejścia do następnej linii
                'zamiast przerywania pracy skryptu w razie wystąpienia błędu
' Za pomocą polecenia Dim deklaruje się nazwy zmiennych w skrypcie
Dim objShell
Dim regActiveComputerName, regComputerName, regHostname
Dim ActiveComputerName, ComputerName, Hostname
' Użycie nazwy zmiennej, a następnie znaku równości (=) oznacza, iż dane
' zmiennej znajdują się po prawej.
' Klucze rejestru są dość długie, najlepiej jest więc dzielić zawierające
' je linie na dwie.
regActiveComputerName = "HKLM\SYSTEM\CurrentControlSet" & _
    "\Control\ComputerName\ActiveComputerName\ComputerName"
regComputerName = "HKLM\SYSTEM\CurrentControlSet\Control" & _
    "\ComputerName\ComputerName\ComputerName"
regHostname = "HKLM\SYSTEM\CurrentControlSet\Services" & _
    "\Tcpip\Parameters\Hostname"

Set objShell = CreateObject("WScript.Shell")
ActiveComputerName = objShell.RegRead(regActiveComputerName)
ComputerName = objShell.RegRead(regComputerName)
Hostname = objShell.RegRead(regHostname)

' Okna komunikatu wywołuje się poleceniem WScript.Echo
' i podaniem zawartości komunikatu.
WScript.Echo activecomputername & " is active computer name"
WScript.Echo ComputerName & " is computer name"
WScript.Echo Hostname & " is host name"
```

Krok po kroku

▶ **Dodawanie dokumentacji w skrypcie:**

1. Otworzyć skrypt w Notatniku.
2. Poprzedzić linię pojedynczym apostrofem (').
3. W pierwszej linii skryptu, po apostrofie, wpisać krótki opis zastosowania skryptu.
4. Zachować skrypt.

Modyfikacja istniejącego skryptu

Po udokumentowaniu skryptu możemy wprowadzić odpowiednie modyfikacje. Dotychczas skrypt odczytywał nazwy trzech komputerów: Active computer, host i computer (te nazwy mogą się w pewnych sytuacjach różnić, więc skrypt ten jest naprawdę użyteczny). Przykłady innych informacji jakie moglibyśmy odczytywać za pomocą tego skryptu przedstawione są w tabeli 1-1. Należy zwrócić uwagę, klucze rejestru są tutaj rozpisane całkowicie – HKEY_LOCAL_MACHINE – a nie, jak we wcześniejszym skrypcie, skrótem *HKLM*. VBScript umożliwia odwoływanie się do rejestru na kilkanaście sposobów. Opisane są one w części poświęconej rejestrowi.

Tabela 1-1 Przydatne klucze rejestru dla piszących skrypty

Informacje	Lokalizacja
Informacje o usługach	HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services
Nazwa użytkownika do logowania na domenę	HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\Logon User Name
Informacje o domenie Microsoft Exchange 2000	HKEY_CURRENT_USER\Software\Microsoft\Exchange\LogonDomain
Informacje o użytkownikach domeny Exchange 2000	HKEY_CURRENT_USER\Software\Microsoft\Exchange\UserName
Serwer zasad grupy	HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Group Policy\History\DCName
Katalog domowy użytkownika	HKEY_CURRENT_USER\Volatile Environment\HomeShare
Serwer, który sprawdza autentyczność zalogowanego użytkownika	HKEY_CURRENT_USER\Volatile Environment\LOGONSERVER
Nazwa domeny DNS zalogowanego użytkownika	HKEY_CURRENT_USER\Volatile Environment\USERDNSDOMAIN

Uwaga Część informacji odczytywanych z rejestru może zostać uzyskana z innych źródeł, jak choćby poprzez Active Directory Service Interface (ADSI) lub Windows Management Instrumentation (WMI) (opis tych narzędzi znajduje się w dalszych rozdziałach). Należy pamiętać o tych możliwościach, gdyż rejestr jest środowiskiem dynamicznym i jego klucze mogą zmieniać pozycje. Oznacza to, iż rejestr nie zawsze jest spójny dla wszystkich maszyn w sieci. Bez wątpienia istnieją różnice pomiędzy Microsoft Windows 95 i Microsoft Windows XP, ale ten ostatni różni się też od Microsoft Windows 2000, a nawet od różnych wersji tego samego systemu. Korzystanie ze źródeł innych niż rejestr zapewni zatem znacznie spójniejsze wyniki. Odczyt z rejestru jest konieczny, gdy informacji nie sposób odczytać innymi metodami.

Odczyt informacji proponowanych w tabeli 1-1 wymaga dokonania zmian w każdej z czterech części skryptu. Znaczne jego fragmenty pozostaną jednak niezmienione, a różnice w poszczególnych częściach będą dotyczyć przede wszystkim nazw. Poniżej przedstawiono proces wprowadzania zmian:

Modyfikacja części nagłówkowej

Trzy pierwsze linie skryptu mogą pozostać bez zmian. Pozostawiamy *Option Explicit*, gdyż wciąż bardzo istotne są deklaracje zmiennych. Aby uniknąć błędów, wynikających z braku wartości lub innych niedopatrzeń, pozostawiamy także polecenie *On Error Resume Next*. Ze względu na odczyty z rejestru, potrzebna będzie zmienna *objShell*. Nie ma sensu zmiana tych nazw. Pierwotne nazwy zmiennych jasno określają ich zastosowanie. Konsekwentnie, korzystając z konkretnych nazw, rozwijamy własną konwencję nazewnictwa na potrzeby danych skryptów.

```
Option Explicit
On Error Resume Next
Dim objShell
```

Pierwsze trzy linie skryptu są w pełni funkcjonalne, należy więc zająć się deklarowaniem zmiennych do odczytu nowych wartości rejestru. W poniższym przykładzie zastosowanie znajdują niektóre z wartości z tabeli 1-1:

```
Dim regLogonUserName, regExchangeDomain, regGPServer
Dim regLogonServer, regDNSdomain
Dim LogonUserName, ExchangeDomain, GPServer
Dim LogonServer, DNSdomain
```

Wciąż wykorzystywana jest ta sama konwencja nazewnictwa z przedimkiem *reg* dla zmiennych przechowujących klucze rejestru oraz braku przedimka przed zmiennymi przechowującymi dane z kluczy rejestru (nazwy obiektów zmiennych różnią się jedynie przedimkiem).

Krok po kroku

▶ **Modyfikacja części nagłówkowej:**

1. Otworzyć Notatnik.
2. Upewnić się, że skrypt zawiera polecenie *Option Explicit*.
3. Upewnić się, że skrypt zawiera polecenie *On Error Resume Next*.
4. Usunąć niepotrzebne zmienne.
5. Dodać nowe zmienne.
6. Zachować skrypt pod inną nazwą.

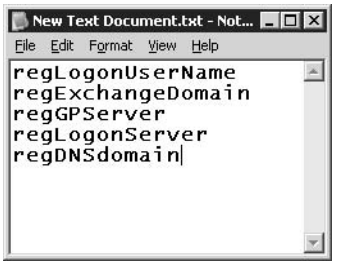
Modyfikacja części odniesieniowej

Ponieważ będziemy korzystać z innych kluczy rejestru, część odniesieniowa musi zostać całkowicie przebudowana. Jednak ogólna postać tej części pozostanie taka sama:

Nazwa zmiennej = Klucz rejestru w cudzysłowie
RegLogonUserName = „HKEY_CURRENT_USER\Software\Microsoft\„&_
„Windows\CurrentVersion\Explorer\Logon User Name”

W odczyt klucza rejestru zaangażowane są trzy części skryptu, ale każda z odczytanych informacji może zostać łatwo zmodyfikowana przez zmianę przydziału wartości do nazw zmiennych, wymienionych w powyższej propozycji składni. Ponieważ zadeklarowaliśmy wszystkie zmienne przeznaczone do przechowywania kluczy rejestru w części nagłówkowej, wystarczy je wyciąć i wkleić w części odniesieniowej. Następnie usuwamy fragmenty z poleceniem *Dim* wraz z przecinkami i ustawiamy każdą zmienną w oddzielnej linii. Po tych zmianach kod powinien wyglądać jak na rysunku 1-3.

```
Dim regLogonUserName, regExchangeDomain, regGPServer
Dim regLogonServer, regDNSdomain
```



Rysunek 1-3 Notatnik niezwykle usprawnia pisanie skryptów

Kiedy już nazwy zmiennych i znaki równości znajdą się na swoich miejscach, można dodać klucze rejestru i zawrzeć je w cudzysłowach. Warto pamiętać o przydatnej funkcji Copy Key Name w Regedit. Po wklejeniu wszystkich kluczy rejestru, część odniesieniowa powinna wyglądać jak

przykład przedstawiony poniżej. Znaki „&” i „_”, wykorzystane w tym kodzie, służą do wyznaczania kontynuacji linii, czyniąc skrypt bardziej przejrzystym. Dzięki takim połączeniom nie trzeba przewijać ekranu w prawo podczas przeglądania bardziej rozbudowanych skryptów.

```
regLogonUserName = "HKEY_CURRENT_USER\Software\Microsoft\" & _
    "Windows\CurrentVersion\Explorer\Logon User Name"
regExchangeDomain = "HKEY_CURRENT_USER\Software\Microsoft\" & _
    "Exchange\LogonDomain"
regGPServer = "HKEY_CURRENT_USER\Software\Microsoft\Windows\" & _
    "CurrentVersion\Group Policy\History\DCName"
regLogonServer = "HKEY_CURRENT_USER\Volatile Environment\" & _
    "LOGONSERVER"
regDNSdomain = "HKEY_CURRENT_USER\Volatile Environment\" & _
    "USERDNSDOMAIN"
```

Krok po kroku

▶ **Modyfikacja części nagłówkowej**

1. Otworzyć Notatnik.
2. Skopiować z części nagłówkowej fragment kodu z poleceniem Dim.
3. Wkleić skopiowany fragment z poleceniem *Dim* do nowego pliku w Notatniku.
4. Z menu edycji wybrać opcję Replace. W polu Find What, wpisać **Dim**. Pole Replace With pozostawić puste. W ten sposób wszystkie słowa *Dim* zostaną usunięte z kodu.
5. Każdą ze zmiennych umieścić w osobnej linii i usunąć przecinki.
6. Otworzyć Regedit i znaleźć odpowiednie klucze rejestru.
7. Korzystając z funkcji Copy Key Name, wkleić klucze po nazwach zmiennych.
8. Upewnić się, że nazwa zmiennej jest oddzielona od klucza rejestru znakiem równości.
9. Upewnić się, że klucze rejestru zawarte są w cudzysłowach.
10. Zachować skrypt.

Modyfikacja części wykonawczej

Pierwsza linia kodu części wykonawczej pozostaje niezmieniona.

```
Set objShell = CreateObject("WScript.Shell")
```

Wykorzystane tu zostają dwie, użyte już wcześniej w części nagłówkowej, zmienne. Kolejnym zadaniem jest modyfikacja każdej linii tak, aby przydzielała do nowych zmiennych z przedimkami *reg* nowoutworzone zmienne *bez* przedimków *reg*:

Nazwa zmiennej	=	Pracownik	Zmienna rejestru w ()
LogonUserName	=	*objShell.RegRead*	*(regLogonUserName)*

Poniżej przedstawiono część wykonawczą po wprowadzonych zmianach:

```
LogonUserName = objShell.RegRead(regLogonUserName)
ExchangeDomain= objShell.RegRead(regExchangeDomain)
GPServer = objShell.RegRead(regGPServer)
LogonServer = objShell.RegRead(regLogonServer)
DNSdomain = objShell.RegRead(regDNSdomain)
```

Wszystkie zmienne zadeklarowano w części nagłówkowej i stamtąd zostały skopiowane i wklejone do osobnych linii w części wykonawczej, z pominięciem fragmentów zawierających wyrażenia *Dim*. W kolejnej części skryptu umieszczamy znak równości i ten sam element wykonawczy (zawsze należy to robić), którym w tym wypadku jest *objShell.RegRead*. Ostatni fragment skryptu zawiera zmienną rejestru, zawartą w nawiasach w ramach części odniesieniowej. Znów wystarczy skorzystać z funkcji kopiowania i wklejania.

Krok po kroku

▶ **Modyfikacja części wykonawczej**

1. Otworzyć Notatnik.
2. Skopiować fragment kodu z poleceniem Dim z części nagłówkowej.
3. Wkleić skopiowany fragment kodu z poleceniem Dim do nowego pliku Notatnika.
4. Z menu edycji wybrać opcję Replace. W polu Find What wpisać Dim. Pole Replace With pozostawić puste. W ten sposób wszystkie słowa *Dim* zostaną usunięte z kodu.
5. Każdą ze zmiennych umieścić w osobnej linii i usunąć przecinki.
6. W każdej linii wkleić znak równości i element wykonawczy *objShell.RegRead*.
7. Wkleić odpowiednią zmienną z części odniesieniowej i umieścić ją w nawiasie.
8. Zapisać skrypt.

Uwaga Nie można nie docenić możliwości kopiowania i wklejania. Zapobiega ono występowaniu błędów w pisowni. Wystarczy jeden drobny błąd tego typu, aby sparaliżować funkcjonalność skryptu. Dlatego warto oszczędzić sobie zbędnych poprawek i w prosty sposób kopiować nazwy zmiennych z części nagłówkowej oraz wklejać je w części wykonawczej.

Po wprowadzeniu zmian w części wykonawczej skryptu, należy poświęcić dodatkową chwilę na sprawdzenie, czy wszystkie zmienne są na swoich miejscach, podobnie jak inne elementy kodu. Następnie zachowujemy skrypt pod inną nazwą, jeśli korzystaliśmy ze skryptu DisplayComputerNames. Próby uruchomienia skryptu na tym etapie skazane są na niepowodzenie, gdyż przed wprowadzeniem zmian w ostatniej części skryptu nie zadziała on poprawnie.

Modyfikacja części wyjściowej

Część wyjściowa ma za zadanie wyświetlić odczytane z rejestru dane. Dokładniejsze omówienie wyświetlania i zapisu danych znajduje się w kolejnych rozdziałach. Tymczasem skupimy się na poleceniu *WScript.Echo.*

Niestety nie możemy wykorzystać części wyjściowej poprzedniego skryptu. Byłoby to zbyt pracochłonne, ze względu na konieczną ilość zmian. Zostawiamy jedynie linie z poleceniem *WScript.Echo*. Wszystko, co następuje po *WScript.Echo,* powinno zostać usunięte. W odpowiednie miejsca wklejamy nazwy wszystkich zmiennych z części wykonawczej. Składnia tej części powinna wyglądać, jak w tabeli:

Polecenie	Zmienna	&	Komentarz
WScript.Echo	*LogonUserName*	&	*" jest obecnie zalogowany"*

Nietrudno dostrzec, iż w kolumnie komentarza po pierwszym cudzysłowie następuje spacja. Przerwa potrzebna jest ze względu na znak „&", łączący dwa wyrażenia. VBScript nie tworzy przerw pomiędzy łączonymi wyrażeniami. Zmodyfikowany kod wygląda następująco:

```
WScript.Echo LogonUserName & " is currently Logged on"
WScript.Echo ExchangeDomain & " is the current logon domain"
WScript.Echo GPServer & " is the current Group Policy Server"
WScript.Echo LogonServer & " is the current logon server"
WScript.Echo DNSdomain & " is the current DNS domain"
```

Aby połączyć wszystkie te linie w całość, wystarczy wyciąć i wkleić każdą zmienną przyporządkowaną kluczowi rejestru w części wykonawczej skryptu, dodać znak „&" i umieścić w cudzysłowie tekst, objęty działaniem polecenia *WScript.Echo*. W późniejszych ćwiczeniach będzie ono wykorzystywane do zawiadamiania piszącego skrypty o problemach.

Krok po kroku

▶ **Modyfikacja części wyjściowej**

1. Otworzyć Notatnik.
2. Skopiować wszystkie zmienne dodane do części wykonawczej.
3. Wkleić skopiowane zmienne do części wyjściowej.
4. Dodać znak „&" po każdej zmiennej.
5. Zamknąć cudzysłowem tekst, który ma zostać wyświetlony na ekranie.
6. Wkleić znak równości i element wykonawczy *objShell.RegRead* w każdej linii.
7. Każdą linię poprzedzić poleceniem *WScript.Echo*.
8. Zachować skrypt.

Uruchamianie skryptów

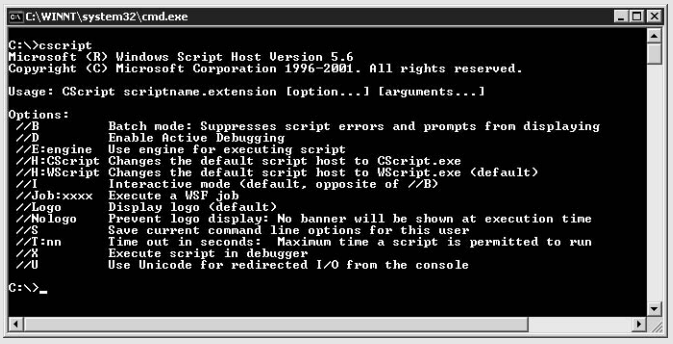
Skrypty można uruchamiać w Windows Server 2003 na kilkanaście sposobów. Zostaną one omówione w kolejnych punktach.

Dwukrotne kliknięcie pliku z rozszerzeniem .vbs

Domyślnie dwukrotne kliknięcie na plik z rozszerzeniem .vbs uruchamia plik poprzez WScript.exe. W związku z tym, konsekwencją umieszczenia polecenia *WScript.Echo* w części wyjściowej skryptu są pojawiające się okna. Dopóki jest ich niewiele, są odpowiednim rozwiązaniem. Jednak nie jest to optymalne rozwiązanie dla domeny z tysiącami użytkowników. Lepszą alternatywą okazuje się metoda CScript.

CScript

CScript jest czymś w rodzaju linii poleceń Windows Scripting Host (rysunek 1-4). W przeciwieństwie do domyślnej obsługi Windows Scripting Host, nie wymaga pracy z oknami (w domyślnej obsłudze Windows Scripting Host praca skryptu zostaje wstrzymana do czasu kliknięcia OK w odpowiednim oknie i wszystkich kolejnych). Aktywowany przez okno poleceń CScript, tryb Quick Edit umożliwia prosty podgląd wyników pracy. Uruchamia się go klikając C:\ w lewym, górnym rogu okna i wybierając opcję Properties z menu akcji. Następnie klikamy zakładkę Options i wybieramy Quick Edit Mode, a potem Save Properties For Future Windows of The Same Title. Umożliwia to zaznaczanie i kopiowanie tekstu z okna linii poleceń. Ze skopiowanym tekstem możemy zrobić wszystko, od wklejenia go do Notatnika, po sortowanie skopiowanych danych w Microsoft Excel. Możliwości te omówiono w dalszych rozdziałach książki.



Rysunek 1-4 CScript udostępnia opcje poprzez linię poleceń

Umieszczanie skryptów na stronach internetowych

Skrypty można umieszczać na stronach internetowych. Ma to zastosowanie w przedsiębiorstwach, gdzie użytkownicy mający dostęp do strony internetowej w intranecie powinni również mieć możliwość uruchomienia z niej skryptu. Taki sposób wykorzystania VBScript może okazać się użyteczny w wypadku gromadzenia informacji lub pomocy dla użytkowników. Ma on jednak swoje wady, które zostaną omówione w późniejszych rozdziałach książki.

Przeciąganie i upuszczanie pliku .vbs na otwarte okno linii poleceń

Plik .vbs można przeciągnąć i upuścić na otwarte okno linii poleceń, uruchamiając w ten sposób skrypt z hostem domyślnym. Metoda ta nie wymaga wpisywania ścieżki pliku, ponieważ Windows Explorer automatycznie umieszcza ją w linii poleceń.

Przeciąganie i upuszczanie pliku .vbs w Notatniku

Plik .vbs można przeciągnąć i upuścić na otwarte okno Notatnika, otwierając w ten sposób przeciągany plik i wyświetlając jego zawartość.

Dodawanie Notatnika do menu SendTo

Szybką edycję skryptu można przeprowadzić w Notatniku. Wystarczy dodać Notatnik do menu SendTo, umieszczając skrót do Notepad.exe w katalogu C:\Documents and Settings\%USERNAME%\SendTo.

Podsumowanie

W rozdziale pierwszym skupiliśmy się na analizie pierwszego skryptu. Trzeba zapamiętać, że skrypt składa się z czterech części: nagłówkowej, odniesieniowej, wykonawczej i wyjściowej. Rozdział zawiera także przegląd zmiennych i metod ich deklarowania. Ponadto omawiliśmy sposoby odczytu rejestru i informacji od użytkowników. Pokazano także modyfikację skryptów, wykorzystując do tego celu ich wcześniej napisane fragmenty oraz elementy zupełnie nowe.

Quiz

P. W jakim celu umieszcza się polecenie *Option Explicit* w pierwszej linii VBScript?

O. Polecenie *Option Explicit* wymusza deklarację wszystkich zmiennych, wykorzystanych w skrypcie. Polecenie to jest wyjątkowo pomocne, jako że zapobiega błędom związanym z literówkami oraz jasno pokazuje, które słowa w skrypcie są poleceniami, a które nazwami zmiennych. Brak polecenia *Option Explicit* powoduje, iż błędnie wpisana zmienna automatycznie staje się nową zmienną.

P. Próba odczytu rejestru kończy się błędem skryptu, gdy próbuje on odczytać konkretny klucz. Jak można temu zapobiec?

O. Istnieją dwa sposoby zapobiegania zatrzymaniom skryptu w wyniku wystąpienia błędu. Pierwszy to zawarcie w skrypcie polecenia *On Error Resume Next*. Drugim jest umieszczenie przed linią zawierającą błąd apostrofu. Dalsze poprawne działanie skryptu udowodni, że to właśnie ta linia zawierała błąd.

P. Jaki związek ma polecenie *Dim* ze zmienną?

O. Poleceniem *Dim* deklarujemy zmienne. Deklaracja wywołuje przydzielenie zmiennej miejsca w pamięci przez system operacyjny. W ramach polecenia *Dim* warto zawrzeć dokumentację, wyjaśniającą zastosowanie zmiennych.

P. Którego polecenia należy użyć do otwarcia okna dialogowego?
O. Polecenie *WScript.Echo* otwiera okno dialogowe w WScrpit lub wysyła dane wyjściowe do okna konsoli w CScript.

P. Z jakich części składa się VBScript?
O. Z czterech części: nagłówkowej, odniesieniowej, części wykonawczej i wyjściowej.

Do samodzielnego ćwiczenia

Ćwiczenie 1: Poznanie VBScript

W tym punkcie poznamy VBScript.

Instrukcje

1. Otwieramy katalog Lab 1, znajdujący się na dysku CD dołączonym do książki. Z katalogu wybieramy skrypt DisplayComputerNames.vbs i otwieramy go w Notatniku.
2. Dodajemy komentarze opisujące każdą cześć skryptu (umieszczamy komentarze do wszystkich części skryptu: nagłówkowej, odniesieniowej, części wykonawczej i wyjściowej).
3. Zachowujemy skrypt pod inną nazwą, na przykład lab1.vbs.
4. Usuwamy całą cześć nagłówkową.
5. Zachowujemy skrypt i spróbujemy go uruchomić. Czy się udaje?
6. Dodajemy polecenie *Option Explicit* i znów zachowujemy. A teraz?
7. Umieszczamy znak komentarza, czyli apostrof, przed *Option Explicit* i zachowujemy plik. Czy uruchamia się?

Ćwiczenie 2: Adaptacja skryptu

Ćwiczenie 2 opisuje sposoby adaptacji skryptu do własnych potrzeb.

Scenariusz

Pracujemy jako administrator sieciowy w firmie Fortune 500. Niedawno miała miejsce awaria serwera, a ten nie wygenerował pliku zrzutowego. Jako że w sieci znajduje się kilkanaście serwerów, zależy nam na ograniczeniu prób odtwarzania konfiguracji po awarii do wykorzystania prostego skryptu. Nie potrafimy co prawda jeszcze zdalnie uruchomić skryptu, ale na szczęście mamy do pomocy stażystę. W związku z tym, obaj wykonujemy co następuje:

1. Tworzymy skrypt odczytujący odtworzone po awarii dane z rejestru. Interesują nas poniższe klucze:

```
"HKLM\SYSTEM\CurrentControlSet\Control\CrashControl\AutoReboot"
"HKLM\SYSTEM\CurrentControlSet\Control\CrashControl\MinidumpDir"
"HKLM\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters\Hostname"
"HKLM\SYSTEM\CurrentControlSet\Control\CrashControl\LogEvent"
"HKLM\SYSTEM\CurrentControlSet\Control\CrashControl\DumpFile"
```

2. Kopiujemy skrypt do lokalnego serwera.
3. Uruchamiamy skrypt pod CScript.
4. Niech pomocnik skopiuje dane wyjściowe z linii poleceń i wklei je do pliku w Notatniku, która ma taką samą nazwę, jak nazwa serwera.

Instrukcje

1. Otwieramy katalog Lab2, znajdujący się na dołączonej dysku CD.
2. Otwieramy plik DisplayComputerName.vbs i zachowujemy go jako ReadCrashRecoveryInformation.vbs.
3. Modyfikujemy część nagłówkową skryptu, dodając zmienne dla wszystkich elementów odczytywanych z rejestru (pamiętajmy, że na każdy z nich przydają dwie zmienne: jeden bezpośrednio dla klucza rejestru, a drugi dla danych zawartych w kluczu).
4. Modyfikujemy część odniesieniową skryptu (zmienne z przedrostkiem reg utworzone w poprzednim punkcie przypisujemy do odpowiednich kluczy rejestru).
5. Modyfikujemy część wykonawczą skryptu (pozostałe zmienne utworzone w trzecim punkcie przypisujemy do regRead w części wykonawczej skryptu).
6. Modyfikujemy część wyjściową skryptu (korzystamy ze zmiennych przypisywanych w poprzednim punkcie).
7. Jeśli zachodzi taka potrzeba, dodajemy do skryptu odpowiednią dokumentację. Powinno się wręcz przesadzać ze szczegółowością i liczbą komentarzy. Rozwiązania, które są dziś jasne, za parę dni mogą wydać się kompletnie niezrozumiałe.
8. Zachowujemy skrypt.
9. Otwieramy linię poleceń.
10. Wpisujemy CScript ReadCrashRecoveryInformation.vbs i naciskamy Enter. W razie nieodnalezienia pliku, zmieniamy katalog na ten, w którym zachowaliśmy skrypt i powtarzamy czynności.

3 Inteligencja

Nie wszyskie operacje mogą zostać zautomatyzowane – dotyczy to też uaktualniania sieciowych systemów operacyjnych. Jednak możliwa jest obsługa za pomocą skryptów takich zadań, jak odczyt wpisów zdarzeń i reagowanie na wydarzenia krytczyne. Trzeci rodział będzie opierać się na wiedzy z dwóch poprzednich rozdziałów, połączonej z trzema potężnymi narzędziami: *If…Then*, *If…Then…ElseIf* i *Select Case*.

Zanim rozpoczniesz

Do zrozumienia zawartości tego rozdziału wymagana jest wiedza z zakresu:

- Deklaracji zmiennych
- Podstawowej obsługi błędów
- Nawiązywania połączenia z obiektem systemu plików
- Korzystania z *For Each…Next*
- Korzystania z *Do While*

Po ukończeniu tego rozdziału, Czytelnik będzie rozumiał funkcje:

- *If…Then*
- *If…Then…ElseIf*
- *If…Then…Else*
- *Select Case*
- Stałe wewnętrzne (wbudowane)

If...Then

If…Then to jedna z podstawowych funkcji programistycznych. Jej działanie opiera się na zasadach logiki. Przykład podstawowej operacji:

If	warunek	Then	czynność
If	sklep otwarty	Then	kup kurczaka

Prawdziwe możliwości *If…Then* (jeżeli, to) ujawnią się w powiązaniu z narzędziami, poznanymi w rozdziale 2. Tym bardziej, że funkcja ta rzadko występuje samodzielnie. Przykładem samotnego jej zaistnienia jest skrypt:

```
Const a = 2
Const b = 3
```

```
Const c = 5
If a + b = c Then
    WScript.Echo(c)
End If
```

Rozpoczyna się on definiowaniem trzech stałych: a, b, i c, które następnie zostały zsumowane, a ich wynik oceniony funkcją *If… Then*. Implementując wyrażenie, należy zwrócić uwagę na trzy rzeczy:

- *If* i *Then* muszą być w tej samej linii.
- Wykonywane polecenie musi się znaleźć w linii następnej.
- Wyrażenie *If… Then* musi kończyć się *End If*.

Brak lub błędne umieszczenie jednego z powyższych elementów zaoowocuje błędem. Jeśli komunikat o błędzie nie pojawia się, choć brakuje jednego z elementów, należy upewnić się czy polecenie *On Error Resume Next* zostało na pewno wykomentowane.

Kolejny, bardziej skomplikowany i użyteczny, skrypt nosi nazwę GetComments.vbs. Znajduje się na załączonym dysku CD. Jego działanie polega na zapisie komentarzy VBScript do osobnego pliku, tworząc w ten sposób oddzielną dokumentację. Jeśli wszystkie komentarze będą przygotowane według ustalonego schematu, wygenerowanej dokumentacji nie trzeba będzie prawie poprawiać:

```
Option Explicit
On Error Resume Next
Const ForReading = 1
Const ForWriting = 2
Dim scriptFile
Dim commentFile
Dim objScriptFile
Dim objFSO
Dim objCommentFile
Dim strCurrentLine
Dim intIsComment
scriptFile = "C:\scripts\displayComputerNames.vbs"
commentFile = "C:\scripts\comments.txt"
Set objFSO = CreateObject("Scripting.FileSystemObject")
Set objScriptFile = objFSO.OpenTextFile _
    (scriptFile, ForReading)
Set objCommentFile = objFSO.OpenTextFile(commentFile, _
    ForWriting, True)
Do While objScriptFile.AtEndOfStream <> True
    strCurrentLine = objScriptFile.ReadLine
    intIsComment = InAtr(1,strCurrentLine,"'")
    If intIsComment > 0 Then
        objCommentFile.Write strCurrentLine & vbCrLf
    End If
Loop
```

```
WScript.Echo("script complete")
objScriptFile.Close
objCommentFile.Close
```

Krok po kroku

▶ Implementacja *If...Then:*

1. W nowej linii skryptu wpisać **If** *warunek* **Then**.
2. W kolejnej linii umieścić polecenie do wykonania.
3. W kolejnej linii wpisać **End If**.

Część nagłówkowa

Część nagłówkowa skryptu wykorzystuje polecenie *Option Explicit*, wymuszające deklarację zmiennych i strzegące przed literówkami w nazwach zmiennych oraz polecenie *On Error Resume Next,* zapewniające podstawową obsługę błędów. Polega ona na pominięciu linii, w której znajduje się błąd. Nie zawsze okazuje się to pomocne. Jest tak w wypadku operacji kopiowania pliku do innej lokalizacji a następnie usunięcia orginału. Może to zakończyć się usunięciem pliku bez wcześniejszego skopiowania go.

Trzecia oraz czwarta linia skryptu GetComments.vbs definiuje dwie stałe, *ForReading* i *ForWriting*, służące przejrzystości skryptu (stałe omówiono w rozdziale 2). Zostaną one wykorzystane po otwarciu pliku DisplayComputerNames.vbs i Comments.txt. Zamiast stałych można bezpośrednio w poleceniu umieścić liczby 1 i 2, jednak skrypt traci wówczas na przejrzystości. Ponadto dzięki stałym późniejsze modyfikacje są łatwiejsze do wprowadzenia.

Po definiowaniu dwóch stałych zaczynamy definiować zmienne. Umieszczanie każdej zmiennej w osobnej linii ułatwia dodawanie do nich poszczególnych komentarzy, informujących o zastosowaniu danej zmiennej. Tak naprawdę nie ma konieczności definiowania zmiennych w konkretnym miejscu kodu, ponieważ kompilator przegląda zawartość skryptu przed jego uruchomieniem. Choć chaos w kodzie nie ma wpływu na działanie skryptu, to po prostu utrudnia on jego odczyt.

Część odniesieniowa

Część odniesieniowa skryptu zawiera przypisanie wartości do zadeklarowanych wcześniej zmiennych. Zmienna *scriptFile* ułatwia późniejsze modyfikacje skryptu, jako że może zostać przypisana do pojedynczego pliku albo może odczytywać po kolei skrypty z wybranego katalogu. Dodatkowo można zaopatrzyć skrypt w linię poleceń do wpisania nazwy skryptu, z którego mają zostać wyodrębnione komentarze. Przypisanie zmiennej do nazwy pliku skryptowego umożliwia wprowadzenie powyższych rozwiązań w krótkim czasie. W tym miejscu nadajemy także nazwę plikowi przechowującemu zebrane komentarze. Niech nosi on nazwę Comments.txt.

W kodzie trzykrotnie pojawia się polecenie *Set*:

```
Set objFSO = CreateObject("Scripting.FileSystemObject")
Set objScriptFile = objFSO.OpenTextFile _
```

```
    (scriptFile, ForReading)
Set objCommentFile = objFSO.OpenTextFile(commentFile, _
    ForWriting, True)
```

W pierwszej linii, w powiązaniu z poleceniem *Set,* pojawia się znana zmienna *objFSO*. Przypisujemy ją do połączenia z obiektem systemu plików, który umożliwia odczyt i zapis plików. Obiekt systemu plików musi zostać stworzony (określenie techniczne), aby można było otwierać pliki tekstowe.

Drugie polecenie *Set* wykorzystuje zmienną *objScriptFile* do odczytu pliku DisplayComputerNames.vbs. Należy pamiętać, że polecenie *OpenTextFile* wymaga wyłącznie jednej informacji: nazwy pliku. Jeśli nie zamieścimy dodatkowych informacji o trybie dostępu, VBScript uzna, że został on otwarty do odczytu. Określamy dwa bity danych upraszczające skrypt:

- Nazwa pliku
- Zastosowanie pliku – odczyt lub zapis

Wykorzystanie tych dwóch zmiennych przy poleceniu *OpenTextFile* czyni skrypt bardziej przejrzystym i elastycznym.

Trzecie polecenie *Set* działa podobnie jak poprzednie. Zmienna *objCommentFile* zostaje przypisana do wyników zwróconych przez polecenie *openTextFile*. W tym wypadku jednak, odbywa się zapis, a nie odczyt pliku. Wykorzystano tu zmienne dla nazwy pliku z komentarzami oraz dla opcji zapisu pliku.

Krótko

P. Czy dozwolone jest, aby *If* znajdowało się w jednej linii a *Then* w następnej?
O. Nie. Obie części wyrażenia muszą być w tej samej linii logicznej. Mogą one znajdować się w osobnych liniach fizycznych, jeśli linie te są połączone symbolem _. Zazwyczaj *If* jest słowem rozpoczynającym linię, a *Then* poleceniem, które ją zamyka.

P. Jeśli klauzula *Then* znajduje się w osobnej linii logicznej wobec wyrażenia *If*... *Then*, jakie polecenie powinno pojawić się dalej?
O. *End If*. Należy pamiętać, że *End If* składa się z dwóch słów.

P. Jaka jest główna przyczyna wykorzystania stałych?
O. Wartości stałych zostają ustalone przed uruchomieniem skryptu i w czasie jego pracy pozostają niezmienne.

P. Jakich dwóch informacji wymaga polecenie *OpenTextFile*?
O. *OpenTextFile* wymaga nazwy pliku i określenia, czy ma być on zapisany, czy odczytany.

Część wykonawcza i wyjściowa

Część wykonawcza i wyjściowa skryptu są jego jądrem, w którym wykonuje się polecenia skryptu. Skrypt GetComments.vbs odczytuje każdą linię pliku DisplayComputerNames,

sprawdzając, czy nie znajduje się w niej apostrof: "'". Jeśli odnaleziona zostanie linia z apostrofem, skrypt zapisuje ją w pliku comments.txt.

Część wykonawcza i wyjściowa skryptu GetComments.vbs rozpoczyna się od wyrażenia *Do While…Loop*:

```
Do While objScriptFile.AtEndOfStream <> True
    strCurrentLine = objScriptFile.ReadLine
    intIsComment = InStr(1,strCurrentLine,"'")
    If intIsComment > 0 Then
        objCommentFile.Write strCurrentLine & vbCrLf
    End If
Loop
WScript.Echo("script complete")
objScriptFile.Close
objCommentFile.Close
```

Wyrażenie *Do While* zostało omówione w rozdziale 2. Zmienna *ObjScriptFile* przechowuje tekst co znaczy, że do czasu dotarcia do końca strumienia tekstu, każda kolejna linia jest przeszukiwana pod kątem obecności apostrofu. Test na obecność symbolu wykonuje, podobnie jak w poprzednim rozdziale, funkcja *InStr*. Funkcja *InStr* zwraca zero lub wartość *null* w razie nieodnalezienia apostrofu albo też liczbową pozycję, jeżeli go nie odkryje.

Jeśli *InStr* odnajdzie apostrof w linii tekstu, wartość zmiennej *intIsComment* staje się większa niż zero. Wobec tego zapis linii w pliku comments.txt odbywa się za pośrednictwem wyrażenia *If…Then*:

```
If intIsComment > 0 Then
    objCommentFile.Write strCurrentLine & vbCrLf
End If
```

Rozaptrywany warunek zawarty jest w wyrażeniu *If…Then*. Jeśli zmienna *intIsComment* ma wartość większą od zera, wybrana linia tekstu z pliku DisplayComputerNames zostaje zapisana przy użyciu polecenia *Write*.

Stałe wewnętrzne (wbudowane)

Polecenie *vbCrLf* wykonuje operację zwaną *powrót karetki* i *wysuw wiersza*; *vbCrLf* jest stałą wewnętrzną, czyli wbudowaną w VBScript. W związku z tym nie ma konieczności definiowania jej, jak regularnej stałej. W bardziej rozwiniętych skryptach stałe wewnętrzne będą wykorzystywane wielokrotnie.

vbCrLf ma swe korzenie w mechanice maszyny do pisania. Jej działanie polegało na przeskoku do kolejnego wiersza (wysuw wiersza) oraz zmianie pozycji głowicy (powrót karetki). Podobnie jak maszyna do pisania, polecenie *vbCrLf* przeskakuje na początek kolejnej linii. Jest ono przydatne przy formatowaniu tekstu, zarówno w oknach dialogowych, jak i plikach tekstowych. Ostatnia linia w ramach wyrażenia *If…Then* zawiera polecenie *End If*. *End If* zawiadamia VBScript o zakończeniu pracy polecenia *If…Then*. Jeśli *End If* nie pojawi się w kodzie, VBScript zakomunikuje błąd.

Po *End If* następuje osobna linia z pojedynczym poleceniem *Loop*. Polecenie *Loop* należy do wyrażenia *Do While*, które zainicjowało część wykonawczą i wyjściową. *Loop* wskazuje

skryptowi ponownie linię z *Do While*. VBScript kontynuuje pracę w pętli, odczytując plik tekstowy w poszukiwaniu apostrofów, dopóki nie osiągnie końca strumienia tekstu. Kiedy dotrze do końca strumienia tekstu ze skryptu DisplayComputerNames, na ekranie pojawi się komunikat o zakończeniu przeszukiwania. Jest on istotny, ponieważ nic innego nie wskaże, iż skrypt zakończył pracę. Następnie zamykamy te dwa pliki. Tak naprawdę nie ma potrzeby zamykania ich manualnie, gdyż po zakończeniu skryptu dzieje się to automatycznie. Jednak w czasie pracy lepiej robić to samodzielnie na wypadek jakichkolwiek problemów.

If...Then...ElseIf

If… Then umożliwia sprawdzenie jednego warunku i podjęcie czynności w zależności od tego warunku. Dodanie *ElseIf* pozwala na wyprowadzenie wielu rozwiązań. Dzieje się to w podobny sposób jak przy poleceniu *If… Then*. Rozpoczynamy od *If… Then* w pierwszej linii części wykonawczej. Kończymy wyrażenie *If… Then* za pomocą *End If*. Jeśli chcemy dodać kolejne warunki do sprawdzenia, umieszczamy w nowej linii polecenie *ElseIf* oraz wybrany warunek.

Krok po kroku

▶ **Stosowanie *If...Then...ElseIf***

1. W nowej linii skryptu wpisać **If** *warunek* **Then**.
2. W kolejnej linii umieścić polecenie, które ma zostać wykonane.
3. W następnej linii wpisać *ElseIf* oraz drugi warunek do sprawdzenia. Linię zakończyć słowem *Then*.
4. W kolejnej linii umieścić polecenie do wykonania po spełnieniu warunku *ElseIf*.
5. W razie potrzeby, powtórzyć odpowiednio kroki 3 i 4.
6. W następnej linii wpisać **End If**.

Liczba linii z *ElseIf* może być dowolna, jednak ich nadmiar nie służy funkcjonalności skryptu. Optymalnym rozwiązaniem, zapobiegającym nadmiernemu wydłużaniu skryptu, jest konwersja do typu strukturalnego *Select Case*, omówionego w jednym z kolejnych punktów tego rozdziału. Działanie wyrażenia *If… Then…ElseIf* zilustruje skrypt CPUType.vbs, identyfikujący typ procesora maszyny. Zazwyczaj jest to x86, ale zdarzają się także stare maszyny Alpha, Power PC, a nawet IA64. Precyzyjne informacje o typach procesorów mogą zapobiec problemom, wynikającym ze zdalnego sterowania maszynami. Skrypt CPUType.vbs wygląda następująco:

```
Option Explicit
On Error Resume Next
Dim strComputer
Dim cpu
Dim wmiRoot
Dim objWMIService
Dim ObjProcessor
```

```
strComputer = "."
cpu = "win32_Processor='CPU0'"
wmiRoot = "winmgmts:\\" & strComputer & "\root\cimv2"
Set objWMIService = GetObject(wmiRoot)
Set objProcessor = objWMIService.Get(cpu)
If objProcessor.Architecture = 0 Then
    WScript.Echo "This is an x86 cpu."
ElseIf objProcessor.Architecture = 1 Then
    WScript.Echo "This is a MIPS cpu."
ElseIf objProcessor.Architecture = 2 Then
    WScript.Echo "This is an Alpha cpu."
ElseIf objProcessor.Architecture = 3 Then
    WScript.Echo "This is a PowerPC cpu."
ElseIf objProcessor.Architecture = 6 Then
    WScript.Echo "This is an ia64 cpu."
Else
    WScript.Echo "Can not determine cpu type."
End If
```

Część nagłówkowa

Część nagłówkowa zawiera typowe dane (omówione w rozdziałach 1 i 2):

```
Option Explicit
On Error Resume Next
Dim strComputer
Dim cpu
Dim wmiRoot
Dim objWMIService
Dim objProcessor
```

Option Explicit wymusza deklarację zmiennych poleceniami *Dim*. *On Error Resume Next* włącza standardową obsługę błędów. Zmienna *strComputer* przechowuje nazwę komputera, którego typ procesora chcemy ustalić. Zmienna *Cpu* informuje VBScript, z którego miejsca Windows Management Instrumentation (WMI) odczytujemy dane. Zmienna *wmiRoot* umożliwia wykonywanie takich zadań jak oddzielenie części połączenia WMI, czyniąc ją łatwiejszą do modyfikacji i odczytu. Zmienne *objWMIService* i *objProcessor* przechowują informacje z części odniesieniowej skryptu.

Część odniesieniowa

Część odniesieniowa zawiera przypisania wartości do zadeklarowanych wcześniej zmiennych. W skrypcie CPUType.vbs istnieją następujące przypisania:

```
strComputer = "."
cpu = "win32_Processor='CPU0'"
wmiRoot = "winmgmts:\\" & strComputer & "\root\cimv2"
```

```
Set objWMIService = GetObject(wmiRoot)
Set objProcessor = objWMIService.Get(cpu)
```

StrComputer jest równoważny symbolowi ".", który oznacza lokalny komputer, na którym wykonywany jest skrypt. Za pomocą zmiennej *cpu* definiujemy miejsce przechowywania przez WMI informacji o procesorach, czyli zmienną *win32_Processor*. Jako że maszyna może korzystać z więcej niż jednego procesor, zapytanie musi zostać ograniczone do *CPU0*. Nie może być to *CPU1*, gdyż *win32_Processor* rozpoczyna liczenie procesorów od 0, a każdy komputer zawsze zawiera CPU0, w przeciwieństwie do CPU1. Skrypt ten służy do identyfikacji typu procesora maszyny, a zatem nie ma konieczności ropoznania wszystkich procesorów w komputerze.

Część wykonawcza i wyjściowa

Pierwsza część skryptu zawiera deklaracje zmiennych, a druga przypisanie im wartości. Kolejna część wykorzystuje zmienne w wyrażeniu *If… Then…ElseIf*, które rozpoznaje typ procesora.

Poniżej przedstawiono część wykonawczą iwyjściową skryptu CPUType.vbs:

```
If objProcessor.Architecture = 0 Then
    WScript.Echo "This is an x86 cpu."
ElseIf objProcessor.Architecture = 1 Then
    WScript.Echo "This is a MIPS cpu."
ElseIf objProcessor.Architecture = 2 Then
    WScript.Echo "This is an Alpha cpu."
ElseIf objProcessor.Architecture = 3 Then
    WScript.Echo "This is a PowerPC cpu."
ElseIf objProcessor.Architecture = 6 Then
    WScript.Echo "This is an ia64 cpu."
Else
    WScript.Echo "Can not determine cpu type."
End If
```

Napisanie takiego skryptu wymaga wiedzy na temat zwrotu informacji przez *win32_Processor*. Umieszczenie tych informacji w konstrukcji *If… Then…ElseIf* umożliwi ich translację na przydatne w tym wypadku dane.

Pierwsze dwie linie powyższego skryptu funkcjonują, jak typowe wyrażenie *If… Then*. Linia rozpoczyna się słowem *If* a kończy słowem *Then*. Wewnątrz konstrukcji *If… Then* znajduje się wyrażenie przeznaczone do sprawdzenia. Jeśli *objProcessor* zwróci zero, gdy zapytanie dotyczy *Architecture*, będzie to oznaczać, iż procesor należy do typu x86. Wniosek zostaje wyświetlony za pomocą polecenia *WScript.Echo*.

Jeśli natomiast *objProcessor* zwróci *1*, oznacza to, że procesor należy do typu MIPS. Dodając do wyrażeń *ElseIf* wyniki zwracanych przez WMI kodów o typach procesorów, umożliwiamy skryptowi ustalenie typu procesora, jakim dysponuje serwer. Po wykorzystaniu odpowiedniej ilości wyrażeń *ElseIf* do sprawdzenia wszystkich kodów, umieszczamy do obsługi niewykorzystanych kodów dodatkową linię, zwierającą polecenie *Else*.

Krótko

P. **Ile linii *ElseIf* można umieścić w skrypcie?**
O. Tyle ile potrzeba.

P. **Czy jeśli potrzebne są więcej niż dwa wyrażenia *ElseIf* istnieje prostsza alternatywa?**
O. Tak. Zastosowanie typu strukturalnego *Select Case*.

P. **Jaki będzie efekt zastosowania w skrypcie *strComputer* = "."?**
O. Kod *strComputer* oznacza lokalny komputer, na którym wykonywany jest skrypt, co znajduje zastosowanie w WMI.

If...Then...Else

If…Then…Else można stosować bez dodatkowych poleceń *ElseIf*. Wówczas skrypt ma do wyboru dwie opcje.

Krok po kroku

▶ **Zastosowanie *If...Then...Else:***

1. W nowej linii skryptu wpisać **If** *warunek* **Then**.
2. W kolejnej linii umieścić polecenie do wykonania.
3. W następnej linii wpisać Else.
4. W kolejnej linii umieścić alternatywne polecenie wykonywane, jeśli warunek nie zostaje spełniony.
5. W ostatniej linii wpisać **End If**.

Zastosowanie *If…Then…Else* przedstawiono w poniższym kodzie:

```
Option Explicit
On Error Resume Next
Dim a,b,c,d
a = 1
b = 2
c = 3
d = 4
If a + b = d Then
    WScript.Echo (a & " + " & b & " is equal to " & d)
Else
    WScript.Echo (a & " + " & b & " is equal to " & c)
End If
```

Zapisany skrypt IfThenElse.vbs zawiera deklarację wszystkich zmiennych w jednej linii. Metodę tę można stosować przy prostych skryptach. Nadaje się ona także do rutynowych zmiennych, powiązanych ze sobą nawzajem jak w wypadku *objWMIService* i *objProcessor* z poprzedniego skryptu. Zaletą wielokrotnych deklaracji w jednej linii jest oszczędność miejsca. Nie ma to wpływu na wydajność skryptu, ale czasem wpływa na jego czytelność. Czytelnik sam będzie musiał sobie odpowiedzieć na pytanie, która forma jest dla niego wygodniejsza: wielokrotna deklaracja w jednej linii, czy deklarowanie jednej zmiennej co linię?

Zastosowanie polecenia *WScript.Echo* wiąże się ze zjawiskiem zwanym konkatenacją, która łączy treść wyjściową, będącą kombinacją zmiennych i łańcuchów tekstu. Należy zauważyć, że wszystkie elementy zawarte są w nawiasach, a żadna zmienna nie jest objęta cudzysłowiami. Konkatenacja w jedną linię zachodzi przez zastosowanie znaku (&). Jako że konkatenacja nie dodaje automatycznie odstępów, należy w wybranych miejscach, wewnątrz cudzysłowów, dodać spacje. Można w ten sposób w danych wyjściowych zawrzeć dodatkowe informacje. Podczas modyfikacji skryptu, należy zwracać baczną uwagę na tę część kodu. Zmiany w kodzie mogą polegać na prostych modyfikacjach wartości zmiennych, ale jednak wpisanie nowego łańcucha tekstowego zajmuje chwilę.

Select Case

Wyrażenie *Select Case* jest zazwyczaj niedostępne dla początkujących. Radzą sobie oni z *If… Then*, a nawet *If… Then… Else*, ale tylko nieliczni umiejętnie korzystają z konstrukcji *Select Case*. Jest to spora strata, gdyż wyrażenie to ma naprawdę wiele możliwości zastosowania. W związku z tym, zostanie ono dokładnie omówione w kolejnych punktach.

Krok po kroku

▶ **Zastosowanie *Select Case:***

1. W nowej linii skryptu wpisać **Select Case** i zmienną do sprawdzenia.
2. W kolejnej linii wpisać **Case 0**.
3. W następnej linii przypisać zmiennej wartość.
4. W kolejnej linii wpisać Case 1.
5. W następnej linii przypisać zmiennej wartość.
6. W kolejnej linii wpisać **End Select**.

Kolejny skrypt ponownie wykorzystuje WMI do uzyskania informacji o komputerze, określając jego rolę w sieci (czy jest on kontrolerem domeny, serwerem członkowskim, czy członkowską stacją roboczą). Trzeba będzie zastosować tu *Select Case* do przetworzenia wyników, otrzymanych z WMI, gdyż będą one w formie 0, 1, 2, 3, 4 lub 5. Sześć możliwości to nazbyt wiele dla konstrukcji *If… Then… ElseIf*:

```
Option Explicit
On Error Resume Next
Dim strComputer
```

```
Dim wmiRoot
Dim wmiQuery
Dim objWMIService
Dim colComputers
Dim objComputer
Dim strComputerRole
strComputer = "."
wmiRoot = "winmgmts:\\" & strComputer & "\root\cimv2"
wmiQuery = "Select DomainRole from Win32_ComputerSystem"
Set objWMIService = GetObject(wmiRoot)
Set colComputers = objWMIService.ExecQuery _
    (wmiQuery)
For Each objComputer In colComputers
    Select Case objComputer.DomainRole
        Case 0
            strComputerRole = "Standalone Workstation"
        Case 1
            strComputerRole = "Member Workstation"
        Case 2
            strComputerRole = "Standalone Server"
        Case 3
            strComputerRole = "Member Server"
        Case 4
            strComputerRole = "Backup Domain Controller"
        Case 5
            strComputerRole = "Primary Domain Controller"
    End Select
    WScript.Echo strComputerRole
Next
```

Informacja nagłówkowa

Część nagłówkowa skryptu computerRoles.vbs została wyodrębniona poniżej. Rozpoczyna się poleceniami *Option Explicit* i *On Error Resume Next*. Dalej znajdują się deklaracje siedmiu zmiennych, pochodzących z omawianego wcześniej skryptu CPUType.vbs. Zmienne *strComputer*, *wmiRoot* i *objWMIService* są dokładnie takie same jak w CPUType.vbs.

WmiQuery różni się jednak od nich. Znajduje się ona w części odniesieniowej, gdzie przypisujemy jej łańcuch zapytania WMI. Osobna deklaracja zmiennej umożliwia dokonanie zmian w zapytaniu WMI, bez konieczności przepisywania całego skryptu.

ObjWMIService przechowuje połączenie z WMI. Tymczasem nazwa *colComputers* jest tak naprawdę myląca. Brzmi ona, jakby zmienna przechowywała zbiór nazw komputerów lub obiektów, ale w rzeczywistości przechowuje zwracane role domen. Zmienna *StrComputerRole* przechowuje opis faktycznej roli komputera i jest wykorzystywana przez polecenie *WScript.Echo* do wyświetlania wyników pracy skryptu. *ObjComputer* zlicza wyniki.

```
Option Explicit
On Error Resume Next
Dim strComputer
Dim wmiRoot
Dim wmiQuery
Dim objWMIService
Dim colComputers
Dim objComputer
Dim strComputerRole
```

Część odniesieniowa

Część odniesieniowa przydziela wartości zmiennym zadeklarowanym w części nagłówkowej ComputerRoles.vbs. ComputerRoles.vbs można nazwać przyjaznym skryptem, gdyż większość jego elementów została odziedziczona z innych skryptów. Zmienne *StrComputer*, *wmiRoot* i *objWMIService* pochodzą ze skryptów omawianych wcześniej. Część odniesieniowa omawianego skryptu ma postać:

```
strComputer = "."
wmiRoot = "winmgmts:\\" & strComputer & "\root\cimv2"
wmiQuery = "Select DomainRole from Win32_ComputerSystem"
Set objWMIService = GetObject(wmiRoot)
Set colComputers = objWMIService.ExecQuery _
    (wmiQuery)
```

Dwie zmienne zostały wymyślone specjalnie na potrzeby tego skryptu. Są to *wmiQuery* oraz *colComputers*. Skrypt ListHardDrives miał w polecenie *GetObject* wbudowane zapytanie WMI. Wydostanie zapytania z polecenia *GetObject* i przypisanie go do zmiennej *wmiQuery* upraszcza skrypt, ułatwiając jego modyfikacje w przyszłości. Natomiast zmienna *colComputers* przypisana jest do wyników pracy zapytania WMI.

Krótko

P. Jak implementować *Select Case*?
O. Funkcja *Select Case* rozpoczyna się poleceniem *Select Case* oraz zmienną do sprawdzenia. Często poprzedzona jest wyrażeniem *For Each*.

P. W jaki sposób *Select Case* podejmuje decyzje?
O. Funkcja *Select Case* przetwarza wyniki zapytań.

P. Jakie są zalety przypisania zapytania WMI do zmiennej?
O. Umożliwa ono VBScript uzyskanie dodatkowych informacji z WMI.

Część wykonawcza i wyjściowa

WMI często zwraca informacje w postaci zbioru (omówione zostało to w rozdziale 2), który musi zostać przetworzony przez polecenie *For Each…Next*, aby uzyskać odpowiednie dane. Część wykonawcza skryptu ComputerRoles.vbs rozpoczyna się od *For Each*. Przedstawiony poniżej skrypt ukazuje sposób, w jaki *Select Case* sprawdza każdy obiekt zbioru *colComputers*. Polecenie *Select Case* rozpoczyna się wskazaniem źródła informacji, którym tutaj jest *DomainRole* zmiennej *objComputer*. Kod opisywanych części wygląda następująco:

```
For Each objComputer In colComputers
    Select Case objComputer.DomainRole
        Case 0
            strComputerRole = "Standalone Workstation"
        Case 1
            strComputerRole = "Member Workstation"
        Case 2
            strComputerRole = "Standalone Server"
        Case 3
            strComputerRole = "Member Server"
        Case 4
            strComputerRole = "Backup Domain Controller"
        Case 5
            strComputerRole = "Primary Domain Controller"
    End Select
    WScript.Echo strComputerRole
Next
```

Struktura pola *DomainRole* opisana jest w materiałach Platform SDK dla Microsoft Windows Server 2003. Ważne są opisy wartości, przedstawione w tabeli 3-1.

Tabela 3-1 WMI Domain Roles w ***Win32_ComputerSystem***

Wartość	Znaczenie
0	Samodzielna stacja robocza
1	Członkowska stacja robocza
2	Serwer samodzielny
3	Serwer członkowski
4	Zapasowy kontroler domeny
5	Główny kontroler domeny

Pierwsza linia polecenia *Select Case* zawiera kluczowe słowa *Select Case* i wskazuje na część połączenia, mającą dostęp do interesujących informacji. Kolejne wyrażenia występują w tej samej postaci:

```
Case 0
    strComputerRole = "Standalone Workstation"
```

Sprawdzany element ma postać wywodzącą się z części zawierającej *Select Case*. Kod *strComputerRole = "Standalone Workstation"* jest przypisaniem do nowej zmiennej. Zmienna *strComputerRole* zostaje wyświetlona, przedstawiając rolę komputera w domenie, po zakończeniu konstrukcji *Select Case* poleceniem *End Select*.

Konstrukcja *Select Case* kończy się słowami *End Select*, podobnie, jak dla wyrażenia *If… Then*, które kończy się słowami *End If*. Po *End Select* pojawia się polecenie *WScript.Echo*, przesyłające wartość *strComputerRole* użytkownikowi. Pamiętajmy, że skrypt ComputerRoles.vbs korzystał z konstrukcji *Select Case* do odkrycia i przypisania wartości DomainRole zmiennej *strComputerRole*. Gdy funkcja wypełni swoje zadanie, używając polecenia *Next* powracamy do pętli *For Each* z początku skryptu.

Podsumowanie

Rozdział 3 omawiał zastosowania konstrukcji decyzyjnych *If… Then* i *Select Case*. Pojawiły się w nim także wariacje podstawowego polecenia *If… Then*: *If… Then… Else* oraz *If… Then… ElseIf*. Wyrażenie *If… Then… ElseIf* pozwala na sprawdzenie trzech lub więcej sytuacji, ale potrafi też negatywnie wpływać na czytelność kodu i stwarzać problemy w jego interpretacji. Sytuacje sprawdzające ponad cztery parametry najlepiej obsługiwać funkcją *Select Case*. Pojawiło się także zagadnienie stałych wewnętrznych (*vbCrLf*), które mogą posłużyć do formatowania danych wyjściowych. Wykorzystywaliśmy też zmienne do podniesienia wydajności zapytań WMI i łańcuchów połączeń.

Quiz

P. Gdzie, w wyrażeniu *If...Then* powinien znaleźć się sprawdzany warunek?
O. Wewnątrz wyrażenia *If… Then*.

P. Która konstrukcja najlepiej posłuży do sprawdzenia dwóch warunków?
O. Najlepszym rozwiązaniem jest *If… Then… Else*.

P. Jak zamykamy konstrukcję *If...Then...Else*?
O. Słowami *End If*.

P. Czym jest stała wewnętrzna?
O. Stała wewnętrzna to stała wbudowana w VBScript, niewymagająca przypisania wartości przed jej wykorzystaniem.

P. Która konstrukcja najlepiej posłuży do sprawdzenia trzech warunków?
O. Możemy skorzystać z konstrukcji *If… Then… ElseIf* lub *Select Case*.

P. Dlaczego funkcja *Select Case* zazwyczaj najlepiej nadaje się do sprawdzania czterech lub więcej warunków instances?
O. Ponieważ zajmuje mało miejsca, a przy tym jest łatwa do odczytu i modyfikacji.

Do samodzielnego ćwiczenia

Ćwiczenie 5 Modyfikacja CPUType.vbs

Ćwiczenie polega na modyfikacji skryptu CPUType.vbs, aby zamiast kilku wyrażeń *If… Then …ElseIf* korzystał z pojedynczej funkcji *Select Case*.

Instrukcje

1. Otwieramy CPUType.vbs i zachowujemy go jako lab5.vbs.
2. Zmieniamy linię z poleceniem *On Error Resume Next* w komentarz.
3. Zamiast *If… Then* wpisujemy *Select Case*. Pozostawiamy bez zmian *objProcessor.Architecture*. Linia *Select Case* powinna wyglądać następująco:

```
Select Case objProcessor.Architecture
```

4. Skupiamy się na sprawdzaniu. Jeśli *objProcessor.Architecture = 0*, wiemy, że procesor należy do typu x86. Zatem pierwszym argumentem do sprawdzenia jest *Case 0*, który wpisujemy w następnej linii:

   ```
   Case 0
   ```

5. Nie ruszamy linii z poleceniem *WScript.Echo*.
6. *ElseIf objProcessor.Architecture = 1* zmieniamy na *Case 1*, co oznacza procesor typu MIPS. Usuwamy całą linię z *ElseIf* i umieszczamy zamiast niej Case 1.
7. Nie ruszamy linii z poleceniem *WScript.Echo*.

   ElseIf objProcessor.Architecture = 2 także zmieniamy na *Case 2*:

   ```
   Case 2
   ```

   Po wszystkich tych zmianach, konstrukcja *Select Case* powinna wyglądać:

   ```
   Select Case objProcessor.Architecture
   Case 0
       WScript.Echo "This is an x86 cpu."
   Case 1
       WScript.Echo "This is a MIPS cpu."
   Case 2
       WScript.Echo "This is an Alpha cpu."
   ```

8. *ElseIf objProcessor.Architecture = 3 Then* zmieniamy na *Case 3*.
9. Nie ruszamy linii z poleceniem *WScript.Echo*.

   Kolejny argument to Case 6, a nie Case 4, jak można by się spodziewać. Wynika to z modyfikacji linii „*ElseIf objProcessor.Architecture = 6 Then*". Kod ma teraz postać:

   ```
   Select Case objProcessor.Architecture
       Case 0
           WScript.Echo "This is an x86 cpu."
       Case 1
           WScript.Echo "This is a MIPS cpu."
   ```

```
Case 2
    WScript.Echo "This is an Alpha cpu."
Case 3
    WScript.Echo "This is a PowerPC cpu."
Case 6
    WScript.Echo "This is an ia64 cpu."
```

10. Ostatni argument *Case* ma zostać sprawdzony w ramach polecenia *Else*, które obsługuje niewypisane możliwości. Argument wprowadzamy przez zmianę słowa *Else* na *Case Else*.
11. Nie ruszamy linii z poleceniem *WScript.Echo* o treści *"Can not determine cpu type"*.
12. Zamiast *End If* piszemy *End Select*. To zakończyło konwersję *If…Then…ElseIf* na *Select Case*.
13. Zachowujemy plik i uruchamiamy skrypt.

Ćwiczenie 6: Modyfikacja ComputerRoles.vbs

Ćwiczenie polega na modyfikacji skryptu ComputerRoles.vbs, aby służył do aktywacji DHCP na stacjach roboczych.

Scenariusz

Pewna sieć została zoorganizowana przez kogoś, kto nie miał najmniejszego pojęcia o DHCP. Konsekwencją są statyczne adresy IP wszystkich stacji roboczych w sieci. Już przy stu stacjach roboczych dawało się to we znaki, a teraz sieć ma juz trzy razy więcej maszyn. Arkusz Microsoft Excel, służący do śledzenia odwzorowań pomiędzy nazwami komputerów a adresami IP nie działa już dłużej, jak należy, przysparzając więcej problemów niż korzyści. Co gorsza, paru użytkowników o wyższych uprawnieniach poczęło zmieniać ostatnie oktety we właściwościach TCP/IP, co przekreśla szansę na odzyskanie zarządzanej sieci. Naszym zadaniem jest napisanie skryptów, które:

- Wykorzystując WMI określą rolę komputera w sieci i wyświetlą jego nazwę, nazwę domeny (jeśli do niej należy) oraz użytkownika przypisanego do komputera.
- Wykorzystując WMI udostępnią DHCP na wszystkich adapterach sieciowych, zainstalowanych na komputerach z obsługą TCP/IP.

Do realizacji pierwszej części zadania wykorzystamy klasę WMI *Win32_ComputerSystem*.

Część A

1. Otwieramy plik ComputerRoles.vbs i zachowujemy go jako lab6a.vbs.
2. Zmieniamy linię, zawierającą polecnie *On Error Resume Next,* w komentarz.
3. Deklarujemy nowe zmienne poleceniem *Dim*:
   - *strcomputerName*
   - *strDomainName*
   - *strUserName*

4. Modyfikujemy *wmiQuery* tak, aby zwracała coś więcej niż *DomainRole Win32_ComputerSystem*. Podpowiedź: zamieniamy *DomainRole* na symbol *.

   ```
   wmiQuery = "Select DomainRole from Win32_ComputerSystem"
   ```

   Po wprowadzeniu zmian, linia wygląda następująco:

   ```
   "Select * from Win32_ComputerSystem"
   ```

5. Ponieważ *colComputers* jest zbiorem, nie możemy kierować do niego bezpośrednich zapytań. Będziemy musieli wykorzystać *For Each Next*, aby uzyskać odpowiedni punkt zaczepienia. Przypisanie nowych zmiennych do obiektów zostanie zrealizowane wewnątrz pętli *For Each Next*. Robimy to w pokazany sposób:
   - *strComputerName = objComputer.name*
   - *strDomainName = objComputer.Domain*
   - *strUserName = objComputer.UserName*
6. Po zakończeniu wyrażenia *Select Case* (*End Select*), ale przed poleceniem *Next* na końcu kodu, umieszczamy polecenie *WScript.Echo*, wyświetlające cztery elementy wymagane przez pierwszą część scenariusza. Korzystając z konkatenacji (znak &) umieszczamy cztery zmienne w jednej linii. Deklaracje tych czterech elementów mają postać:
   - *Dim strComputerRole*
   - *Dim strcomputerName*
   - *Dim strDomainName*
   - *Dim strUserName*
7. Zachowujemy plik i uruchamiamy go.
8. Modyfikujemy skrypt tak, aby każda zmienna była wyświetlona w osobnej linii. Podpowiedź: korzystamy ze stałej wewnętrznej *vbCrLf* i konkatenacji (&) do połączenia linii. Wygląda to tak:

```
strComputerRole & vbCrLf & strComputerName
```

9. Zachowujemy plik i uruchamiamy go.
10. Korzystamy z polecenia *WScript.Echo* do wyświetlenia komunikatu:

    ```
    WScript.Echo("all done")
    ```

11. Zachowujemy plik i uruchamiamy go.

Część B

12. Otwieramy skrypt ComputerRoles.vbs i zachowujemy go jako lab6b.vbs.
13. Zmieniamy linię, zawierającą polecnie *On Error Resume Next* w komentarz.
14. Deklarujemy nowe zmienne poleceniem *Dim*. Podpowiedź: możemy zmienić nazwę następujących elementów:
    - *Dim colComputers*
    - *Dim objComputer*
    - *Dim strComputerRole*

15. Nowe zmienne:
    - *colNetAdapters*
    - *objNetAdapter*
    - *DHCPEnabled*
16. Modyfikujemy *wmiQuery* do postaci:

```
wmiQuery = "Select * from Win32_NetworkAdapterConfiguration where
IPEnabled=TRUE"
```

17. Zmieniamy wyrażenie Set:

```
Set colComputers = objWMIService.ExecQuery (wmiQuery)
```

    Zamiast *colComputers* w wyrażeniu znajdzie się *colNetAdapters*:

```
Set colNetAdapters = objWMIService.ExecQuery (wmiQuery)
```

18. Usuwamy konstrukcję *Select Case*. Rozpoczyna się ona od następującej linii:

```
Select Case objComputer.DomainRole
```

    Kończy się słowami: *End Select*.

19. Powinniśmy otrzymać:

```
For Each objComputer In colComputers
    WScript.Echo strComputerRole
Next
```

20. Modyfikujemy linię, zawierającą *For Each*, do postaci:

```
For Each objNetAdapter In colNetAdapters
```

21. Przypisujemy *DHCPEnabled* do *objNetAdapter.EnableDHCP()*. Można to zrobić w sposób:

```
DHCPEnabled = objNetAdapter.EnableDHCP()
```

22. Korzystając z *If…Then…Else* oceniamy czy operacja zakończyła się pomyślnie.Wartość *DHCPEnabled* będzie wynosić 0, jeśli DHCP zostanie udostępnione. Efektem powinien być komunikat, wygenerowany przez polecenie *WScript.Echo*. Kod ma postać:

```
If DHCPEnabled = 0 Then
    WScript.Echo "DHCP has been enabled."
```

23. Jeśli wartość *DHCPEnabled* będzie inna niż 0, procedura nei zadziała. Musimy umieścić w kodzie także warunek *Else*:

```
Else
    WScript.Echo "DHCP could not be enabled."
End If
```

24. Skrypt kończymy poleceniem *Next*, które równocześnie dopełnia konstrukcję *If…Then…Next*. Nie trzeba umieszczać tu komunikatów zamykających, gdyż odpowiednie wiadomości zostały przygotowane już wcześniej, w związku z poleceniem *DHCPEnabled*.
25. Zachowujemy i uruchamiamy skrypt.

11 Wstęp do Active Directory Service Interfaces

Poniższy rozdział zawiera obraz koncepcji Active Directory Service Interfaces (ADSI). Składają się na nią dwa elementy: Active Directory oraz interfejsy usługi. Zrozumienie obydwu tych elementów jest konieczne do efektywnego korzystania z Microsoft Visual Basic Script (VBScript). Rozdział ten nie zawiera pełnego omówienia tematu, skupiając się na zastosowaniu ADSI do automatyzacji i planowania wykonywania rutynowych czynności.

Zanim rozpoczniesz

Do zrozumienia zawartości tego rozdziału wymagana jest wiedza z zakresu:

- Tworzenia tablicy
- Eksportu danych do plików tekstowych
- Odczytu informacji z plików tekstowych
- Implementacji konstrukcji *For…Next*
- Implementacji konstrukcji *Select Case*

Po ukończeniu tego rozdziału, Czytelnik będzie umiał:

- Nawiązać połączenie z Active Directory
- Pracować z przestrzeniami nazw Active Directory
- Tworzyć jednostki organizacyjne (OU) w Active Directory
- Utworzyć użytkownika w Active Directory
- Dysponował wiedzą na temat dostawców ADSI

Praca z ADSI

Poniższe punkty przedstawiają zastosowanie ADSI oraz VBScript do wykonywania podstawowych zadań administracji sieciowej:

- Import listy nazw i tworzenie kont użytkowników
- Import listy i zmiana haseł użytkowników
- Import listy i tworzenie jednostek organizacyjnych (OU) bezpośrednio po aktualizacji do Microsoft Windows Server 2003

- Odczyt katalogu Microsoft Exchange 5.x oraz ustawienie wyświetlanej nazwy w Active Directory z wartością z Exchange 5.x
- Odczyt katalogu Exchange 5.x dla domyślnego osobistego adresu SMTP i ustawienie go w Active Directory
- Odczyt nazwy komputera lub adresu IP oraz przypisanie użytkownikom lokalnych drukarek
- Tworzenie użytkownikom personalizowanych skrótów podczas logowania na podstawie przynależności do grupy
- Odwzorowanie napędów na podstawie przynależności do OU

Krok po kroku

▶ **Nawiązanie połączenia z Active Directory:**

1. Zaimplementować połączenie z Active Directory.
2. Skorzystać z odpowiedniego dostawcy.
3. Określić ścieżkę do odpowiedniego obiektu w Active Directory.
4. Za pomocą *SetInfo* zapisać zmiany w Active Directory.

Skrypt, CreateOU.vbs, wykonuje wymienione wyżej punkty. Skrypt ten został pozbawiony części nagłówkowej, aby skupić się przede wszystkim na działaniach mających na celu nawiązanie połączenia z Active Directory oraz utworzenia OU. CreateOU.vbs korzysta ze zmiennych w każdym z przedstawionych kroków.

```
provider = "LDAP://"
OU = "ou=hiring, ou=hr,"
domain = "dc=a,dc=com"
oClass = "organizationalUnit"
oOU = "ou="
oOUname = "myOU"

 Set objDomain = GetObject(provider & OU & domain)
Set objOU = objDomain.create(oClass, oOU & oOUname)

 objOU.SetInfo

 WScript.Echo("OU " & oOUname & "was created")
```

Część odniesieniowa

Część odniesieniowa skryptu konfiguruje połączenie z Active Directory oraz określa ścieżkę i cel operacji. Tymczasem decydujemy, z którego dostawcy należałoby skorzystać. Dobrze jest przyjrzeć się każdemu z nich przed podjęciem tej decyzji.

Dostawcy ADSI

Tabela 11-1 zawiera czterech dostawców, z których mogą korzystać użytkownicy ADSI. Połączenie z systemem Microsoft Windows NT 4 wymaga zastosowania specjalnego dostawcy *WinNT*. Podczas migracji Active Directory zazwyczaj przenosi się za pomocą osobnego skryptu użytkowników z domeny Windows NT 4 do Active Directory OU lub domeny Microsoft Windows Server 2003. Niekiedy (chociażby w przypadku dowolnych schematów nazewnictwa) napisanie skryptu jest o wiele prostsze niż korzystanie z Active Directory Migration Tool (ADMT).

Tabela 11-1 Dostawcy obsługiwani przez ADSI

Dostawca	Cel
WinNT:	Komunikacja z Primary Domain Controllers (Głównymi Kontrolerami Domeny, PDC) Windows NT 4.0 oraz Backup Domain Controllers (Zapasowymi Kontrolerami Domeny, BDC) i bazami danych z kont lokalnych Windows 2000 oraz nowocześniejszych stacji roboczych
LDAP:	Komunikacja z serwerami LDAP, w tym katalog Exchange 5.x oraz Active Directory Windows 2000
NDS:	Komunikacja z serwerami Novell Directory Services
NWCOMPAT:	Komunikacja z serwerami Novell NetWare

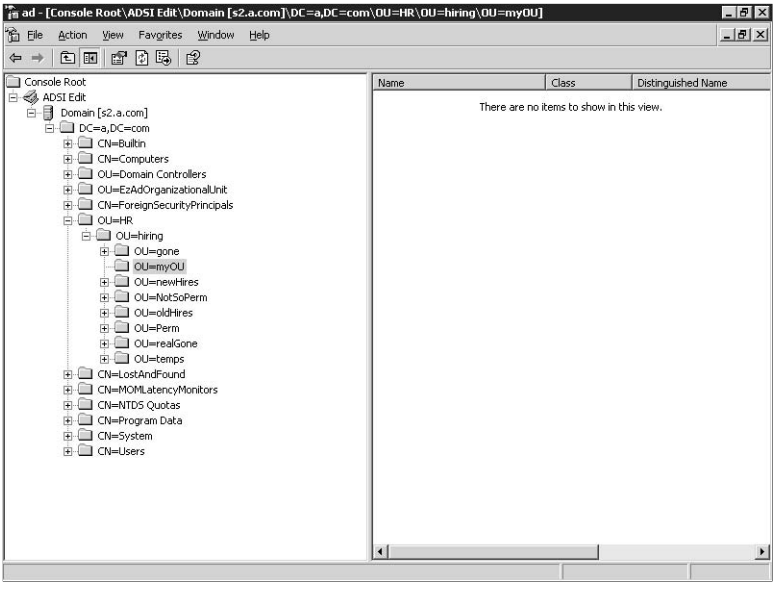
Nawiązywanie połączenia z maszyną Windows NT za pomocą ADSI bywa kłopotliwe. Nazwa dostawcy *WinNT* musi zostać dokładnie przepisana z tabeli 11-1. Nie możemy zmieniać wielkości liter. Wynika to z ograniczeń zwrotów Pascala, które częściowo składają się z małych a częściowo z dużych znaków, o czym nie wolno zapominać. Oznaką błędu w pisowni nie jest zawiadomienie o błędzie, ale nieudane wykonanie operacji łączenia.

Po wyborze dostawcy ADSI należy określić ścieżkę docelowego katalogu. Konieczna jest tu odrobina wiedzy o Active Directory, a konkretnie o hierarchicznej strukturze przestrzeni nazw. Nawiązywanie połączenia z dostawcą usług LDAP (Lightweight Directory Access Protocol) wiąże się ze wskazaniem celu połączenia w hierarchii bazy danych LDAP, ponieważ hierarchia ta jest powiązana jedynie z bazą, a nie protokołem czy dostawcą. Na przykład w skrypcie CreateOU.vbs tworzymy OU, która znajduje się w dzierżawionej OU, znajdującej się w OU HR. Brzmi to zawile, ale wystarczy wiedzieć, że OU HR zawarta jest w domenie o nazwie *a.com*. Stąd wniosek, iż znajomość i zrozumienie hierarchii, z którą się pracuje, jest konieczne. Narzędziem, które może w tym pomóc, jest ADSI Edit.

Uwaga Prawdopodobnie najtrudniejsze w korzystaniu z ADSI jest ustalenie nazw w katalogu. Wynika to z braku powiązań z wyświetlanymi nazwami, które można przejrzeć w narzędziach takich jak Active Directory Users and Computers. Przykłady znajdują się w Dodatku B „Dokumentacja ADSI".

ADSI Edit znajduje się na dysku Windows Server 2003, w katalogu support\tools. Instalacja następuje po kliknięciu na Suptools.msi oraz po wcześniejszym zamknięciu wszystkich programów. Trwa to parę minut i nie ma konieczności restartowania komputera. Po instalacji

otwieramy czystą konsolę MMC i dodajemy moduł dodatkowy ADSI Edit. Po instalacji modułu, klikamy prawym przyciskiem myszy ikonę ADSI Edit, zlecamy nawiązanie połączenia i wybieramy z rozwijanej listy domenę w sposób przedstawiony na rysunku 11-1.



Rysunek 11-1 Poznajemy hierarchię lasu, aby zapobiec jakimkolwiek pomyłkom w ścieżce skryptu.

Nazwy LDAP

Podczas określania nazw OU i domeny mamy do czynienia z konwencją nazewnictwa LDAP, gdzie przestrzeń nazw figuruje jako zbiór elementów nazw, zwanych *relatywnymi wyszczególnionymi nazwami* (*relative distinguished names*, RDN). Przypisywanie wartości RDN polega na zastosowaniu znaku równości. Składając razem wszystkie RDN oraz części z nazw stojących wyżej w hierarchii, aż do korzenia, otrzymamy pojedynczą, wyszczególnioną unikatową nazwę globalną.

RDN zazwyczaj składa się z typu atrybutu, znaku równości oraz wartości łańcuchowej. Część atrybutów, związanych z Active Directory, przedstawiono w tabeli 11-2.

Tabela 11-2 Najczęstsze typy atrybutów RDN

Atrybut	Opis
DC	Komponent Domeny
CN	Nazwa Publiczna
OU	Jednostka Organizacyjna
O	Nazwa Organizacji
Street	Adres Ulicy

Tabela 11-2 Najczęstsze typy atrybutów RDN

Atrybut	Opis
C	Nazwa Kraju
UID	ID Użytkownika

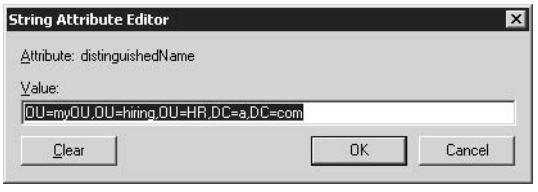
Część wykonawcza

Część wykonawcza skryptu zawiera dwie linie kodu: pierwsza dokonuje łączenia a druga tworzy OU. Wykonanie tych czynności wymaga skonstruowania wyszczególnionej nazwy. Gdy już ją będziemy mieli, OU zostanie utworzony zaraz po nawiązaniu połączenia z odpowiednim poziomem Active Directory.

W skrypcie CreateOU.vbs wyszczególniona nazwa jest konkatenacją dwóch osobnych zmiennych:

```
OU = "ou=hiring, ou=hr,"
domain = "dc=a,dc=com"
```

Werifikacja poprawności połączenia do OU może nastąpić za pośrednictwem ADSI Edit. Wystarczy kliknąć prawym przyciskiem odpowiednią OU, otworzyć właściwości i wybrać Distinguished Name z listy dostępnych właściwości. Pojawi się wówczas okno dialogowe, przedstawione na rysunku 11-2.



Rysunek 11-2 Wykorzystanie edytora atrybutów łańcuchów w ADSI Edit do szybkiej weryfikacji wybranej nazwy potencjalnego celu dla skryptu ADSI

Kolejna linia w części odniesieniowej określa klasę obiektu, z którym pracujemy. Gdy korzystamy z metody *Create,* musimy określić jaki typ obiektu tworzymy. W CreateOU.vbs interesujący nas kod ma następującą postać:

```
oClass = "organizationalUnit"
```

IADsContainer

Skrypt zawiera metodę *Create* dobrze znanego interfejsu *IADsContainer*. Umożliwia poziomowi ADSI tworzenie, usuwanie i zarządzanie obiektami ADSI. Wszystkie obiekty o określonym poziomie Active Directory korzystają z *IADsContainer*. *IADsContainer* obsługuje pięć metod, wypisanych w tabeli 11-3, które mogą zostać zastosowane na obiekcie ADSI w Active Directory. Będziemy z nich korzystali w dalszych skryptach.

Tabela 11-3 Metody *IADsContainer*

Metoda	Znaczenie
GetObject	Łączy element katalogu z wybraną *ADsPath* i przypisuje je do nazwanej zmiennej.
Create	Tworzy na danym poziomie nowy obiekt wybranej klasy.
Delete	Usuwa obiekt wybranej klasy z danego poziomu.
CopyHere	Tworzy kopię obiektu z wybraną *ADsPath* na danym poziomie. Pamiętajmy, że obiekt musi znajdować się w tej samej przestrzeni nazw katalogu. Na przykład niemożliwe będzie skopiowanie obiektu z przestrzeni *LDAP:* do *WinNT:*.
MoveHere	Przenosi obiekt z wybraną *ADsPath* do danego poziomu z poprzedniej lokacji. Do tej metody odnoszą się te same ograniczenia przestrzeni nazw, co do metody *CopyHere*.

W skrypcie CreateOU.vbs implementujemy metodę *IADsContainer Create* do utworzenia OU. Korzystamy przy tym z dwóch zmiennych. Pierwsza z nich to *oOU*, przechowująca klasę obiektu, który chcemy utworzyć. Przypisujemy jej wartość *OU*. Drugą zmienną jest *oOUname*, która przechowuje nazwę OU. Służy do utrzymania połączenia z metodą *Create* po implementacji połączenia za pomocą polecenia *Set*:

```
Set objOU = objDomain.create(oClass, oOU & oOUname)
```

Łączenie

Nim uczynimy cokolwiek z ADSI, musimy nawiązać połączenie z obiektem w Active Directory, co nazywane jest procesem *łączenia*. *Łańcuch łączenia* umożliwia wykorzystanie różnych elementów ADSI, w tym metod i właściwości. Cel zaproponowanej czynności zostaje określony jako komputer, kontroler domeny, użytkownik lub inny element, pozostający w strukturze katalogowej. Łańcuch łączenia składa się z pięciu części. Każdą z nich przedstawiono w łańcuchu łączenia ze skryptu CreateOU.vbs:

Słowo kluczowe	Zmienna	Polecenie	Dostawca	ADsPath
Set	*objDomain*	*GetObject*	*LDAP://*	*OU=hiring, OU=hr, dc=a, dc=com*

Uwaga Po nawiązaniu połączeń i utworzeniu elementów należy upewnić się, czy zmiany te zostały zatwierdzone w Active Directory. Zmiany w Active Directory są zazwyczaj tymczasowe, należy więc zadbać o ich zatwierdzenie. Należy skorzystać z metody *SetInfo,* w sposób zastosowany w skrypcie CreateOU.vbs: *objOU.SetInfo*.

Część wyjściowa

Skrypt nie ma domyślnie żadnych danych wyjściowych, jednak zawiera pojedyncze polecenie *WScript.Echo*, które wyświetla nazwę utworzonego poziomu. Ponieważ OU przechowywane jest w zmiennej *oOUname*, polecenie wyświetla tak naprawdę zawartość zmiennej:

```
WScript.Echo("OU " & oOUname & "was created")
```

Krótko

P. Jak nazywa się proces nawiązywania połączenia z Active Directory?
O. Proces łączenia.

P. Jaką nazwę nosi cel operacji ADSI?
O. *ADsPath*.

P. Nazwa LDAP składa się z kilku części. Jak nazywają się te części, oddzielone od siebie przecinkami?
O. Jest to relatywna wyszczególniona nazwa.

Tworzenie użytkowników

Jedną z możliwości ADSI jest tworzenie użytkowników. Utworzenie pojedynczego użytkownika za pomocą GUI nie stwarza problemów, ale utworzenie tuzina jest już niełatwe. Ponadto ze względu na podobieństwo skryptów ADSI, usunięcie tuzina lub więcej użytkowników jest tak proste, jak ich utworzenie. Ponieważ możemy korzystać z tego samego wejściowego pliku tekstowego w różnych skryptach, ADSI doskonale nadaje się do tworzenia tymczasowych kont.

Krok po kroku

▶ **Tworzenie użytkowników:**

1. Wybrać odpowiedniego dostawcę.
2. Nawiązać połączenie z danym poziomem dla użytkownika.
3. Określić domenę.
4. Określić klasę *User* obiektu.
5. Nawiązać połączenie z Active Directory.
6. Za pomocą metody *Create* utworzyć użytkownika.
7. Używając metody *Put* określić właściwość *sAMAccountName*.
8. Stosując *SetInfo* przypisać użytkownika do Active Directory.

Poniższy skrypt CreateUser.vbs przypomina skrypt CreateOU.vbs. Tak naprawdę CreateUser.vbs jest pochodną CreateOU.vbs, więc dokładna analiza skryptu nie jest konieczna. Jedyną różnicą jest fakt, że *oClass* równe jest klasie „*User"*, a nie „*organizationalUnit"*.

```
provider = "LDAP://"
OU = "ou=hiring, ou=hr,"
domain = "dc=a,dc=com"
oClass = "User"
oCN = "CN="
oUname = "myuser"

 Set objDomain = GetObject(provider & OU & domain)
Set objUser = objDomain.create(oClass, oCN & oUname)
objUser.Put "sAMAccountName", oUname
objUser.SetInfo

 WScript.Echo("User " & oUname & "was created")
```

Część odniesieniowa

Część odniesieniowa zawiera przypisania wartości zmiennym, które są deklarowane w skryptach tego typu. Tutaj dostawcą jest *LDAP://*. Nie wolno zapominać, że nazwa dostawcy rozróżnia wielkie i małe litery, a cała nazwa napisana jest w tych pierwszych. Następnie określamy OU, którą wykorzystamy w części *ADsPath* łańcucha łączenia. Celem będzie OU o nazwie *hiring*, znajdująca się w OU *hr*. Nazwa domeny składa się z dwóch *komponentów domeny* lub DC oddzielonych przecinkami. Nazwa domeny to *a.com*, a więc pierwszy komponent to *dc=a*, a druga *dc=com*.

Podczas tworzenia kont użytkowników musimy określić klasę użytkownika. Nazwa użytkownika oddzielona jest przedrostkiem „*cn="*. Z tabeli 11-2 wiadomo, że *cn* oznacza *common name*. Należy określić właściwość nazwy publicznej dla obiektu użytkownika.

Do zalogowania się w sieci potrzebna będzie użytkownikowi co najmniej *sAMAccountName*. Może być ona taka sama jak właściwość nazwy publicznej i często tak właśnie jest. Reszta pozostaje według ustaleń domyślnych, w tym blokada konta. Podczas ćwiczeń zostanie jednak utworzone bardziej rozbudowane konto użytkownika.

Część wykonawcza

W części wykonawczej skrypt zaczyna odbiegać od schematu z poprzednich przykładów, zawiera ona cztery linie kodu:

```
Set objDomain = GetObject(provider & OU & domain)
Set objUser = objDomain.create(oClass, oCN & oUname)
objUser.Put "sAMAccountName", oUname
objUser.SetInfo
```

Łączenie z ADSI jest analogiczne do procesu z poprzedniego skryptu. Korzystamy nawet z tej samej nazwy zmiennej. W kolejnej linii, gdy wywołujemy metodę *Create,* stosujemy jednak inne zmienne, gdyż zamiast *OU* tworzymy użytkownika. Klasa *oClass* jest równoważna *User*,

oCN jest równoważne „*CN=*", a *oUname* przechowuje wartość użytkownika, który ma zostać utworzony.

Wykonujemy teraz metodę *Put*, aby określić właściwość *sAMAccountName*. Korzystamy tu z nazwy *CN oUname* i stosujemy tę zmienną przy *sAMAccountName*. Po zakończeniu pracy wywołujemy *SetInfo* i zapisujemy dane w Active Directory.

Część wyjściowa

Po utworzeniu użytkownika należy zasygnalizować koniec procesu. Według schematu z poprzedniego skryptu korzystamy z *WScript.Echo* do wyświetlenia zmiennej *oUname* z wiadomością o utworzeniu użytkownika.

Krótko

P. Jaką klasę należy określić, aby utworzyć użytkownika?
O. Klasę *User*.

P. Do czego służy metoda *Put*?
O. Do zapisu dodatkowych danych właściwości obiektu, z którym jest związana.

Podsumowanie

Rozdział ten zawiera opis wykorzystania ADSI do nawiązywania połączeń z Active Directory i tworzenia OU oraz użytkowników. Omówiono też proces łączenia z Active Directory i komponenty, które składają się na konstrukcję łącza. Przedstawieni zostali różni dostawcy, z których można korzystać w pracy z ADSI z podkreśleniem faktu, że w nazwach i i zapytaniach rozróżniana jest wielkość liter. Omówiono także metodę *Create*, służącą do tworzenia obiektów w Active Directory oraz proces tworzenia użytkowników i OU. Pokazano również dodatkowe właściwości dostępne obiektowi *User* oraz zastosowanie metody *Put* do zmiany wartości różnych cech obiektu *User*. Wskazano też, jak korzystać z polecenia *setInfo* do zapisu danych w Active Directory.

Quiz

P. Do czego służy dostawca ADSI?
O. Zawiera schemat katalogu, z którym się porozumiewa i który ukrywa przed użytkownikiem. Każdy dostawca jest wykorzystywany w jednolity dla wszystkich sposób, aby wprowadzić pewien standard.

P. Wymień czterech dostawców ADSI.
O. Są to *LDAP*, *WinNT*, *NDS* oraz *NWCOMPAT*.

P. Co oznacza atrybut relatywnej wyszczególnionej nazwy LDAP o oznaczeniach *CN*?
O. *Common name*, czyli nazwę publiczną.

P. W jaki sposób wykonywana jest metoda *IADsContainer* o nazwie *GetObject*?
O. Łączy ona obiekt katalogu z określoną ADsPath i przypisuje je nazwanej zmiennej.

Do samodzielnego ćwiczenia

Ćwiczenie 22: Tworzenie OU

Ćwiczenie to polega na utworzeniu OU. Rezultatem pracy będzie podprocedura, która będzie mogła znaleźć zastosowanie w innym skrypcie.

Instrukcje

1. Otwieramy Notatnik.
2. W pierwszej linii umieszczamy **Option Explicit**.
3. Deklarujemy zmienne: *provider*, *domain*, *oClass*, *oOU*, *objDomain*, *objOU*, *oOUname* oraz *oDescription*.
4. Przypisujemy dostawcę LDAP do zmiennej *provider*:

   ```
   provider = "LDAP://"
   ```

5. Przypisujemy zmienną *domain* do domeny udostępnionej przez sieć, na przykład *a.dom*. Dzielimy każdą część nazwy domeny na komponenty:

   ```
   domain = "dc=a,dc=com"
   ```

6. Przypisujemy zmienną do klasy *organizationalUnit*. Należy pamiętać o zawarciu nazwy klasy w cudzysłowiu:

   ```
   oClass = "organizationalUnit"
   ```

7. Przypisujemy wartość zmiennej, przechowującej nazwę *OU*. Tutaj zmienną tą będzie *oOUname*, a wartością *Lab22*:

   ```
   oOUname = "Lab22"
   ```

8. Przypisujemy odpowiedni opis zmiennej *oDescription*:

   ```
   oDescription = "For Lab 22 Use"
   ```

9. Za pomocą polecenia *Set* przypisujemy zmienną *objDomain* do rezultatu działania funkcji *GetObject* podczas pracy ze zmienną dostawcy i zmienną domeny:

   ```
   Set objDomain = GetObject(provider & domain)
   ```

10. Używając polecenia *Set* przypisujemy zmienną *objOU* do rezultatu działania metody *Create* podczas pracy ze zmiennymi *oClass*, *oOU* i *oOUname*:

    ```
    Set objOU = objDomain.create(oClass, oOU & oOUname)
    ```

11. Stosując metodę *Put* umieszczamy dane ze zmiennej *oDescription* w polu *Description*. Rozdzielamy zmienną i nazwę pola przecinkiem:

    ```
    objOU.Put "description", oDescription
    ```

12. Metodą *SetInfo* dokonujemy zmian w Active Directory:

    ```
    objOU.SetInfo
    ```

13. Skrypt kończymy poleceniem *WScript.Echo*, wyświetlającym nazwę *oOUname* oraz odpowiedni opis podjętych czynności:

```
WScript.Echo("OU " & oOUname & "was created")
```

14. Zachowujemy skrypt pod nazwą **Lab22Solution.vbs**.
15. Uruchamiamy skrypt. Nie ma tu znaczenia, czy zostanie uruchomiony jako CScript, czy WScript.
16. Otwieramy Active Directory Users and Computers, aby zweryfikować obecność OU Lab22.
17. Klikamy prawym klawiszem myszy na OU Lab22 i wybieramy właściwości z menu. W polu General sprawdzamy, czy opis przypisany w punkcie 11 znajduje się w polu opisowym.
18. Zamykamy wszystko. Nie usuwamy OU Lab22, ponieważ będzie potrzebna w kolejnym ćwiczeniu.

Ćwiczenie 23: Tworzenie wielowartościowych użytkowników

To ćwiczenie polega na tworzeniu użytkowników. Utworzonego użytkownika umieścimy w OU utworzonej w ćwiczeniu 22. Efektem pracy jest podprocedura, którą będzie można zastosować w innych skryptach, gdzie korzystamy z *Users*.

Instrukcje

1. Otwieramy Notatnik.
2. W pierwszej linii umieszczamy **Option Explicit**.
3. Deklarujemy zmienne: *provider*, *ou*, *domain*, *oClass*, *oCN*, *objDomain*, *objUser*, *oUname* oraz *oDescription*.
4. Przypisujemy dostawcę LDAP do zmiennej *provider*:

```
provider = "LDAP://"
```

5. Przypisujemy OU Lab22 do zmiennej *OU*:

```
OU = "ou=lab22,"
```

6. Przypisujemy domenę z punktu 5 ćwiczenia 22 do zmiennej *domain*. Powinna ona znajdować się w sieci lokalnej i być tą samą domeną, którą utworzyliśmy w OU Lab22:

```
domain = "dc=a,dc=com"
```

7. Przypisujemy klasę *User* do zmiennej *oClass*:

```
oClass = "User"
```

8. Przypisujemy wartość „*CN=*" do zmiennej *oCN*:

```
oCN = "CN="
```

9. Przypisujemy zmiennej *oUname* nazwę utworzonego użytkownika. Niech bedzie to *labUser*:

```
oUname = "labUser"
```

10. Przypisujemy nowemu użytkownikowi odpowiedni opis. Polega to na przypisaniu zmiennej *Description* wartości:

```
oDescription = "created for lab22 use"
```

11. Korzystając z polecenia *Set* przypisujemy zmiennej *objDomain* rezultat działania funkcji *GetObject* podczas pracy ze zmienną dostawcy, zmienną *OU* oraz zmienną *domain*:

```
Set objDomain = GetObject(provider & OU & domain)
```

12. Za pomocą polecenia *Set* przypisujemy zmiennej *objUser* rezultat działania metody *Create* podczas pracy ze zmiennymi *oClass*, *oCN* i *oUname*:

```
Set objUser = objDomain.Create(oClass, oCN & oUname)
```

13. Używając metody *Put* umieszczamy dane ze zmiennej *oUname* w polu *sAMAccountName*. Oddzielamy zmienną od nazwy pola przecinkiem:

```
objUser.Put "sAMAccountName", oUname
```

14. Stosując metodę *Put* umieszczamy dane ze zmiennej *oUname* w polu *DisplayName*. Oddzielamy zmienną od nazwy pola przecinkiem:

```
objUser.Put "DisplayName", oUname
```

15. Korzystając z metody *Put* umieszczamy dane ze zmiennej *oDescription* w polu *escription*. Oddzielamy zmienną od nazwy pola przecinkiem:

```
objUser.Put "description", oDescription
```

16. Za pomocą metody *SetInfo* dokonujemy zmian w Active Directory:

```
objUser.SetInfo
```

17. Kończymy skrypt poleceniem *WScript.Echo*, wyświetlającym nazwę *oUname* oraz odpowiedni opis podjętych czynności:

```
WScript.Echo("User " & oUname & "was created")
```

18. Zachowujemy skrypt pod nazwą **Lab23Solution.vbs**.
19. Uruchamiamy skrypt. Nie ma tu znaczenia, czy zostanie on uruchomiony pod CScript, czy WScript.
20. Otwieramy Active Directory Users and Computers, aby zweryfikować obecność nowego użytkownika. Powinien znajdować się w OU Lab22.
21. Klikamy prawym klawiszem myszy na nowego użytkownika i wybieramy właściwości z menu. W polu General sprawdzamy, czy przypisana wcześniej nazwa oraz opis znajdują się na swoim miejscu.
22. Zamykamy wszystko.